AIML425 - PROJECT

Vinh Nguyen - 300473488

ABSTRACT

A Deep Reinforcement Learning (DRL) method has not been fully developed for the Flexible Job Shop Scheduling Problem (FJSP). However, this has changed with the recently introduced paper "Flexible Job Shop Scheduling via Graph Neural Network and Deep Reinforcement Learning" by Wen Song, Xinyang Chen, Qiqiang Li, and Zhiquang Cao. They has developed a DRL method for learning high-quality PDRs end-to-end using Heterogeneous Graph Neural Network for representing and capturing complex relationship between operations and machines. The proposed method was found to be both efficient and competitive in benchmarks results.

***Index Terms*—** Flexible Job-shop Scheduling, Graph Neural Network, Deep Reinforcement Learning

1. INTRODUCTION

For this project, I am required to analyse the proposed deep learning research paper [1] and provide a related contribution. First I will go over the paper to talk about its content and impact with a dedicated part on the background of Graph theory and Graph Neural Network (GNN) with Graph Attention Network (GAT) aggregation. Expanding on their work, a contribution is made to improve the model ability to extract richer information from the data with less computations using GAT.

2. BACKGROUND

2.1. Preliminaries

2.1.1. Job Shop Scheduling

In job shop scheduling (JSS) [2], a number of jobs need to be processed by a set of machines. Each job has a sequence of operations following order constraints. An operation cannot start until its preceding operations have completed and its assigned machine is idle. Each machine can only process one operation at a time+. The main objective in JSS is to process the operations in an order that minimise the total duration (makespan) of the schedule. JSS has been shown to be an important combinatorial optimisation problem that closely match real-world applications such as cloud computing.

Flexible Job Shop Scheduling (FJSS) is an extension of JSS [3] but with the added flexibility of each operation can be processed by a set of candidate machines rather than a single machine. This additional task of machine assignment decision result in a strongly NP-hard problem for FJSS. To solve the FJSS problem, we need to assignment each operation to a compatible machine and determine its start time, such that the makespan is minimised. More details on the preliminaries can be found in section III of the paper [1] page 3.

2.2. Graph Theory

This was originally a branch of mathematics which focuses on networks of points (nodes) connected by lines (edges). Graph theory has grown significantly over the years and become an important part in research and application in production scheduling like job shop scheduling (JSS) [4].

2.2.1. Disjunctive graph for JSS

A common representation used in JSS is a disjunctive graph $G = (V, C, D)$, where
-V is set of nodes representing operations of the jobs with two dummy nodes (start and end) representing the beginning and end of the schedule.
-C is a set of conjunctive arcs representing the sequence of operations.
-D is a set of undirected disjunctive arcs that connect operations that can be processed on the same machines.

2.2.2. Heterogeneous Graph for FJSS

The above representation work well with static JSS but becomes much more dense with Flexible Job Shop Scheduling (FJSS) where each operation can be processed by a set of candidate machines instead of just one. A way around this is to utilize heterogeneous graph (HetG) representation. The new graph is $H = (O, M, C, \varepsilon)$, where
-O and C sets are kept the same.
-M is a set of nodes representing the available machines.
-ε is a set of undirected arc that connect an operation node with a compatible machine node and contain the corresponding processing time of the pair.

The HetG provides a much more meaningful representation of nodes and edges in the graph [5] and is crucial for the embedding stage with graph neural networks.

2.3. Graph Neural Networks

2.3.1. *Objective Functions*

The main objective of GNN is to perform inference on the data using the input graph representation. For this FJSS problem, the focus is on finding the embedding for each node that correlates with a priority property (likelihood that it being chosen as the next operation/machine in the schedule). However, because there are different type of relationships between nodes in FJSS (e.g. machine-to-machine, operation-to-operation, operation-machine, ..) we need to employ an attention mechanisms such as Graph Attention Network (GAT) to learn the important between nodes and its neighbours before the embeddings.

2.3.2. *Aggregation with GAT*

Typical aggregation treats all neighbours equally using the basic formula in sum, mean, max and min. In a JSS context, some nodes are more important than others (e.g for a given machine, an operation that is expected to start earlier might be more important than those that start later). This motivates the use of GAT [6] which can automatically learn the important between nodes by applying the attention mechanism.
First it computes the attention coefficient e_{ij} between i and each of its first-order neighbour j (including itself):

$$e_{ij} = LeakyReLU(W_a^T[W_{xi}||W_{xj}) \tag{1}$$

where $W_a^T \in R^{2d'}$ and $W \in R^{d' \in d}$ are learned parameters, d' is the embedding dimension, and $||$ is the concatenation operation. This step is the most important as it indicates the importance of node j features to node i. Further more it allow for every node to attend on every other node, regardless the structural information.
The coefficients are then normalized across the neighborhood:

$$a_{ij} = \frac{exp(e_{ij})}{\sum\limits_{q \in N(i)} exp(e_{iq})}, \forall j \in N(i) \tag{2}$$

The final aggregation step aggregates the embeddings from neighbors together then scale by the attention scores:

$$x_i^{'} = \sigma(\sum_{j \in N(i)} a_{ij} W x_j) \tag{3}$$

where the GAT applies a non-linearity step σ to the linearly transformed the features over $N(i)$ to acquire embedding of i

2.3.3. *Update*

The updated GNN is then:

$$h_i = \sigma(W_1 x_i^{'} + \sum_{j \in N(i)} W_2 x_j^{'}) \tag{4}$$

which formulates a single GNN layer for a single node i.

3. PAPER ANALYSIS

3.1. Method Overview

3.1.1. *An end-to-end DRL method for FJSP*

The main goal of the paper is to develop an end-to-end Deep Reinforcement Learning (DRL) method for solving the Flexible Job Shop Problem. By using DRL, their method was able efficiently lean high-quality priority dispatching rules (PDRs) with good generalisation and performance.

Fig. 1. shows the workflow of end-to-end method during training. It starts with the raw data from the FJSP instance represented as a state. This data is then transformed into a heterogeneous graph structure which is feed into a heterogeneous graph neural network for computing the embeddings. The embeddings information is sent to the DRL agent which apply its policy for finding the best action for a given state. Once completed, the environment transition into the next state with an updated state and calculated reward.

3.1.2. *MDP formulation*

In order to train the model using DRL, they develop a Markov Decision Process [7] (MDP) formulation for the FJSP. As the main focus of AIML425 is not on reinforcement learning, I decided forcus my analysis on the graph neural network part of the paper. For more details on the MDP formation, please see chapter IV. A of the paper [1].

3.1.3. *Heterogeneous Graph Structure*

Due to the added complexity of FJSP, a simple disjunctive graph is not enough to represent the information between neighbours (machine and operations). To get around this, they utilise the HetG structure from 2.2.2 to represent the operation and machine nodes and their states.

A HetG representation allows for multiple type of nodes and edges to be on the same graph without the added density a disjunctive graph would have. This is particularly useful for the embedding stage in HGNN as the input information contains more comprehensive information and rich semantics.

Fig. 2. provides a visualisation of how an FJSP instance and solution would look like using HetG.

3.1.4. *HGNN based Neural Architecture*

This is the most important part of the training process where state information are extracted and embedded before being passed to the decision making process. They have formulated a custom HGNN architecture for FJSP that can handle graph states with different sizes (due to the dynamic of JSS). The

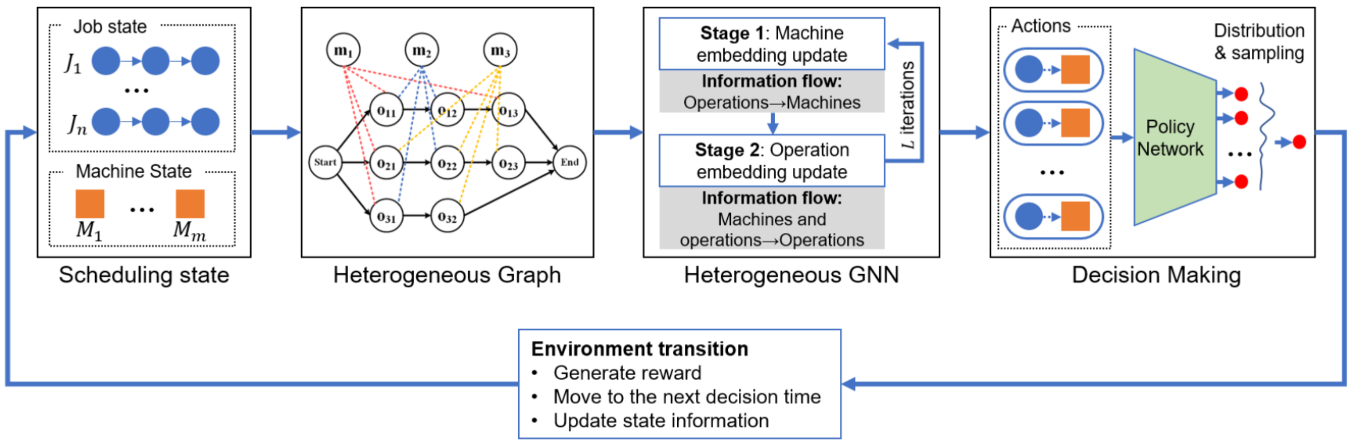

Fig. 1. The workflow of the proposed method

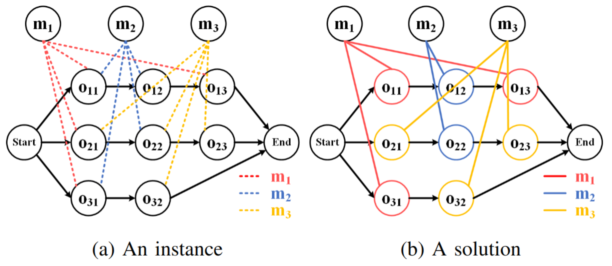

Fig. 2. Heterogeneous graph of FJSP. Dotted line mean processable, while solid line means scheduled

proposed HGNN go to a two stage embedding process (Fig .3) to extract the machine embeddings in the first stage then operation embeddings in the second stage.

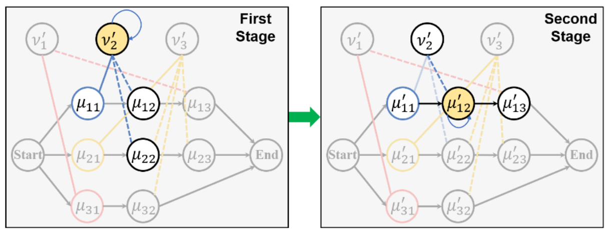

Fig. 3. Two-stage embedding scheme

Stage one utilises graph attention network similar to the one discussed in 2.3.2 but with a few tweaks to work for the FJSP. Due to the important of the edge feature, O-M arcs (processing time), the operation raw feature μ_{ijk} is extended to include the O-M arcs information of machine-operation pair. To get the machine attention coefficient $eijk$ for a given machine v_k to its neighboring operation μ_{ijk} is as followed:

$$e_{ijk} = LeakyReLU(W_a^T[W^M v_k || W^O \mu_{ijk}) \quad (5)$$

where $W_a^T \in R^2d$, $W^M \in R^{d\times 3}$ and $W^O \in R^{d\times 7}$ are trainable weights for machine and operation node respectively. This allow the information from heterogeneous nodes and the O-M arcs to be included in the attention calculation of the embeddings.

As an exception, for machine to itself (e_{kk}) this is computed slightly differently:

$$e_{kk} = LeakyReLU(W_a^T[W^M v_k || W^M v_k]) \quad (6)$$

The attention coefficient are then normalized via softmax to get a_{ijk} and a_{kk}. The final aggregation is for getting the machine embedding $v_k^{'}$:

$$v_k^{'} = \sigma(a_{kk} W^M v_k + \sum_{O_{ij} \in N_t(M_k)} a_{ijk} W^O \mu_{ijk}) \quad (7)$$

By modifying the GAT calculation they were able to compute a meaningful embedding for machine using the raw features of machine and operation as well as the edge information for the processing time.

In stage two, multiple MLPs are used to compute the final embedding for the operation O. Due to having neighbors of multiple types (machines, predecessor/successor operation, and the operation itself) they have gone with five MLPs ($MLP_{\theta 0}$, ..., $MLP_{\theta 4}$). These are for the final projection, processing predecessor operation $\mu_{i,j-1}$, processing successor operation $\mu_{i,j+1}$, processing eligible machines $M_k \in N_t(O_{ij})$, and itself μ_{ij} respectively.
The operation embedding $\mu_{ij}^{'}$ is as follows:

$$\mu_{ij}^{'} = MLP_{\theta 0}(ELU[MLP_{\theta 1}(\mu_{i,j-1})||MLP_{\theta 2}(\mu_{i,j+1}) \\ ||MLP_{\theta 3}(\overline{v}_{ij})||MLP_{\theta 4}(\mu_{ij}))]$$

where $\overline{v}_{ij}$ is an element-wise sum of the machine(s) eligible with operation μ_{ij}

Due to the inconstancy between neighbors features and importance to an operation, they choose to combine the outputs multiple MLPs to hopefully capture all the information in the current state for the operation embedding.

The embeddings above are considered as a HGNN layer which take in the raw features v_k for machine and μ_{ij} for operation and compute the corresponding embeddings $v_k^{'}$ and $\mu_{ij}^{'}$. To enhance the feature extraction in the embedding, they stacked L HGNN layers with same structure but with independent trainable parameters to achieve the final embeddings $v_k^{'(L)}$ and $\mu_{ij}^{'(L)}$. To use this new embeddings we need to average across the nodes to have a meaningful output $\frac{1}{O}(\sum_{O_{ij}\in O} v_k^{'(L)})$ and $\frac{1}{M}(\sum_{M_k\in M} \mu_{ij}^{'(L)})$.
Finally the two embeddings are concatenated as the final output for the updated graph state h_t:

$$h_t = [\frac{1}{O}(\sum_{O_{ij}\in O} v_k^{'(L)})||\frac{1}{M}(\sum_{M_k\in M} \mu_{ij}^{'(L)})] \tag{8}$$

3.1.5. *Decision Making*

The final part of the purposed method uses the embeddings information from the HGNN to feed directly to the DRL agent and critic. For an action a_t in a given current state s_t, the graph state h_t, machine embedding $v_k^{'(L)}$, and operation embedding $\mu_{ij}^{'(L)}$ are computed and combined together and feed to the MLP to calculate the priority index of all feasible actions.

$$P(a_t|s_t) = MLP_w[v_k^{'(L)})||\mu_{ij}^{'(L)}||h_t] \tag{9}$$

They then sample across the softmax distribution and pick the next action for the schedule.

$$\pi_w(a_t|s_t) = softmax(P(a_t|s_t), \forall a_t \in A_t \tag{10}$$

where A_t contains all the feasible actions in the current state.

During training, the actor acts as the policy network while the critic predicts the value of a current state using only the graph state h_t as the input. At every 10 iterations, the policy is verified on a independent validation instance and the batch data is replaced every 20 iteration to simulate the variability in the current job shop. The loss function used is Proximal Policy Optimization [8] (PPO) loss which is tailored for reinforcement learning algorithm.

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{11}$$

where,
θ is the policy parameter.
$\hat{E}_t$ is the empirical expectation over time-steps.
r_t is the ratio of the probability under the new and old policies, respectively.
$\hat{A}_t$ is the estimated advantage at time t.
ϵ is a hyper-parameter, usually 0.1 or 0.2.

3.2. Impact

Using the proposed method, they were able to achieve good results on well-known benchmarks instances that outperforms human-design priority dispatching rules and rival population-based methods such Genetic Algorithms. In terms of real-world impact, as more and more real-world applications move towards cloud computing this has the potential to solve a lot of the difficulties that come with job scheduling and allocation.

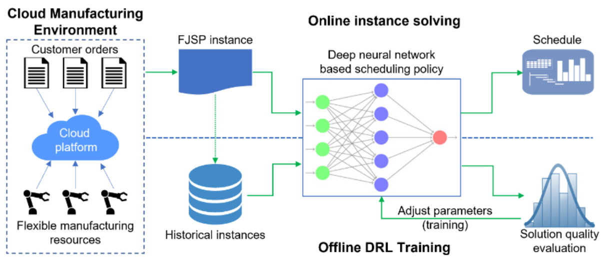
Cloud Manufacturing Environment
Customer orders
Cloud platform
Flexible manufacturing resources
FJSP instance
Historical instances
Online instance solving
Deep neural network based scheduling policy
Adjust parameters (training)
Offline DRL Training
Schedule
Solution quality evaluation

Fig. 4. Potential application in cloud computing of the DRL-based method

4. CONTRIBUTION

My contribution to this paper is a modification to the operation embedding in stage two of the HGNN. Instead of using multiple MLPs to capture the information, I instead replace the process with two GATs and a MLP for final projection.

4.1. Reasoning

As stated in 3.1.4. The authors decided that the most effective way to extract the information for the operation embedding is to use 5 MLPs each responsible for processing a different piece of information with the exception of the first one that is for the final projection. This proves effective as their results indicate that the model were able to effectively use the embedding information for the operation.

Although MLPs are simple to use and can effectively process the operation information they do not aggregate messages between nodes during the HGNN stage. This is a crucial part for learning the importance between nodes for the final embedding so I believe the GAT aggregation should be implemented here.

4.2. Modification

Rather than using multiple MLPs, I went with two GATs (one for the predecessor operation to current operation and another one for successor operation to the current operation and a single MLP used for final projection.

4.2.1. Predecessor operation GAT

The embedding for a given predecessor operation to its neighboring operation (current operation) is as followed:

$$\mu_{ijk-pre} = softmax(LeakyReLU(W_a^T[W\mu_{i,j-1}||W\mu_{ijk}])) \tag{12}$$

4.2.2. Successor operation GAT

The embedding for a given successor operation to its neighboring operation (current operation) is as followed:

$$\mu_{ijk-suc} = softmax(LeakyReLU(W_a^T[W\mu_{i,j+1}||W\mu_{ijk}])) \tag{13}$$

4.2.3. Final projection

Both embeddings are then concatenated and feed into a MLP to perform the final projection and we use this output as the final operation node embedding $\mu_{ij}^{'}$.

$$\mu_{ij}^{'} = MLP[\mu_{ijk-pre}||\mu_{ijk-suc}] \tag{14}$$

4.3. Results & Discussion

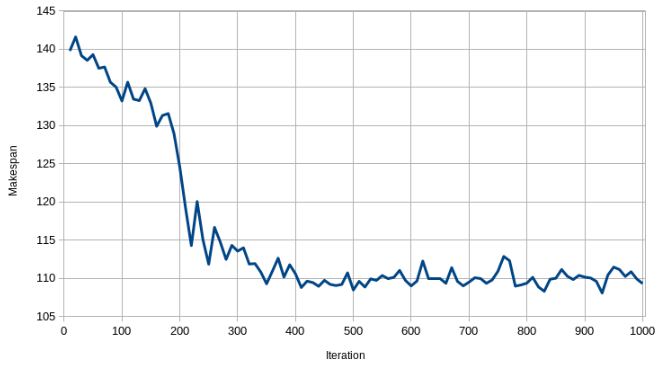

Fig. 5. Mean validation makespan during training on 10x5 instances on the original algorithm of the paper

Looking at the results, the original algorithm converges faster with lower overall mean makespan on the validation instances. However, the end results are mostly similar across both indicating that the model was able to extract sufficient information from the node embedding. I have also tested the modified algorithm on the test data (for 10x5 instances) and

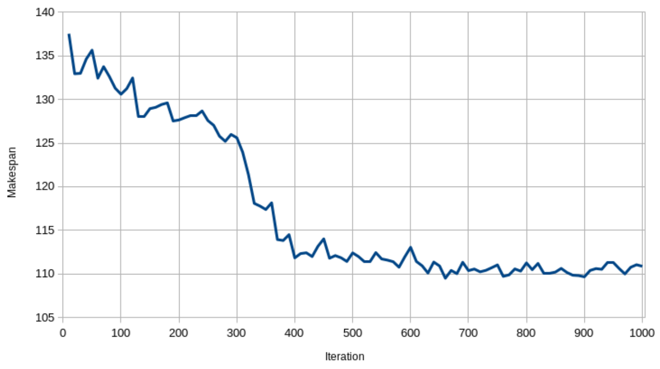

Fig. 6. Mean validation makespan during training on 10x5 instances on the modified algorithm

achieve similar results to the ones found the paper [1] in Table II of chapter 5.

Besides being able to effectively perform the GAT aggregation in HGNN, the reduce number of trainable parameter also speed up the training process significantly. The original algorithm took my laptop 4 hours to train while the modified one only took 2.2 hours, nearly twice faster.

On larger instances, the modified algorithm does struggle to extract enough information for the node embeddings and is outperformed by the original algorithm using multiple MLPs. I suspect the added number of possible actions in each state in the larger instances are too much for the two GATs to handle, thus unable to capture the full information and perform worse. Overall, the modification made prove that the two GATs and a single MLP used were sufficient for the operation embedding at least on smaller instances.

5. CONCLUSION

In this report, I discussed the theory behind FJSS, GNN, and GAT which serves as a background to by analysis of the paper. I then talked about my contribution, the reasoning behind it, what was modified and the results and discussion of the original algorithm and modified one. The proposed method used by the authors provided a novel DRL-based method for learning high-quality PDRs that outperform human-designs ones with great applicability for real-world tasks. Finally, my contribution shows how using GATs aggregation instead of multiple MLPs could provide sufficient information for the node embeddings with comparable results to the original method.

For code, please visit:
`https://github.com/Vinsukii/AIML425_Project`

6. REFERENCES

[1] Song, W., Chen, X., Li, Q., Cao, Z. *"Flexible Job Shop Scheduling via Graph Neural Network and Deep Reinforcement Learning,"* IEEE Transactions on Industrial Informatics, 2022.

[2] Manne, A. E. *"On the job-shop scheduling problem,"* Operations Research, 1960, p162.

[3] Burker, P., Schlie, R. *"Job-shop scheduling with multi-purpose machines,"* Computing, 1990

[4] Feng, W., Jie, L., Xiao-hua, L. *"Model of Job Shop scheduling Based on Graph Theory and Combination of Machines,"* IEEE Xplore, 2008.

[5] Chuxu, Z., Dongjin, S., Chao, H., Ananthram, S., Nitesh, V. C. *"Heterogeneous Graph Neural Network,"* Association for Computing Machinery, 2019.

[6] Velckovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengjo, Y. "Graph attention networks," International Conference on Learning Representations (ICLR), 2018.

[7] Bellman, R. *"A Markovian Decision Process,"* Journal of Mathematics, 1957.

[8] John, S., Filip, W., Prafulla, D., Alec, R., Oleg, K. *"Proximal Policy Optimization Algorithms,"*, cs.LG, 2017.