Graph neural networks-based Scheduler for Production planning problems using Reinforcement Learning

Preprint · June 2023

DOI: 10.13140/RG.2.2.13862.96326

Mohammed Sharafath Abdul Hameed
South Westphalia University of Applied Sciences
11 PUBLICATIONS **52** CITATIONS
SEE PROFILE

Andreas Schwung
South Westphalia University of Applied Sciences
185 PUBLICATIONS **1,144** CITATIONS
SEE PROFILE

Reinforcement Learning on Job Shop Scheduling Problems Using Graph Networks

Mohammed Sharafath Abdul Hameed *, Andreas Schwung†
Department of Automation Technology
South Westphalia University of Applied Sciences
Soest, Germany
Email: * sharafath.mohammed@fh-swf.de , †schwung.andreas@fh-swf.de,

***Abstract*—This paper presents a novel approach for job shop scheduling problems using deep reinforcement learning. To account for the complexity of production environment, we employ graph neural networks to model the various relations within production environments. Furthermore, we cast the JSSP as a distributed optimization problem in which learning agents are individually assigned to resources which allows for higher flexibility with respect to changing production environments. The proposed distributed RL agents used to optimize production schedules for single resources are running together with a co-simulation framework of the production environment to obtain the required amount of data. The approach is applied to a multi-robot environment and a complex production scheduling benchmark environment. The initial results underline the applicability and performance of the proposed method.**

***Index Terms*—Job Shop Scheduling, Reinforcement Learning, Graph Neural Networks, Distributed Optimization**

I. Introduction

THE task of manufacturing or production scheduling receives increasing attention from manufacturing enterprises to increase profitability and productivity on the shop floor, especially in a globally competitive market. Job shop scheduling problems (JSSP) are concerned with the allocation of tasks over time to a limited number of resources in such a way that one or more objectives for e.g., lower production costs, shorter processing time, etc. are optimized. It derives the most appropriate time to execute each operation by taking into account temporal relations between production processes and constraints of the shared manufacturing resources. JSSP are in general NP-hard.

The majority of approaches for JSSP are implemented as centralized algorithms which have full knowledge of the production process. Various approaches ranging from simple heuristics [1], constraint propagation and satisfaction [2], petri-nets [3] to neural networks [4], expert systems [5] and various meta-heuristics like genetic algorithms [6] or simulated annealing [7] have been developed. However, such centralized approaches have some inherent drawbacks. They rely on a single computing unit which provides difficulty in large scale environments as the problem size in general increases exponentially. Furthermore, they have to repeat their calculation if information used for the optimization or the production environment itself changes. The above weaknesses require additional approaches which are particularly suitable to tackle the complexity of large scale environments and the variability of production. For the first aspect, graph based representation can provide a way to handle the complexity by abstracting away low level relations. The second aspect requires for reactive job shop scheduling strategies which are easier to be accomplished by distributed scheduling algorithms. Such distributed algorithms [8] are most often implemented in form of multi-agent systems [9] where individual agents optimize their performance based on local information only. In this way, the global scheduling problem is distributed to considerably smaller local problems which makes the problem more scalable and adjustable as agents can be added or removed as required. However, most of the proposed agent architectures are hard coded which does not allow the agents to learn within the environment and hence, the agents cannot fully use experiences received during previous scheduling operations.

In this paper we present a novel approach for JSSP which combines the strength of learning-based agents with a distributed learning framework and a novel inter-agent information exchange mechanism. The resulting distributed reinforcement learning (RL) agents allow for a scalable architecture and high adaptability in large scale and changing production environments. RL has recently shown considerable success for optimization and learning various areas including games [10], robotics [11] and continuous control tasks [12]. This success can particularly be attributed to the improved expressivity and learning performance of neural networks used as function approximators in RL. To tackle the complexity of production environments, we propose the use of Graph Neural Networks (GNNs) which are used to represent the various relations within the different resources and orders. Particularly, we propose a novel framework to cast relevant features of production environments into a graph neural network architecture which is then used to optimize the production schedule in a reactive setting. We apply the proposed approach to a multi robot manufacturing station as well as to a complex benchmark problem in JSSP with very encouraging results.

The contributions of the work can be summarized as follows:

- We present a novel data based approach for reactive job shop scheduling problems by means of distributed RL

where individual resources are equipped with learning agents.
- We set-up a novel framework to model the interrelation within production environment using a graph neural network with message passing. The integration with the distributed RL allows for an end-to-end trainable representation of the production scheduling problem.
- We extend the approach by means of curiosity driven exploration and a gradient monitored regularizer which considerably reduces the complexity of the reward and network design.
- We employ the approach to two application scenarios, namely a multi robot coordination problem in manufacturing cells and a complex benchmark JSSP.

The paper is organized as follows. Related work is presented in Section II. In Section III the GNN-framework for the JSSP is introduced. Section III presents the distributed RL algorithms for JSSP using GNN and limited communication between the agents. Section V presents the results and experiments obtained for two application examples while Section VI gives conclusions and outlooks future work.

II. Related Work

We first discuss the related work on general JSSP approaches, the RL is discussed along with distributed RL, and and finally we focus on the graph neural networks.

Job Shop Scheduling: Scheduling problems are known to be NP-hard [13], meaning it is considered that they cannot be solved in polynomial time. Traditionally many algorithms are used to solve the scheduling problem, like genetics algorithm [6], integer programming [14] [15], meta-heuristic algorithm [16] etc. Our approach applies RL to the scheduling problem.

Reinforcement Learning: RL has achieved super-human performance in recent years in playing board games such as Go, Chess [17] [12] [10] [18] and robotics [11]. Not only is RL capable of learning and performing a specific task well, but it can also generalize well [19] and has proven to be capable of multi-task learning [20]. The advancements in the capabilities of deep neural networks have also helped push the boundaries of application of RL [21].

The developments in reinforcement learning provides an alternative to solving scheduling problems. [22] was one of the first works to use RL to solve a static scheduling problem. Further [23] used a model-free multi-agent approach to develop a adaptive reactive scheduling agent. [24] uses a multi agent DQN while [25] uses a model-based RL.

Distributed RL: The transition from single- to multi-agent RL is non-trivial in general as thoroughly discussed in [26]. Specifically, as the individual agents change their behavior during learning, the environment of each agent becomes non-markovian resulting in a partially observable MDP. Early approaches use cooperative settings based on Q-learning assuming that the actions of other agents are made to improve the collective reward [27], [28]. In recent years, various approaches based on deep MARL have been introduced. In [29], [30], deep MARL with centralized critic is proposed. Although, no communication is required during operation, extensive communication between the agents is required during training. In [31], [32], a common reward function for all agents is applied which also requires constant inter-agent communication between all agents and restricts the rewards considerably. Fully independent RL-agents are used in [33] where a deep Q-network (DQN) is employed in a multi-agent setting however performing poorly in partially observable environments. In our work, we rely on fully distributed RL-agents based on ACRL with limited communication between the agents.

Graph Neural Networks: Deep learning methods do not fare well on data in the non-euclidean space. All the above mentioned RL advancements either use convolutional neural networks [34] or a Multi-Layer Perceptron (MLP), which use feature engineering to achieve performance. The problem with this approach is that they are not size or permutation invariant. Addition of a single element to the scheduling environment means training the entire agent from scratch since the previously trained notions are not valid anymore. GNNs are size and permutation invariant and additionally they also provide information on pair-wise interaction between the objects [35]. [36] developed an input size invariant network for image classification, while [37] developed an algorithm that is generally applicable to estimation, classification, and outlier detection problems. Although there are many variants of the GNNs like attention [38], gated convolutions [39], representation learning [40], at the time of writing this paper, there is only one paper focusing on using GNNs on reinforcement learning in scheduling problems [41]. It focuses on the training performance rather than generalization performance. Our algorithm focuses on the generalization and scalability of the trained agents.

III. Graph Neural Networks for Production Scheduling

A. JSSP

The goal of a scheduling algorithm is to use a limited number of resources or machines 'm', to process a specified number of jobs or tasks 'n',while optimizing an specified objective like make-span 'C_{max}'. Each of these 'n' jobs have a specified operation sequence 'o', through the machines with a specified processing time 't' at the corresponding machine. When the job has passed through the last operation sequence it is considered finished. It should be noted that make-span is just one of the many objectives that can be optimized. Other objectives include, reducing tardiness, reducing due date violations, etc,. The scheduling problem also has other constraints that needs to taken care of, like completion of preceding task, limit on the number of jobs processed at any machine, usually one.

The JSSP can be represented as a feature vector that describes the complete system which is used to take actions by the Reinforcement Learning agent. While it provides a good local representation of the current system for the agent, it fails to capture the relationship between the other entities in the JSSP, and the impact these entities have on the agent. Although it could be argued that this can be learned inside the MLP, MLPs provide only a weak implicit relational inductive bias after training. Graph Neural Network provides for a direct representation of JSSP, the machines and the jobs can be represented as the nodes, while the connection between the machines is given by the edges. It becomes apparent that for a JSSP problem Graph Neural Networks provide a more suitable alternative, where the relational inductive bias is explicitly structured [35].

B. Graph Neural network

Deep learning revolutionized many machine learning tasks like object detection, image classification, image segmentation, machine translation, etc. which are all represented in the euclidean space. But typical graph structured data are represented in the non-euclidean space. Hence it requires special algorithms called Graph Neural Networks. The core idea in graph neural network is that the constraints provided in graph structured database is not effectively used by the existing neural networks. They provide a weak implicit inductive bias between the inputs and the outputs. The graph can be defined as a tuple $G = (V, E)$, where V is the vertices or nodes present in the graph and E is the edges. In certain applications a third term U, representing the global attributes in used to define the graph. Here V represents the set of node features of cardinality N^v, given by $\{v_i\}_{i=1:N^v}$, while E represents the set of edges of cardinality N^e, given by $\{(e_k, r_k, s_k)\}_{k=1:N^e}$. r_k is the receiver node and s_k is the sender node for the edge k.

Graph neural networks use a feature called message passing. There are two phases in the forward pass of message passing: a message passing phase and a readout phase [42]. The message passing algorithm is run for t time steps, and is as shown in equation 2.

$$h_i^0 := v_i,\ for\, i\, \epsilon\, N^v \tag{1}$$

$$m_i^{t+1} = \sum_{j \epsilon N^e} M_t(h_j^t, e_{i,j}) \tag{2}$$

$$h_i^{t+1} = R_t(h_i^t, m_i^{t+1}) \tag{3}$$

Here h_i^{t+1} message extracted feature vector of node v_i at time step t. The message passing phase is when the neighboring nodes h_j^t are concatenated with the edge feature $e_{i,j}$ and are sent into a differentiable function M_t, here an MLP is used. This message passed neighbors are then aggregated using the aggregator sum, as shown in equation 2. For the aggregator the other choices include $mean$, $average$ etc. After the message is aggregated, the extracted and aggregated message is concatenated with the feature vector of the node h_i^t and the sent in to another differentiable function R_t, as shown in equation 3. R_t can MLP or RNN based on the requirement.

$$s_t = flatten(h_i^{t+1}),\ for\, i\, \epsilon\, N^v \tag{4}$$

This process is repeated for T time-steps. T is a hyper-parameter that depends on how deep the graph is. For a shallow graph T should be ideally high, while for a deep graph T should be low. The feature extracted graph information is then flattened, as shown in 4 and then used by the RL agent to take action on the graph. We use python library $Pytorch$ for deep learning/RL and $Pytorch-geometric$ for graph neural net processing.

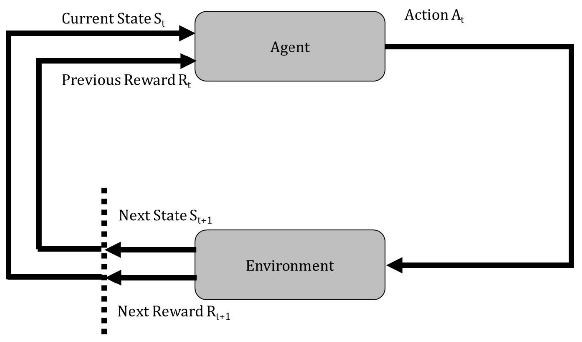


Fig. 1: The interaction of agent and environment as MDP

IV. Distributed Reactive Reinforcement Learning

A. RL

Reinforcement Learning (RL) is the branch of machine learning that deals with training agents to take an action a, as a response to the state of the environment at that particular time, s_t, to get a notion of reward, r as shown in Fig. 1. The objective of the RL agent is to maximize the collection of this reward. Sutton and Barto define RL as, " learning what to do – how to map situations to actions – so as to maximize a numerical reward signal" [43].

A reinforcement learning system has two major components: the agent and the environment where the overall system is characterized as a Markov Decision Process (MDP). The dynamics of the MDP are defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, p_0)$, with the set of states $\mathcal{S}$, the set of actions $\mathcal{A}$, a transition model $\mathcal{P}$, in which for a given state s and action a, there exists a transition probability to the next state $s' \in \mathcal{S}$, a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ which provides a reward for each state transition $s_t \to s_{t+1}$ induced by action a_t, and an initialization probability p_0. The policy $\pi(a|s)$, provides an action $a\, \epsilon\, \mathcal{A}$, for a state s presented by the environment. In value based methods, a policy could use the *state-value function*, $v(s) = E[R_t|S_t = s]$, which is the expected return from the agent when starting from a state s, or an *state-action-value function*, $q(s, a) = E[R_t|S_t = s, A_t = a]$, which is the expected return from the agent when starting from a state s, while taking action a. Here, $R_t = \sum_t \gamma^t r_t$ is the discounted reward that the agent collects over t times steps and γ is the discount factor, where $0 \leq \gamma \leq 1$. The policy then can be defined by an $\epsilon - greedy$ strategy where the actions are chosen based on $\pi(a|s) = argmax(q(s, \mathcal{A}))$ for a greedy action or a completely random action otherwise. Alternatively, policy gradient methods use a *parameterized policy*, $\pi(a|s, \theta)$, to take action without using the value functions at all. However, as shown by the policy gradient theorem, value functions may still be useful to improve the learning of the policy itself as seen in Advantage Actor Critic (A2C) [44]. The objective of the agent is to find an optimal policy, $\pi^*(a|s)$, that collects the maximum reward. To find the optimal policy, the trainable parameters of the policy are updated in such a way that it seeks to maximize the performance as defined by the cost function $J(\theta_t)$ as

$$\theta_{t+1} = \theta_t + \rho \nabla J(\theta_t), \tag{5}$$

where θ are the parameters of the policy π and ρ is the learning rate. To derive at $J(\theta_t)$ different algorithms have been developed. We use PPO [12].

B. Distributed RL

We consider modular systems with multiple individually controlled subsystems. We design fully distributed learning schemes with a decentralized state and action space and an individual reward structure per agent. More formally, let $\mathcal{N} = 1; \ldots; N$ be the set of agents and $\mathcal{X} \in \mathbb{R}^S$ be the set of states with the state vector $x_t = \left(x_t^k\right)_{k=1}^S \in \mathcal{X}$. We assume that every agent $i \in \mathcal{N}$ is influenced only by a subset of states $\mathcal{X}_i \subseteq \mathcal{X}$ consisting of neighboring states to which agent i has a direct physical relation described either by equations or edges in the production graph. Note that the state subsets $\mathcal{X}_i$ are not distinct, i.e. agents can share components of their state vector. Furthermore, let $\mathcal{A} \subseteq \mathbb{R}^Q \times \{0, 1\}^R$ denote the set of actions of all agents and $\mathcal{A}^i \subseteq \mathbb{R}^{Q_i} \times 0, 1^{R_i}$ the subset of actions of agent i. We denote $a_t^i \in \mathcal{A}^i$ the action vector of agent i at time t. Note that we assume a hybrid action set consisting of continuous and discrete actions. We further define $a^{-i} = \left(a_t^1; \ldots; a_t^{i-1}; a_t^{i+1}; \ldots; a_t^{Q_i}\right)$ as the vector of all agents' actions except that of agent i such that $a_t = \left(a_t^i; a_t^{-i}\right)$.

For each agent we define a reward function $r^i : \mathcal{X}^i \times \mathcal{A} \times \mathbb{N} \leftarrow \mathbb{R}$, such that, at every time t, each player receives a reward $r^i(x_t^i; a_t^i; a_t^{-i})$. We apply the previously described DDPG algorithm to each agent individually. Thus, each agent optimizes the own cumulative reward by taking actions from its action set $\mathcal{A}^i$ and having knowledge only to its own state set $\mathcal{X}_i$. Note that this definition avoids any direct communication between the agents. However, the agent indirectly share information due to shared states and the environment changes due to other agents' action choices.

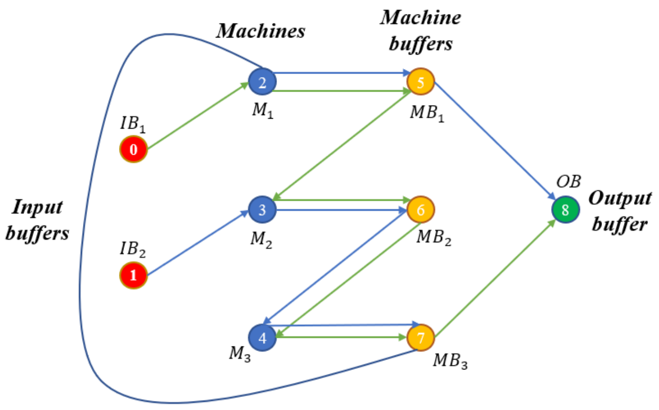


Fig. 2: RMC as Graph

V. Experiments and Results

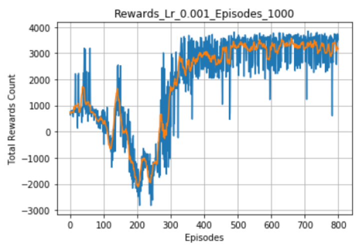
A. Robot Manufacturing Cell

In the first experiment the graph structured JSSP is implemented in a robot automated manufacturing cell. The environment consists of two industrial robots set up to work on a common platform, which is considered the shop floor here. The shop floor has six work stations, two input stations (IB1, IB2), three processing stations (M1, M2, M3), three buffer station for each of the processing stations (MB1, MB2, MB3), and one output station (OB). No queue is considered in front of the processing stations, the buffers are placed after the processing stations. The machines can only process what is moved to them at that particular point of time and then the machines stay idle when the part is moved. This is diagrammatically shown in 2. A finite supply of work-piece is considered in the system for which the agent must create a schedule.

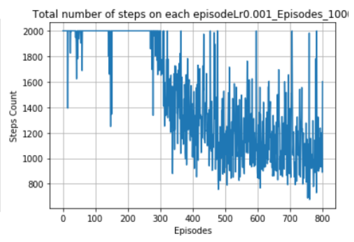
The agents action here is the activation of the edges connecting the nodes. If a node is activated then the work-piece is moved from the 'from'-node to the 'to'-node. There are two different types of nodes defined in the system, to move different work-pieces (WP1 and WP2). WP1 goes through IB1, M1, MB1, M2, MB2, M3, MB3, and then OB, while WP2 goes through IB2, M2, MB2, M3, MB3, M1, MB1, and then OB. An RFID-chip embedded in the work-piece gives information to the robots. And all the work stations are accessible by both the robots.

$$N = [WP1_count, WP2_count, machine, buffer] \tag{6}$$

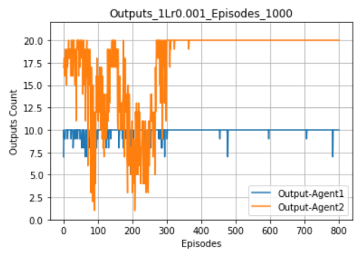
As described earlier, every graph is defined by the tuple of node features v, edge features e, and edge index $e_{i,j}$. The node feature is a vector of size four as given by equation 6. The first location is the number of work-piece, the second location is the number of work-piece two, the third and fourth locations are for differentiating between machine (1, 0) and buffer (0, 1). For example, at the start of the episode, IB1 will have a node



(a) Rewards

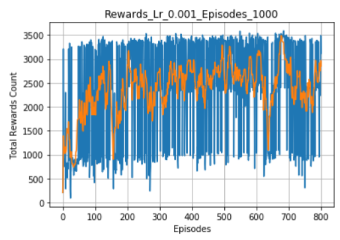


(b) Steps

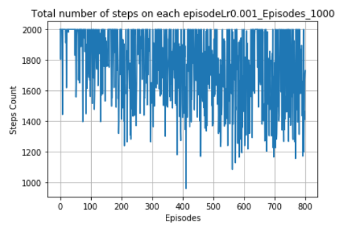


(c) Targets

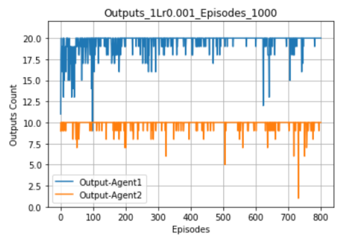
Fig. 3: Output Targets: 20, 20



(a) Rewards



(b) Steps



(c) Targets

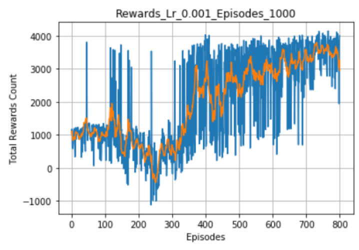
Fig. 4: Outputs target: 10, 20

feature vector of $[20, 0, 1, 0]$. The node features are combined together as a tensor of size $[N^v, feature\, length]$, where N^v is the number of nodes. The RMC graph has nine nodes. The edge index is nothing but a tensor of size $[2, N^e]$, where N^e is the number of edges. Edge index provides information about how the nodes are connected to one another. For example, the first edge index is given by $[s, r]$, where s is the sender node, and r is the receiver node. And each of these edges have an attribute that define which work piece the edge can carry. So the edges that carry work-piece one have an attribute $[1, 0]$ and the edges that carry work-piece two have an attribute $[0, 1]$. The RMC module has 14 edges, 7 which carry work-piece one and 7 which carry work-piece 2.

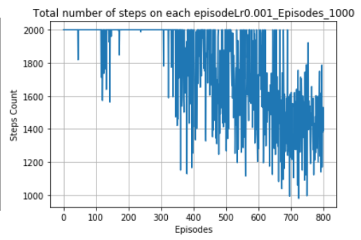
With the above information, the message passed feature vector can be extracted. This information is then used by the RL agent as the states to take action. There are two agents one for each of the work-pieces. The action is defined by whether the edges activate or not. So when the edges activate they carry their corresponding work-piece from the 'sender node' to the 'receiver node'. The size of the action space is for each agent is 128, given the 2^7 number of possible actions.

B. Robot Manufacturing Cell Results

Initial results show that, the GNN based RL comfortably performs better than the results from the previous research on the same environment [45]. Not only does it converge 10x faster, the stability achieved after convergence is also very good as shown in figures 3 4 5. Another very encouraging result is that, while the previous algorithms could not converge for variable targets like $[10, 20], and [20, 10]$, the GNN based RL could easily those targets as well as depicted in the figures 4 5.



(a) Rewards



(b) Steps

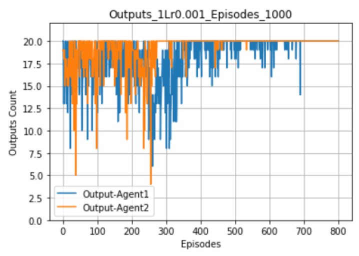


Fig. 5: Outputs target: 20, 10

C. Injection Molding Machines

Next we apply the algorithm fine-tuned on the RMC, to a bigger environment. The environment is set up based on a dataset taken from the Polytechnique University of Valencia (UPV) for injection molding presses. It has four injection molding presses which process 30 different jobs that follow different machining sequences. Each work-piece must through all four machines to reach output buffer (OB) and there are different processing times for every work-piece on their corresponding machines. An example of flow of a work-piece on machines is shown below, for the complete list of processing sequence and times refer to the Annexure. For example, Job 1 has a machining sequence of $IB->M2->M3->M1->M4->OB$, with processing times of 14, 12, 20, and 10 time-steps respectively to reach the OB. And the OB collects the completed work-pieces. The machines can process only one work-piece at a single time. The finished work-pieces from the preceding

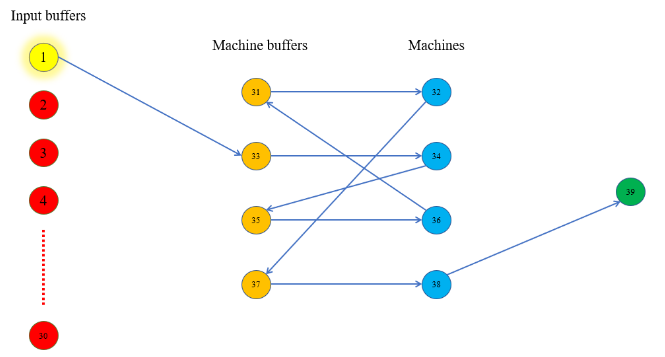


Fig. 6: Example of one job in the injection molding machines

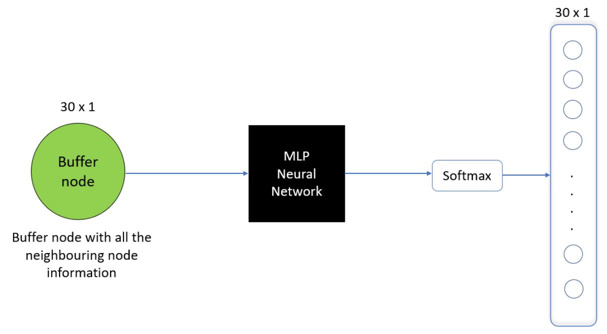


Fig. 7: RL agent action input

machine moves in to the succeeding machine's buffer. The machine is which is empty after completion of the previous task picks the next work-piece from their machine buffer. The objective is to finish the molding process of all work-pieces within the least make-span possible. There are other objectives which can also be considered such as minimizing the total weighted completion time, minimizing the maximum of lateness, minimizing the total weighted tardiness, etc [5]. But this experiment is concentrated on minimizing the make-span and processing the jobs using coordinated of the RL agents.

The GNN structured environment of the IMM is similar to the RMC with a few differences. There are four different nodes, input buffers, machines, machine buffers, and output buffer with feature sizes $[1], [2], [30], and [30]$ respectively. The features are [Job number], [Job number, Time remaining], [Number of respective Jobs], and [Number of respective Jobs]. But in order to process them all together in the GNN the node features should be of similar size. Hence input buffer feature is one-hot encoded to the size of 30, while the machine features are padded with zeros to arrive at the size of 30. There are 30 different edges with a one-hot encoded edge feature vector of size 30.

There are four agents in the IMM setup, each operating on the machines where jobs are processed. The machines request for a job when they are empty. Using the above described structure, the message passed feature vector is extracted for each of the nodes. But unlike in the RMC setup, in IMM only the extracted features of the machine buffer node is used to take action. This is because the machine buffer node containing the information of all neighboring nodes due to the message passing. This also future-proofs the algorithm, since when this solution is scaled up to bigger scheduling problems, the state space will not increase exponentially as only the local information is used. **Although initial results were found to be encouraging, they will be published in the future work for the Injection Molding Machines environment.**

VI. Conclusion

MLP normally used in RL of the manufacturing scheduling problems use just the feature vector of the states. While this seems sufficient for simple environment, in complex environments the agents find it difficult to learn anything meaningful. This is because of the weak implicit inductive bias provided by the MLP. GNNs overcome this by processing the graph structure of the manufacturing scheduling problems as graphs. The message passing algorithm of GNNs provide better implicit inductive bias that are also invariant to the position of the nodes and edges.

Our initial results from the application of GNNs to solve the manufacturing scheduling problems show great promise. The learning times are 10x times lower while proving higher stability in the RMC environment. The agents also comfortably solve the IMM environment with 30 jobs and 4 machines accommodating the processing times as well. Although these are not shown here, the initial results highlight the highly promising nature of the Graph Neural Network application in job shop scheduling problems. Further enhancements to the learning algorithm is required to ensure stable learning in the environment which forms the future work on this topic.

References

[1] LUDO F. GELDERS and NARAYANASAMY SAMBANDAM, “Four simple heuristics for scheduling a flow-shop,” *International Journal of Production Research*, vol. 16, no. 3, pp. 221–231, 1978, ISSN: 0020-7543. DOI: 10.1080/00207547808930015.

[2] P. Baptiste and C. Le Pape, “A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling,” in *IJCAI (1)*, 1995, pp. 600–606.

[3] D. Y. Lee and F. DiCesare, “Scheduling flexible manufacturing systems using petri nets and heuristic search,” *IEEE Transactions on robotics and automation*, vol. 10, no. 2, pp. 123–132, 1994.

[4] A. S. Jain and S. Meeran, “Job-shop scheduling using neural networks,” *International Journal of Production Research*, vol. 36, no. 5, pp. 1249–1272, 1998, ISSN: 0020-7543.

[5] J. J. Kanet and H. H. Adelsberger, “Expert systems in production scheduling,” *European Journal of Operational Research*, vol. 29, no. 1, pp. 51–59, 1987.

[6] Hsiano-Lan Fang, Peter Ross, and Dave Corne, "A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems," *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 375–382, 1993.

[7] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra, "Job shop scheduling by simulated annealing," *Operations research*, vol. 40, no. 1, pp. 113–125, 1992.

[8] S. H. Chung, F. T. S. Chan, and H. K. Chan, "A modified genetic algorithm approach for scheduling of perfect maintenance in distributed production scheduling," *Engineering Applications of Artificial Intelligence*, vol. 22, no. 7, pp. 1005–1014, 2009.

[9] S. Cavalieri, M. Garetti, M. Macchi, and M. Taisch, "An experimental benchmarking of two multi-agent architectures for production scheduling and control," *Computers in Industry*, vol. 43, no. 2, pp. 139–152, 2000.

[10] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017. DOI: 10.1038/nature24270.

[11] S. Gu, E. Holly, T. Lillicrap, and S. Levine, *Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates*, 2016. [Online]. Available: http://arxiv.org/pdf/1610.00633v2.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. [Online]. Available: https://arxiv.org/pdf/1707.06347.

[13] J.K. Lenstra and A. Rinnooy Kan, "Computational complexity of discrete optimization problems," *Annals of Discrete Mathematics*, vol. 4, pp. 121–140, 1979, ISSN: 0167-5060. DOI: 10.1016/S0167-5060(08)70821-5.

[14] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," *Computers & Operations Research*, vol. 73, pp. 165–173, 2016, ISSN: 03050548. DOI: 10.1016/j.cor.2016.04.006.

[15] C. Özgüven, Y. Yavuz, and L. Özbakır, "Mixed integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence dependent setup times," *Applied Mathematical Modelling*, vol. 36, no. 2, pp. 846–858, 2012, ISSN: 0307904X. DOI: 10.1016/j.apm.2011.07.037.

[16] A. Baykasoğlu, L. özbakir, and A. İ. Sönmez, "Using multiple objective tabu search and grammars to model and solve multi-objective flexible job shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 15, no. 6, pp. 777–785, 2004, ISSN: 0956-5515. DOI: 10.1023/B:JIMS.0000042663.16199.84.

[17] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018, ISSN: 0036-8075. DOI: 10.1126/science.aar6404. [Online]. Available: http://science.sciencemag.org/content/362/6419/1140.

[18] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, *Mastering atari, go, chess and shogi by planning with a learned model*, 2019. [Online]. Available: https://arxiv.org/pdf/1911.08265.

[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. DOI: 10.1038/nature14236.

[20] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt, *Multi-task deep reinforcement learning with popart*, 2018. [Online]. Available: http://arxiv.org/pdf/1809.04474v1.

[21] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017, ISSN: 09252312. DOI: 10.1016/j.neucom.2016.12.038.

[22] W. Zhang and T. G. Dietterich, "A reinforcement learning approach to job-shop scheduling," in *IJCAI*, vol. 95, 1995, pp. 1114–1120.

[23] T. Gabel and M. Riedmiller, "Adaptive reactive job-shop scheduling with reinforcement learning agents," *International Journal of Information Technology and Intelligent Computing*, vol. 24, no. 4, 2008.

[24] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek, "Optimization of global production scheduling with deep reinforcement learning," *Procedia CIRP*, vol. 72, no. 1, pp. 1264–1269, 2018.

[25] I.-B. Park, J. Huh, J. Kim, and J. Park, "A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities: Ieee transactions on automation science and engineering, 1-12," *IEEE Transactions on Automation Science and Engineering*, pp. 1–12, 2020, ISSN: 1545-5955. DOI: 10.1109/TASE.2019.2956762.

[26] S. Kapoor, "Multi-agent reinforcement learning: A report on challenges and approaches," *CoRR*, vol. abs/1807.09427, 2018. arXiv: 1807.09427.

[27] M. Lauer and M. A. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *Proceedings of the 17th International Conference on Machine Learning, San Francisco, USA*, 2000, pp. 535–542.

[28] L. Matignon, G. J. Laurent, and N. L. Fort-Piat, "Hysteretic q-learning :an algorithm for decentralized reinforcement learning in cooperative multi-agent teams,"

2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 64–69, 2007.

[29] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Annual Conference on Neural Information Processing Systems 2017, Long Beach, USA*, 2017, pp. 6379–6390.

[30] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, USA*, 2018, pp. 2974–2982.

[31] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. F. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, Stockholm, Sweden,*, 2018, pp. 2085–2087.

[32] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," in *Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia*, 2017, pp. 2681–2690.

[33] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *CoRR*, vol. abs/1511.08779, 2015. arXiv: 1511.08779.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[35] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, *Relational inductive biases, deep learning, and graph networks*, 2018. [Online]. Available: https://arxiv.org/pdf/1806.01261.

[36] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun, "Spectral networks and locally connected networks on graphs," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun, Eds., 2014.

[37] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc, 2017, pp. 3391–3401. [Online]. Available: http://papers.nips.cc/paper/6931-deep-sets.pdf.

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ.

[39] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel, *Gated graph sequence neural networks*, 2015.

[40] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Red Hook, NY, USA: Curran Associates Inc, 2017, pp. 1025–1035, ISBN: 9781510860964.

[41] T. Seito and S. Munakata, "Production scheduling based on deep reinforcement learning using graph convolutional neural network," in *Proceedings of the 12th International Conference on Agents and Artificial Intelligence*, SCITEPRESS - Science and Technology Publications, 22-02-2020 - 24-02-2020, pp. 766–772, ISBN: 978-989-758-395-7. DOI: 10.5220/0009095207660772.

[42] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl, "Neural message passing for quantum chemistry," Doina Precup and Yee Whye Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 1263–1272. [Online]. Available: http://proceedings.mlr.press/v70/gilmer17a.html.

[43] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction / Richard S. Sutton and Andrew G. Barto*, ser. Adaptive computation and machine learning. Cambridge, Mass. and London: MIT Press, 1998, ISBN: 0262193981.

[44] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *International Conference on Machine Learning*, pp. 1928–1937, 2016, ISSN: 1938-7228. [Online]. Available: http://proceedings.mlr.press/v48/mniha16.html.

[45] A. Schwung, D. Schwung, and M. S. Abdul Hameed, "Cooperative robot control in flexible manufacturing cells: Centralized vs. distributed approaches," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, Piscataway, NJ: IEEE, 2019, pp. 233–238, ISBN: 978-1-7281-2927-3. DOI: 10.1109/INDIN41052.2019.8972060.