A Multi-Graph Attributed Reinforcement Learning based Optimization Algorithm for Large-scale Hybrid Flow Shop Scheduling Problem

Fei Ni*
School of Computers Science and Technology, Huazhong University of Science and Technology
nifei@hust.edu.cn

Jianye Hao†
College of Intelligence and Computing,Tianjin University
Huawei Noah's Ark Lab
jianye.hao@tju.edu.cn

Jiawen Lu
Xialiang Tong
Mingxuan Yuan
Huawei Noah's Ark Lab
{jiawen.lu,tongxialiang}@huawei.com
Yuan.Mingxuan@huawei.com

Jiahui Duan
College of Computer Science,Sichuan University
duanjiahui@stu.scu.edu.cn

Yi Ma
College of Intelligence and Computing,Tianjin University
mayi@tju.edu.cn

Kun He†
School of Computers Science and Technology, Huazhong University of Science and Technology
brooklet60@hust.edu.cn

ABSTRACT

Hybrid Flow Shop Scheduling Problem (HFSP) is an essential problem in the automated warehouse scheduling, aiming at optimizing the sequence of jobs and the assignment of machines to utilize the makespan or other objectives. Existing algorithms adopt fixed search paradigm based on expert knowledge to seek satisfactory solutions. However, considering the varying data distribution and large scale of the practical HFSP, these methods fail to guarantee the quality of the obtained solution under the real-time requirement, especially facing extremely different data distribution. To address this challenge, we propose a novel Multi-Graph Attributed Reinforcement Learning based Optimization (MGRO) algorithm to better tackle the practical large-scale HFSP and improve the existing algorithm. Owing to incorporating the reinforcement learning-based policy search approach with classic search operators and the powerful multi-graph based representation, MGRO is capable of adjusting the search paradigm according to specific instances and enhancing the search efficiency. Specifically, we formulate the Gantt chart of the instance into the multi-graph-structured data. Then Graph Neural Network (GNN) and attention-based adaptive weighted pooling are employed to represent the state and make MGRO size-agnostic across arbitrary sizes of instances. In addition, a useful reward shaping approach is designed to facilitate model convergence. Extensive numerical experiments on both the publicly available dataset and real industrial dataset from Huawei Supply Chain Business Unit demonstrate the superiority of MGRO over existing baselines.

CCS CONCEPTS

• **Applied computing** → **Supply chain management**; • **Computing methodologies** → **Planning with abstraction and generalization**.

KEYWORDS

Hybrid Flow Shop Scheduling, Reinforcement Learning, Graph Neural Network

ACM Reference Format:
Fei Ni, Jianye Hao, Jiawen Lu, Xialiang Tong, Mingxuan Yuan, Jiahui Duan, Yi Ma, and Kun He. 2021. A Multi-Graph Attributed Reinforcement Learning based Optimization Algorithm for Large-scale Hybrid Flow Shop Scheduling Problem. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467135

1 INTRODUCTION

With the development of Internet-of-Things technology and the growing number of business orders, the automated warehouse has become a heated topic in recent years. Efficient warehouse management plays an essential role in the automated warehouse, in which how to schedule orders dramatically affects the economic benefit and customer satisfaction. The workflow of the automated warehouse is shown in Figure 1. Generally, a customer's order consists of multiple products, and each product is packed by different types and quantities of boxes, which are determined by the information system of the warehouse and denoted as jobs in the following. Firstly, the required raw materials are allocated from the inventory according to the demands of orders. Then all jobs of the order need to go through three stages, including picking, packing

*Work done as intern at Huawei Noah's Ark Lab.
†Corresponding authors.

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467135

and weighting before leaving the warehouse. Each stage contains multiple processors, namely machines and each job can be executed on any one of the machines in the corresponding stage. An efficient scheduling plan can greatly reduce the completion time of each order and enlarge the throughput of the warehouse.

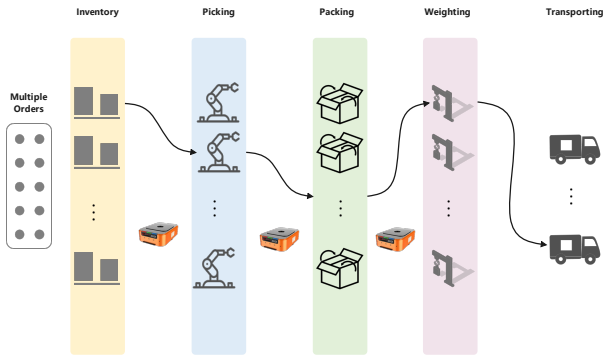


Figure 1: A workflow example of the automated warehouse.

The order scheduling problem in the automated warehouse can be modeled as the Hybrid Flow Shop Scheduling Problem (HFSP) without considering some factors listed in Section 3. The HFSP has been proved to be NP-hard [21], which needs to determine the sequence of jobs in each stage and the allocation plan for each machine. The difficulty of the practical problem is mainly owing to a larger scale, higher real-time requirement and more varying data distribution. The common solution methods to deal with the HFSP are exact algorithms and heuristic-based algorithms. Exact algorithms are rarely used due to its extremely expensive computational cost in practical scenarios. The heuristic-based approaches can be generally divided into two categories. The first one is heuristic methods [10, 19], which are designed on the specific characteristics of the problem and construct a satisfactory solution directly without improving procedure. This class of method is fast but cannot guarantee the quality of the obtained solution. The other one is meta-heuristic methods [1, 16, 20] through the iterated improving process. In these methods, the fine-tuning search framework is designed which relies heavily on the domain and expert knowledge and the pattern of using search operators is fixed during the entire improving process, which limits the search efficiency when facing extremely different order distribution in practical scenarios. To tackle the above obstacles, some research work [11, 13] has tried to introduce learning mechanism into traditional algorithms to improve the search efficiency, enhance the generalization and ease the design of search framework. They all focus on learning to construct the solution directly in the *end-to-end* way. Without the improving procedure, these methods can quickly obtain the solution, but the quality can rarely satisfy the quality requirements in practical scenarios. Moreover, the insufficiently powerful representation in these methods makes them can just distinguish between different instances and fail to distinguish the different solutions of the same instance, which limits them to guide the choice of the search paradigm in each step in meta-heuristic methods. Despite the above limitations, the learning-based methods have shown the great potential for solving large-scale practical problems.

Hence, in this paper, we propose a Multi-Graph attributed Reinforcement learning-based Optimization algorithm (MGRO) to solve the practical large-scale HFSP, incorporating the generalization ability of learning algorithms with the classic search paradigm of traditional algorithms. Specifically, we first present a Markov Decision Process (MDP) formulation for HFSP, where the actions are different existed search paradigms, and the states are captured by reformulating the Gantt chart into the multi-graph-structured data. The powerful multi-graph based representation provide MGRO with the capability to distinguish the different solution of the same instance. Then we design an attention-based weighted pooling method to draw global logic dependencies between graphs and capture the relations across the entire stage sequence regardless of their temporal distances. In addition, we propose a reward shaping method to learn the combinational usage of search paradigms, which can accelerate the convergence of the model. The model was trained by reinforcement learning-based algorithm without supervision. The strong generalization ability brought by learning mechanism helps MGRO learn to choose classic search operators without expert knowledge not only at different instances but also at each step in the iterated improving process. MGRO can offer more detailed guidance and get a better solution than *end-to-end* learning-based methods. Moreover, MGRO has a more flexible policy for choosing existed search paradigm and higher search efficiency than traditional meta-heuristic algorithms and can *learn to improve* on the basis of arbitrary solutions.

The contributions made in this paper are summarized as follows:

- To the best of our knowledge, we are the first to incorporate learning method with HFSP in a *learn-to-improve* way from a multi-graph perspective, where the policy network trained by reinforcement learning (RL) serves as the search guider to improve the quality of solutions.
- We reformulate the Gantt chart decoded by the specific solution into a multi-graph-structured data to capture the raw features and latent relations.
- We deploy the proposed algorithm, MGRO, in the warehouse control system of the specific Huawei automated warehouse. The results demonstrate the effectiveness and generalization ability of the MGRO by achieving better performance than other existing baselines.

2 RELATED WORK

2.1 Traditional methods for HFSP

The variants of the HFSP concerning the scheduling environment, different objective functions and the solution techniques can be found in review papers [21]. The common methods to deal with the HFSP are the exact algorithm and the heuristic-based algorithm. Exact methods contain branch-and-bound [2], branch-and-cut [15], etc. However, due to the NP-hardness of the HFSP, exact methods are still difficult to solve medium and large instances and are too complex for practical problems. Even an instance with 15 jobs cannot be solved during our testing within one hour by the commercial solver GUROBI [18]. Heuristic-based methods include heuristic algorithms and meta-heuristic algorithms. The heuristic method[19] is designed on the specific characteristics of the problem and construct a satisfactory solution directly without improving procedure.

This class of method is fast, but the quality of the obtained solution is relatively poor. There are many meta-heuristic methods recently, such as Tabu Search Algorithm[5], Genetic Algorithm[1] and Artificial Bee Colony Algorithm[3]. The most widely used framework of meta-heuristic is Iterated Greedy (IG)[20], which will serve as one of our baselines in subsequent numerical experiments. IG design some useful search paradigm to seek a better solution and improves iteratively through search on the basis of the initial solution obtained by NEH[17] or other heuristic methods. The disadvantage of IG is that the feature and characteristics of problem and solution are not captured, which means missing the guidance to the direction and the pattern of search. Although the framework of IG is adaptive to almost all HFSP instances, the search paradigm in it still needs expert experience to be designed manually specifically for each individual instance or specific distribution of instances and fixed the same during the entire iterative improving process.

In conclusion, traditional methods have some deficiencies that make them difficult to be applied for practical large-scale HFSP: 1) The design of these methods heavily relies on domain and expert knowledge; 2) The relatively fixed and rigid search pattern result in the low search efficiency for extremely diverse distribution data.

2.2 Learning-based Method for Scheduling

Recently, there are many research works on applying learning methods for scheduling problems. Same with two main classes of traditional methods, existing learning-based methods for scheduling fall into two categories. The first category is *end-to-end* approach by directly constructing near-optimal solutions from input by a learning method[8, 12, 14, 23, 27], which can be considered to incorporate learning method with heuristic methods. In [27], an end-to-end deep reinforcement learning method is proposed to learn priority dispatching rule (PDR) in JSSP automatically. The second category is *learn-to-improve* approach by learning to perform designed search pattern on the basis of the initial solution to improve the quality of solution in an iterated paradigm, which can be regarded as learning methods incorporate with meta-heuristics algorithms. [4] applied RL to learn local search heuristics to improve on the basis of initial solution for solving job scheduling and resource management problem with Directed Acyclic Graph (DAG) representing the states, which is limited by generalization ability to arbitrary numbers of jobs and resources. Most of these works mainly focus on job shop scheduling problem (JSSP) or DAG-based resource scheduling problem, belonging to the most basic problems in the scheduling region. Compared with JSSP, HFSP is more complicated in the decoding process due to the factors of multi-stage and hybrid scheduling way, and it remains challenging to design effective representation and learning mechanisms. [13] design some statistical indicators calculated by some designed rules such as the average completion time or average utilization ratio of the machine to represent the states to solve arbitrary scales of problems, which may cause redundancy and lose some potentially vital information. [11] proposed a bi-level framework to solve HFSP by Pointer Network, which will serve as one of the baselines in this paper. The limitation is that state features are just denoted as the temporal sequence of jobs, which makes the representation of different solutions similar and even nearly identical.

These works belong to the *end-to-end* paradigm, which often fails to satisfy the demands of real industrial scenarios. The *learn-to-improve* paradigm consumes more computational time but can obtain higher quality solutions. Even a tiny quality improvement can bring huge economic benefits, especially in large-scale practical scenarios. To the best of our knowledge, we are the first to combine learning method with HFSP in a *learn-to-improve* way.

3 PROBLEM FORMULATION

Similar to the simple example illustrated in Figure 1, there are many complicate scenarios with more procedures in Huawei's logistics supply chain. The characteristic shared in common is that there are many machines running in parallel during each procedure and both the specific deployment time sequence of raw materials and the allocation of machines directly determine the work efficiency and completion time. Based on the production status and historical data, we make some reasonable assumptions: 1) The transportation time between different procedures or stages is constant so that it can be considered into the processing time of each individual procedure. 2) The raw materials in the inventory meet the requirements of all orders. Under the above assumptions, the order scheduling problem in the automated warehouse can be modeled as the hybrid flow shop scheduling problem (HFSP).

Table 1: Problem notation.

Sets	
J	Set of jobs $\{j_1, j_2, ..., j_N\}$
I_s	Set of machines at stage $s \in S$
S	Set of stages $\{s_1, s_2, ..., s_H\}$
Parameters	
P_{sj}	Processing time of job $j \in J$ at stage $s \in S$
Q	A very large number
j_0	Label of the first virtual job on each machine
j_{N+1}	Label of the last virtual job on each machine
s_0	Label of the first virtual stage
Decision variables	
x_{ijm}	1 if job i precedes job j on machine m, 0 otherwise
C_{sj}	The completion time of job j in stage s
Z	The makespan

In the HFSP, there are H stages in series, where $H > 1$. A set of N jobs, each of which must be sequentially processed in H stages following the same production flow: stage s_1, stage s_2, ..., stage s_H. Each job $j \in J$ has an uninterrupted and non-negative processing time p_{sj} in stage $s_i, s_i \in S$. There are $|I_s| \geq 1$ identical parallel machines in each stage $s_i \in S$, where $|I_s| > 1$ at least one stage. As parallel machines within a stage are identical, a job could be processed by any machine $m \in I_s$ at stage s_i. At a time, a machine could process only one job, and a job could be processed at time zero and job preemption is not allowed. The travel time between consecutive stages and setup times are included in the processing of jobs at the corresponding stage. The problem data is assumed to be deterministic and known in advance. The HFSP aims to find a schedule that optimizes a given objective. In this study, we consider the most common objective: minimizing the maximum completion time (makespan). With the notations provided in Table1, the mixed

integer programming (MIP) model of the HFSP with makespan minimization objective is given as follows:

$$minimize \quad \max_{j \in J} C_{s_H j} \tag{1}$$

s.t.

$$\sum_{m \in I_s} \sum_{\substack{i \in J \cup \{j_0\} \\ i \neq j}} x_{ijm} = 1; \forall j \in J, s \in S \tag{2}$$

$$\sum_{m \in I_s} \Big(\sum_{\substack{i \in J \cup \{j_0\} \\ i \neq j}} x_{ijm} - \sum_{\substack{k \in J \cup \{j_{N+1}\} \\ k \neq j}} x_{jkm} \Big) = 0; \forall j \in J, s \in S \tag{3}$$

$$\sum_{i \in J} x_{j_0 im} \leq 1; \forall m \in I_s, s \in S \tag{4}$$

$$C_{s_k j} \geq C_{s_{k-1} j} + P_{s_k j}; \forall j \in J, k \in \{1, 2, ..., H\} \tag{5}$$

$$C_{s_0 j} = 0; \forall j \in J \tag{6}$$

$$C_{sj} \geq C_{si} + P_{sj} - Q \times (1 - \sum_{m \in I_s} x_{ijm}); \forall i, j \in J, i \neq j, s \in S \tag{7}$$

$$0 \leq C_{sj} \leq Z; \forall j \in J, s \in S \tag{8}$$

The makespan as the objective is shown in (1). Constraint (2) ensures that every job is processed by only one machine in each stage. Constraint (3) together with constraint (4) guarantees that no machine can process more than one job at a time. The precedence constraint is satisfied by constraint (5) and (6). Job completion time is calculated by constraint (7) and the makespan is captured via constraint (8).

4 METHOD

The overall framework is illustrated in Figure 2. We incorporate the learning representation with powerful search framework of the traditional algorithm. For a specific HFSP instance, we start with an initial solution which can be generated by a random way or a heuristic algorithm such as NEH [17]. Then we extract useful features from the Gantt chart after multi-graph reformulation and attention-based weighted pooling. The features are fed into a graph convolutional network trained by reinforcement learning approach without supervision to decide which search paradigm to choose in the current situation. We design some search operator as different search paradigm to adapt to different instances and solutions. The solution is updated following the selected search operator and is fed into a new round of iterative process as the updated solution. The optimal solution obtained in the iterative process serves as the final solution until the terminal criterion is reached. After the iterative process finishes, rewards are assigned to operators selected in each step following the proposed reward shaping approach. The model can learn which operator to choose in each step according to a specific instance and current solution to minimize the objective of the final solution.

In this section, we explain the three key components of our method. Firstly, we formulate the traditional iterated greedy framework into Markov Decision Process with search operators as actions. Secondly, the decoded Gantt chart is reformulated as multi-graph-structured data to facilitate the extraction of information. Thirdly, we propose an attention-based weighted pooling method to capture the dependencies and latent relations between graphs and a reward shaping approach to accelerate the learning of combinational usage of search operators for long-term reward.

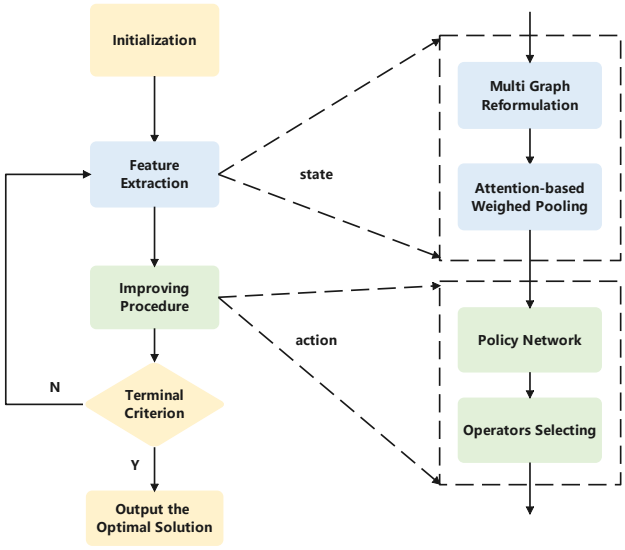


Figure 2: The overall framework of MGRO.

4.1 Markov Decision Process Formulation

State. It is natural to use the solution on each step as the current state of the MDP. In a traditional HFSP algorithm, a solution is represented by a permutation of n jobs denoted by $\pi = (1, 2, \ldots, n)$. To decode the solution for a specific problem, each job is arranged into machines by priority rules. Jobs are scheduled to the next stage following the sequence of finishing time of the last stage, which are called first available machine (FAM) rule and first in first out (FIFO) rule respectively[21]. This type of solution representation can satisfy the unique correspondence with the scheduling situation after decoding, but it contains no useful information and features. To overcome this obstacle, we choose the Gantt chart decoded by the solution as the state, which contains wealthy enough information and features to utilize. The details of how we extract features will be introduced in section 4.2 and 4.3.

Action. In the *learning to improve* framework, our action is used to obtain an updated solution in each step, which is to determine the search operator and the search scale. The detailed description is provided in Appendix B.

Terminal Criterion. The quality of a solution often gets better with more computational time and cost when the iterated steps get larger. For fairness, we set the maximum number of iterated steps, and the improvement process is terminated once the trajectory length reaches the threshold.

Reward. Compared with the traditional reward design method of assigning the raw immediate return, we propose a reward shaping method based on modified local minimum values, which concentrates on long term reward of combinational usage of operators rather than the raw immediate reward of search operator reward in each step. The proposed reward shaping can accelerate the learning to combinational usage of operators and improve the performance

of the model after convergence. The details of reward shaping will be introduced in section 4.4.

4.2 Reformulation Into Multi Graph

Once the instance and solution are given, the Gantt chart can be decoded uniquely with all the information about the raw feature of the state. The traditional features are the statistics calculated by some designed rules such as the average completion time, average waiting time, average idle time or average utilization ratio of the machine, which may cause redundancy and lose some potentially vital information. To overcome the obstacle, we model the Gantt chart into multi-graph-structure data, retaining enough raw features to be aggregated by GNN with strong distinguish power.

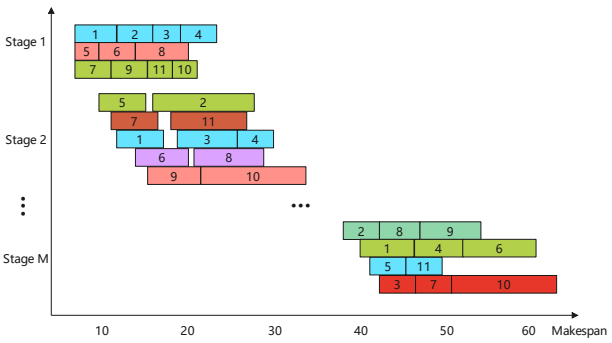
Graph Representation Formulation. We assume each job is a node in the graph, and the edges are sets of sequential or temporal relationships between jobs. This view offers high flexibility to encode different solution and instance properties as we can decide different connectivity structures between the node points. First, we separate each stage in the Gantt chart and focus on the assignment of jobs in each machine in the stage. If a batch of jobs is assigned to the same machine on the same stage, then their schedule order sequence determines the connection of these job nodes. If job i and job j are two jobs allocated adjacently and job i is assigned after j, then the directed edge e_{ij} between job node V_i and V_j is connected. The raw information and topology are included in the primitive embedding for nodes or edges, respectively. Taking Figure 3(a) as a example, there are five machines in stage 2 and the job sequences in each machine of stage 2 are [5,2], [7,11], [1,3,4], [6,8], [9,10] respectively. The graph corresponding to stage 2 is Graph 2 in Figure 3(b). As shown in Figure 3(b), every single graph shares the same set of nodes, but the difference between edges is quite huge. Nodes of the same colour indicate that they are all assigned to the same machine on this specific stage. In other words, the number of colours indicates the number of machines in the specific stage, and nodes of different colours are not be connected by edges.

Let $\mathbf{G} = \{\mathcal{G}_k = (\mathcal{V}, \mathcal{E}_k), k \in \{1, 2, \ldots, M\}\}$ be the overall graphs set for HFSP, where $\mathcal{G}_k$ denotes the graph in stage k and M denotes the number of stages. $\mathcal{V}$ stands for the nodes set consisting of J jobs shared by all the stage and its specific graph, and $\mathcal{E}_k$ indicates the edges set for stage k, which varies on different stages. Now we concentrate on the single graph $\mathcal{G}_k$ of the specific stage k. We design 6 representative feature for job node $V_i \in \mathcal{V}$ in stage k , as follows: 1) the idle time in this stage I_{ik} 2) the waiting time in this stage W_{ik} 3) the arriving time in this stage A_{ik} 4) the starting time in this stage S_{ik} 5) the completion time in this stage C_{ik} 6) the processing time in this stage P_{ik} defined in previous text. The elements for both embeddings are normalized. It is noted that the first five feature is the specific feature corresponding to a specific solution and the last feature is the raw feature of the instance which keeps fixed when solutions change. The details of features are illustrated in Appendix E. Let h_i^k denotes the node feature associated with node v_i as

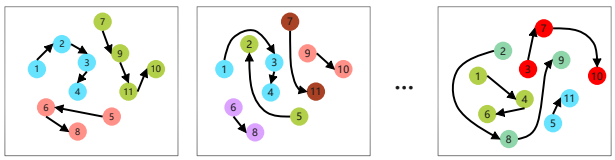
$$h_i^k = C(I_{ik}, W_{ik}, A_{ik}, S_{ik}, C_{ik}, P_{ik}) \tag{9}$$

where C is the concatenation operator with dimension $d_0 = 6$.

Graph Neural Networks. We apply the most widely used graph neural networks GCN (Graph Convolutional Network)[9], which



(a) Gantt chart for the illustrated example.



(b) Multi graph for the illustrated example.

Figure 3: The reformulation to multi graph.

can scale linearly in the number of graph edges and learn hidden layer representations that encode both local graph structure and features of nodes. The multi-layer node embedding is updated as follows (we omit the normalization coefficient part for brevity):

$$(m_i^k)^{l+1} = \sum_{u \in \mathcal{N}^k(i)} (h_u^k)^l \tag{10}$$

$$(h_i^k)^{l+1} = \sigma(W^l((m_i^k)^{l+1} + (h_i^k)^l)) \tag{11}$$

$$\hat{h_k} = Readout(\{(h_i^k)^{l+1} | i \in |\mathcal{V}|\}) \tag{12}$$

where we denote by $\mathcal{N}^k(i)$ the set of immediate neighbours of node v_i (can be obtained by $\mathcal{E}_k$) at stage k, by $(m_i^k)^{l+1}$ the aggregated representation of those neighbors, by $\sigma(\cdot)$ an activation function (e,g., element-wise sigmoid or ReLU), and by W^l the matrix of model parameters to be learned by the GCN. The first layer of node embedding is initialized as $(h_i^k)^0 = h_i^k$. The goal of our model is to map graphs into a set of action labels, which means the subsequent task is an unsupervised graph classification problem rather than a node classification problem. So the last layer of GCN uses aggregate function(e.g., average function or max function) for the subsequent graph classification task. In this way, we can obtain the graph feature embedding $\hat{h_k}$ for each single graph $\mathcal{G}_k$ at stage k with a small computational cost.

4.3 Attention-based Weighted Pooling

After feeding a set of graphs $\mathbf{G} = \{\mathcal{G}_k = (\mathcal{V}, \mathcal{E}_k), k \in \{1, 2, \ldots, M\}\}$ into graph neural network, we can obtain a set of graph embedding of all stages $H = [\hat{h_1}, \hat{h_2}, \ldots, \hat{h_M}]$. The number of graphs M varies with the number of stages for each different instance. It means we have to find an effective way to merge the embeddings of all single stages to fixed dimensional embeddings to guide the subsequent

learning task. To overcome this obstacle, we propose the Attention-based Weighted Pooling (ABWP) approach to help pooling the embeddings. In general, the relations given in different graphs may not have the same importance in the global graph, and self-attention mechanism can draw global logic dependencies between stages and capture the relations across the entire stage sequence without their temporal distances.

Multi Self-Attention Layer. After getting the set of multi single stage embeddings H, we feed them into the self-attention layer to better capture the global graph pattern.

$$F = softmax(\frac{(HW^Q)(HW^K)^T}{\sqrt{d}})(HW^V) \tag{13}$$

where the projection matrices $W^Q, W^K, W^V \in \mathbb{R}^{d\times d}$ and d denotes the hidden dimension. However, transmission loss may occur in self-attention operations. After that, we add a residual connection after the feed-forward network, which makes the model easier to leverage low-layer information inspired by [24]. The formulation follows as $E = F + ReLU(MLP(F))$. Moreover, Dropout regularization are applied to alleviate overfitting dilemma. For simplicity, we define the above whole self-attention mechanism as : $E = SAN(H)$.

To derive better feature representations for policy learning, the above self-attention layer are successively processed by L blocks with the same structure but different parameters. We design a multi-layers model which capture different types of features and patterns in the set of graphs to help learn more complex latent pattern inspired by [26]. The 1-st layer is defined as $E^1 = E$. The L-th ($L > 1$) self-attention layer is defined as: $E^{(L)} = SAN(E^{(L-1)})$ where $E^{(L)} \in \mathbb{R}^{M\times d}$ is the final output of the multi-layer attention network and M denotes the number of stages or graphs.

Adaptive Weighted Pooling. Each individual graph embedding has a different importance to global representation. For this purpose, we propose an adaptive weighted pooling approach with the flexibility to learn the weight of each individual graph embedding adaptively and to integrate the information from all graph topologies. $W \in \mathbb{R}^{M\times 1}$ is the trainable weight parameter follows as:

$$W = Softmax(MLP(E^{(L)})) \tag{14}$$

The final embedding after pooling are given as: $\hat{H} = W^T E^{(L)}$.

Then the final embedding is fed into the policy network consisted of basic MLP and softmax layer to produce the probability of selecting the action of the next step:

$$P = softmax(MLP(\hat{H})) \tag{15}$$

The comparisons between our proposed attention-based weighted pooling (ABWP) method and the original pooling method will be provided in ablation studies in section 5.3.2.

4.4 Reward Shaping

Like the general MDPs, each step of the action in the improving process could get an immediate reward related to the difference between the objective value of the previous solution state. But it is absolutely unfair to assign the immediate reward to each action so simply after a trajectory is finished. Some operator actions may get worse solutions, but they may help jump out of the local optimum dilemma in a sense. The original immediate reward gives too much

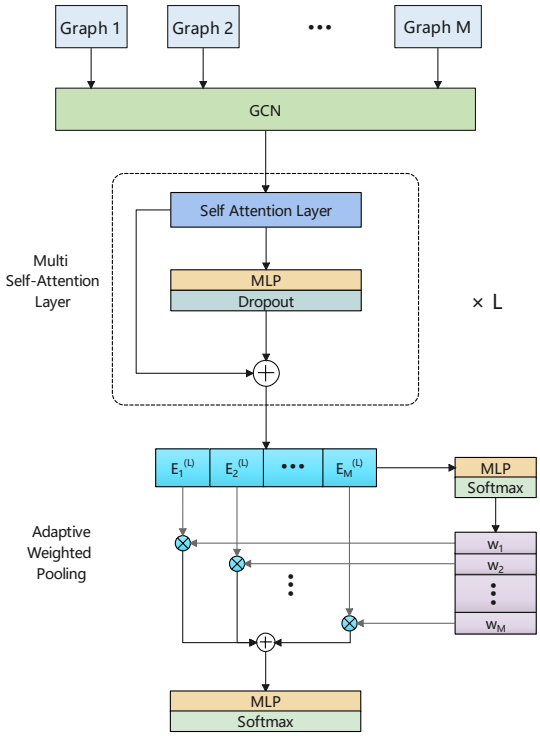


Figure 4: Attention-based Weighted Pooling Architecture.

attention to the return of the single-step and neglects the long term reward. For the reasons mentioned above, we propose a reward shaping method for optimized MDPs, which can be used in the paradigm of *learning-to-improve*. We concentrate on the performance of the combination of operators instead of the short-term reward of a single operator.

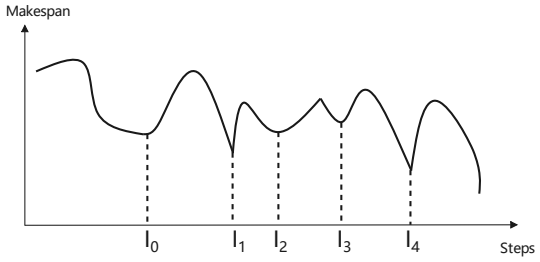


Figure 5: An example of reward shaping.

The property of operators that can accept the worse solution causes the objective values to oscillate frequently. It means local minimum point often appears in our formulated trajectory, which inspires us to divide the trajectory by local minimum points into several sub-segments. The actions taken in the sub-segment can be regarded as the combinational usage of operators. However, a local minimum appears on average of 5 steps and even less in quite many cases. Frequent and trivial division of the trajectory is not suitable for learning the combined usage of operators. We make a tiny but useful modification on the selection of local minimum. The local minimum larger than any local minimum before will be discarded and maintain the $\mathcal{S}$ is monotonically decreasing. For clarity, we

define I_p as the p-th index time step reaching local minimum in the entire trajectory and the corresponding objective value is O_{I_p}. For example, in the instance of Figure5, the original $\mathcal{S} = [I_0, I_1, I_2, I_3, I_4]$, and the original local minimum set is $\mathcal{O} = [O_{I_0}, O_{I_1}, O_{I_2}, O_{I_3}, O_{I_4}]$. But we discard the I_2 and I_3 following the rule mentioned above, the modified local minimum set becomes $\hat{\mathcal{S}} = [I_0, I_1, I_4]$. The remained three local minimum divides the whole trajectory into three sub-segments. We take the improvement in each sub-segment as a local reward in this period. In order to avoid giving original negative rewards to a certain action that may potentially help jump out of the local minimum, we divide the local reward of the entire sub-segment equally to each action on the sub-segment. The shaped reward of each step is followed as:

$$r_i = \frac{O_T - O_1}{T-1} + \frac{O_{I_{p-1}} - O_{I_p}}{I_p - I_{p-1}} \quad for \quad i \in [I_{p-1}, I_p) \tag{16}$$

The reward of each step consists of two parts, one part is the global reward during the entire trajectory, and the other is the local reward of the sub-segment where the step is located.

Hence, when the discount factor $\gamma = 1$, the cumulative reward is $\sum_{i=1}^{T} r_i = 2(O_T - O_1)$. Given the O_1 of the initial solution in the first time step in the trajectory is constant, maximizing the cumulative reward coincides with minimizing the long-term makespan.

5 EXPERIMENTS

We present the experimental results in this section. The evaluations are performed both on public HFSP benchmarks and real industrial scheduling datasets in Huawei Supply Chain Business Unit.

5.1 Experimental Setup

Dataset. We evaluate our method on instances of various sizes provided in [6], which offers an empirically hard testbed for comparison. 480 instances are generated in the experiments to form the two sets of 240 small and big-size instances. The best-known value for each instance is given to facilitate the comparison. In this dataset, the job number varies from 10 to 240, and the stage number varies from 5 to 20. Each scale contains ten different instances, and the machine number in instances may be different. Then we test the algorithm using the real industrial scheduling dataset in Huawei Supply Chain Business Unit. Scheduling records from January to December 2020 are treated as the real online dataset. The dataset is divided into a different scale of 50, 80, 100, 150, 200, 300, 400 and 500 orders according to the order extraction. It is worth mentioning that the number of jobs contained in the instances varies with the same order size according to different distribution of orders. The scale of jobs reaches 500+ even in the smallest order size of 50, which is more complicate than any instance in standard dataset. Each scale contains 50 different instances for generality.

Baselines. There are many heuristics and meta-heuristics algorithms for HFSP in literature with various performances, and we can not compare with them exhaustively. We choose NEH as our initial solution way and IG as the improved algorithm, which contains iterative disturbance and local search. We additionally introduced an *end-to-end* learning-based approach as a benchmark. It is worth mentioning that we have tested Gurobi[18] experiments and found that results on the standard dataset are almost the same as the best-known results provided in the dataset. However, Gurobi fails to give results within 5 hours and cannot satisfy the execution time limits even in the 50 orders scale instance, the simplest dataset on the large-scale real industrial dataset. For the reasons mention above, we decide not to adopt Gurobi as the baseline.

- **NEH.** We choose NEH[17], one of the well-known and nearly the most widely used heuristic method, as our initialization way. As all schedules must be evaluated, the computational complexity is $O(n^3m)$ [19].
- **BDS.** [11] proposed a novel bilevel deep reinforcement learning scheduler (BDS) to deal with the HFSP in an end-to-end learning way as one benchmark of the RL method, which serves as one of our benchmarks.
- **IG.** Iterated Greedy(IG)[20] is a representative meta-heuristic algorithm with an iterated improvement process framework, which is also the basis of our proposed method.

Models and configurations. We adopt the Proximal Policy Optimization (PPO) algorithm [22], and the details for the model and training configurations is presented in Appendix A.1.

5.2 Results

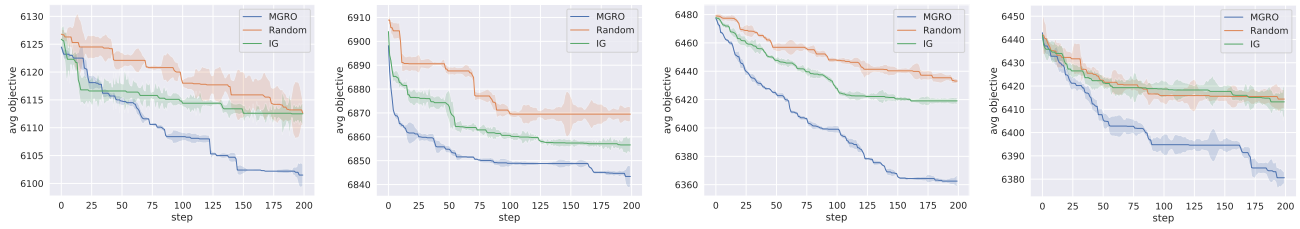
We trained the model following the setting mentioned above on the standard public dataset and real industrial dataset separately. To demonstrate the performance of our model, we compare it with other methods. The experimental results of all method on two types of datasets are illustrated below. The details of settings for all baselines are provided in Appendix C.

Table 2: Results on standard dataset.

Size	NEH	BDS	IG	**MGRO**	Best Known
10	1034.28	924.43	891.63	**865.48**	863.85
15	1213.43	1058.23	1011.53	**980.28**	972.77
20	1315.00	1143.48	1075.82	**1043.40**	1025.93
25	1338.63	1212.20	1117.07	**1081.90**	1055.80
30	1561.08	1378.40	1302.68	**1251.18**	1218.60
35	1693.52	1527.43	1418.70	**1369.88**	1335.93
40	1737.35	1549.80	1458.89	**1442.71**	1373.23
80	2614.83	2439.55	2318.31	**2277.00**	2186.63
120	3684.60	3481.73	3353.29	**3307.87**	3204.48
160	4674.08	4406.08	4298.33	**4243.62**	4130.08
200	5613.40	5357.33	5236.04	**5183.66**	5101.68
240	6800.00	6620.43	6451.70	**6419.14**	6300.50

Table 3: Results on real industrial dataset.

size	NEH	BDS	IG	**MGRO**	Lower Bound
50	1727.6	1807.3	1706.2	**1628.8**	1480.8
80	2025.2	2033.6	1935.2	**1760.4**	1539.6
100	2463.8	2443.8	2333.8	**2109.2**	1918.2
150	4448.2	4520.2	4350	**4174.2**	4057.4
200	6772.4	6902.8	6760	**6418.2**	6376.1
300	8283.6	8420.5	8276.2	**8119.4**	8068.6
400	-	18023.6	11333.4	**10470.6**	10018.5
500	-	49201.3	43546.9	**42628.8**	41322.4



(a) 240-5 (b) 240-10 (c) 240-15 (d) 240-20

Figure 6: Comparison of different selection methods of operators on standard dataset 240.

Table 2 displays the comparisons between methods on standard dataset. It can be observed that MGRO can always get a solution of better quality with higher efficiency of searching. For simplicity, we take the average value of all stages in the same job scale as indicators of performance. The complete and detailed results with different stage scales on the standard dataset are provided in Appendix D.

In the standard dataset, the average gap of MGRO and IG with best-known results is 2.42% and 4.04% respectively, which means that MGRO reduces the gap by 40.18% compared to IG with the same computational time. The results on the real industrial dataset are shown in Table 3. There are no best-known results for comparisons in the real industrial dataset, and we choose to calculate theoretical lower bound as the indicators of dataset difficulty. The details for lower bound is provided in Appendix F. The average gap of MGRO and IG with the lower bound calculated is 3.38% and 7.30% respectively. The performance of MGRO exceeds IG over 53.7%. The average computational time of baselines is 6230s (NEH), 1025s (BDS), 2102s (IG), 2078s (MGRO) respectively on the real industrial dataset. The computational time of BDS is the smallest because of the end-to-end learning-based approach, and the execution time of IG and MGRO is maintained the same.

In the real industrial scenario in Huawei, it usually needs within one hour from the time the daily order information is given to the start of scheduling processing. The algorithm needs to give a sufficiently high-quality solution within the required response time. The quality of the solution directly determines the working hours of workers or machines, which need to be decreased to reduce manufacturing costs. NEH obviously timed out and fails to satisfy the requirements of execution time. Although BDS meets the time requirements, the quality of solutions is relatively poor, which would greatly increase manufacturing costs. As for IG and MGRO, we can set the iterated numbers to control the execution time. MGRO can get better performance than IG with similar search paradigm and same computational time.

5.3 Ablation Study

5.3.1 Does the RL policy network learn how to select operators? We design some basic operators as actions in the MDP to help improve the quality of the solution. But the following experiment shows that not only our operator works but also the policy network learned from our RL method. We choose random operators to take in each step in the MDP and found it performs poorly and the effect of using each operator alone is also relatively not that good. In addition to the previous IG algorithm, we design an algorithm to choose the operators randomly as the benchmark of Random. We use these three algorithms to perform on the standard dataset for fairness. Without the loss of generality, we choose the results on the most complicated instances (job scale reaches 240) to show the comparisons in Figure 6. It can be observed that the performance of selecting operators randomly is nearly equal to or even worse than IG. But the agent trained by the RL policy can learn how to use proper operators, and the performance can outperform IG. The result shows that MGRO takes less time than IG to reach the same quality of the solution especially in large scale dataset.

5.3.2 Does the proposed pooling method based on self-attention mechanism bring benefits? We remove the attention-based weighted pooling (ABWP) from MGRO and replace it with the vanilla mean-pooling method. For reproducibility, we make comparisons on the real industrial dataset and show the results on instances of 500 orders as the most complicated dataset without the loss of generality. The results in Figure 7 verify the contribution of ABWP module and show that ABWP can draw global dependencies and capture the relations across the entire stage sequence regardless of the temporal distances. MGRO can learn the particular weight of each single graph to denotes the importance of the single graph in the whole multi-graph structure. The convergence speed of the model would slow down, and the performance becomes worse when model converges if we ignore the difference between graphs and simply perform mean pooling method on them.

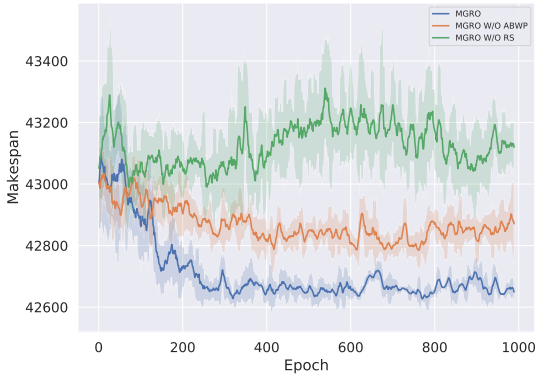


Figure 7: Training curve comparisons on 500 orders.

5.3.3 Does the proposed the reward shaping approach matter? We also verify the effectiveness of our proposed reward shaping method on real industrial dataset 500 orders as an example without the loss of generality. The result in Figure 7 demonstrates that results become worse and extremely oscillating if the only simple vanilla immediate reward is applied. The vanilla immediate reward will discourage the usage of operators with negative improvement, which

may help escape out of the local minimum dilemma potentially. The reward shaping method concentrates on the overall contribution of the combined usage of operators and can capture the fair usage way of operators. In addition, the comparisons show that the reward shaping method can reduce variance and improve training stability.

5.3.4 Does MGRO have strong generalization ability ? We put the model trained on a small scale data set on a large-scale problem to validate for testing the generalization ability of MGRO. Without the loss of generality, we directly use the policies trained on $20-10$ and $120-15$ dataset to solve $120-15$ and $240-20$ dataset. The results are summarized in Table 4, where the result for each instance size is averaged by ten times of evaluation. As shown in the table, both our policies trained on much smaller instances perform reasonably well on the large-size ones and the results outperform other baselines. Moreover, we tried to directly apply the model trained on the standard dataset on the real industrial dataset, and the results are relatively good. The gap between the model trained on the standard dataset and real industrial dataset is reasonable, which may be caused by the difference of the distribution or the pattern of the data. This observation shows that our method can extract knowledge from the graph network of small size instances and apply it to reasoning on large-sized instances, which is a desirable property in practical application.

Table 4: Generalization Ability Test.

Dataset Tested	Dataset Trained	Makespan	IG
120-15	20-10	3379.90	3419.63
	120-15	**3350.57**	
240-20	20-10	6380.76	6417.27
	240-20	**6379.53**	
200 orders	Standard Dataset	6445.9	6760.3
	50 orders	6430.3	
	200 orders	**6418.2**	
500 orders	Standard Dataset	42731.2	43546.9
	200 orders	42837.5	
	500 orders	**42628.8**	

6 CONCLUSIONS

In this paper, we propose a multi-graph attributed reinforcement learning-based optimization algorithm (MGRO) to solve the practical large-scale hybrid flow shop scheduling problem (HFSP) with highly efficient searching policy and strong generalization ability across various problem distributions and scales. We reformulate the Gantt chart decoded by the instance solution into the multi-graph-structured data and apply GNN to capture the feature both from the solution and the instance. Numerical experiments show that MGRO can learn how to search properly and adjust the search direction flexibly under the hugely diverse distribution of daily orders and obtain good-quality solutions within a limited time in large scale practical scenarios. We evaluate our proposed method on the publicly available dataset and real industrial dataset from the Huawei Supply Chain and achieve demonstrably superior performance over existing baselines on both datasets. For future work, an intriguing direction is to generalize the problem by considering additional constraints, multiple objectives, and uncertainty.

REFERENCES

[1] A Adressi, S Hassanpour, and V Azizi. 2016. Solving group scheduling problem in no-wait flexible flowshop with random machine breakdown. *Decision Science Letters* 5, 1 (2016), 157–168.

[2] Ahmet Bolat, Ibrahim Al-Harkan, and Bandar Al-Harbi. 2005. Flow-shop scheduling for three serial stations with the last two duplicate. *Computers & Operations Research* 32, 3 (2005), 647–667.

[3] Fei Chen, Song Wu, Hai Jin, Yin Yao, Zhiyi Liu, Lin Gu, and Yongluan Zhou. 2017. Lever: towards low-latency batched stream processing by pre-scheduling. In *Proceedings of the 2017 Symposium on Cloud Computing*. 643–643.

[4] Xinyun Chen and Yuandong Tian. 2018. Learning to perform local rewriting for combinatorial optimization. *arXiv preprint arXiv:1810.00337* (2018).

[5] Atabak Elmi, Maghsud Solimanpur, Seyda Topaloglu, and Afshin Elmi. 2011. A simulated annealing algorithm for the job shop cell scheduling problem with intercellular moves and reentrant parts. *Computers & industrial engineering* 61, 1 (2011), 171–178.

[6] Victor Fernandez-Viagas and Jose M Framinan. 2020. Design of a testbed for hybrid flow shop scheduling with identical machines. *Computers & Industrial Engineering* 141 (2020), 106288.

[7] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428* (2019).

[8] Helga Ingimundardottir and Thomas Philip Runarsson. 2018. Discovering dispatching rules from data using imitation learning: A case study for the job-shop problem. *Journal of Scheduling* 21, 4 (2018), 413–428.

[9] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[10] Geun-Cheol Lee. 2009. Estimating order lead times in hybrid flowshops with different scheduling rules. *Computers & Industrial Engineering* 56, 4 (2009), 1668–1674.

[11] Longkang Li, Hui-Ling Zhen, Mingxuan Yuan, Jiawen Lu, Jia Zeng, Jun Wang, Dirk Schnieders, et al. 2020. Bilevel Learning Model Towards Industrial Scheduling. *arXiv preprint arXiv:2008.04130* (2020).

[12] Chun-Cheng Lin, Der-Jiunn Deng, Yen-Ling Chih, and Hsin-Ting Chiu. 2019. Smart manufacturing scheduling with edge computing using multiclass deep Q network. *IEEE Transactions on Industrial Informatics* 15, 7 (2019), 4276–4284.

[13] Shu Luo. 2020. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing* (2020), 106208.

[14] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*. 270–288.

[15] B Naderi, Sheida Gohari, and M Yazdani. 2014. Hybrid flexible flowshop problems: Models and solution methods. *Applied Mathematical Modelling* 38, 24 (2014), 5767–5780.

[16] Bahman Naderi and Rubén Ruiz. 2010. The distributed permutation flowshop scheduling problem. *Computers & Operations Research* 37, 4 (2010), 754–768.

[17] Muhammad Nawaz, E Emory Enscore Jr, and Inyong Ham. 1983. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* 11, 1 (1983), 91–95.

[18] Gurobi Optimization. 2014. Inc.,"Gurobi optimizer reference manual," 2015.

[19] Quan-Ke Pan, Ling Wang, Jun-Qing Li, and Jun-Hua Duan. 2014. A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. *Omega* 45 (2014), 42–56.

[20] Rubén Ruiz and Thomas Stützle. 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research* 177, 3 (2007), 2033–2049.

[21] Rubén Ruiz and José Antonio Vázquez-Rodríguez. 2010. The hybrid flow shop scheduling problem. *European journal of operational research* 205, 1 (2010), 1–18.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[23] Penghao Sun, Zehua Guo, Junchao Wang, Junfei Li, Julong Lan, and Yuxiang Hu. 2020. Deepweave: Accelerating job completion time with deep reinforcement learning-based coflow scheduling. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20.*

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[25] Bagas Wardono and Yahya Fathi. 2004. A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. *European Journal of Operational Research* 155, 2 (2004), 380–401.

[26] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. 2019. Graph Contextualized Self-Attention Network for Session-based Recommendation.. In *IJCAI*. 3940–3946.

[27] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. 2020. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. *Advances in Neural Information Processing Systems* 33 (2020).

A TRAINING DETAILS OF MODEL

A.1 Learning Algorithm

Our proposed algorithm follows as : We train the policy network using Proximal Policy Optimization (PPO)[22], which is an actor-critic algorithm. The actor refers to the policy network π_θ, the critic v_ϕ shares the same attention-based multi graph network with the actor. To boost the training, we use N actors, each solves one HFSP instance drawn from the dataset distribution $\mathbb{D}$. So we use all data collected by the N actors to update. Here we provide details of our training algorithm in terms of pseudo-code in Algorithm 1.

Algorithm 1 PPO for HFSP

Input: actor network π_θ and behaviour actor network $\pi_{\theta_{old}}$, with trainable parameters $\theta_{old} = \theta$; critic network v_ϕ with trainable parameters ϕ; number of training steps U; update epoch K; trajectory max length T; policy loss efficient c_p; value function loss coefficient c_v; entropy loss efficient c_e; clipping ratio ϵ.
Initialize π_θ, $\pi_{\theta_{old}}$, and v_ϕ;
for each u = 1, 2, ..., U **do**
  Draw N HFSP instances from $\mathbb{D}$;
  for n = 1, 2, ..., N **do**
    for t = 0, 1, 2, ... **do**
      Sample $a_{n,t}$ based on $\pi_{\theta_{old}}(a_{n,t}|s_{n,t})$;
      Receive vanilla reward $r_{n,t}$ and next state $s_{n,t}$;
      ratio $r_{n,t}(\theta) = \frac{\pi_\theta(a_{n,t}|s_{n,t})}{\pi_{\theta_{old}}(a_{n,t}|s_{n,t})}$
      if $s_{n,t}$ *is terminal* **then**
        update the vanilla reward $r_{n,t}$ to shaped reward $\hat{r}_{n,t}$
        update the advantage value $\hat{A}_{n,t} = \hat{r}_{n,t} - v_\phi(s_{n,t})$
        break;
      end if
    end for
    $L_n^{CLIP}(\theta) = \sum_0^t min(r_{n,t}(\theta)\hat{A}_{n,t}, clip(r_{n,t}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{n,t})$;
    $L_n^{VF}(\phi) = \sum_0^t (v_\phi(s_{n,t}) - \hat{A}_{n,t})^2$;
    $L_n^s(\theta) = \sum_0^t S(\pi_\theta(a_{n,t}|s_{n,t}))$, where $S(\cdot)$ is entropy;
    Aggregate losses: $L_n(\theta,\phi) = c_p L_n^{CLIP}(\theta) - c_v L_n^{VF}(\phi) + c_e L_n^s(\theta)$;
  end for
  for k = 1,2,..., K **do**
    update θ, ϕ with cumulative loss by Adam optimizer:
    $\theta, \phi = argmax(\sum_{n=1}^N L_n(\theta,\phi))$
  end for
  $\theta_{old} \leftarrow \theta$
end for
Output:

A.2 Implementation Details

Experiments were performed in PyTorch using code from the PyTorch Geometric Library[7]. For GCN and pooling layers, we set the hidden dimension to 64, the number of GCN layers to 3, the number of SAN layer to 4. Too many SAN layers will decrease the performance of the model because using more layers would make the model easier to lose low-layer information. For PPO, we set the epochs of updating network to 2, the clipping ratio ϵ to 0.2, and the coefficient for policy loss, value function and entropy to 2, 1, and 0.1, respectively. For training, we choose ReLU as the activation function and we use Adam optimizer with constant learing rate $l_r = 2e-4$. The hardware we use is a server group of Ubuntu 18.04.4 LTS with 111 Intel(R) Xeon(R) Platinum 8180M CPUs@ 2.50GHz and 2 Nvidia Tesla V100 GPU.

B DETAILS OF ACTIONS

The search operator and search scale are determined by our action. The combined choices contain two aspects of information: which direction to search and how deep the search is, which affect the final obtained solution. Inspired by the iterated greedy (IG) algorithm, we design three operators: 1) pick *ss* job nodes from all jobs randomly 2) pick *ss* job nodes randomly from the jobs which the total idle time at all stages ranks top 50% 3) pick *ss* job nodes randomly from the jobs which the total waiting time at all stages ranks top 50%. *ss* means the search scale. Three levels are set for search scale: 1) $ss = 2.5\% \cdot N$ 2) $ss = 5\% \cdot N$ 3) $ss = 10\% \cdot N$, where N is jobs number.

The selected *ss* jobs are removed from the current solution and are reinserted into the partial solution in a sequential manner. Firstly, for each job one by one in the selected job list, reinsert it into all possible positions of the partial solution to obtain a set of new solutions. Secondly, the best feasible one is used to replace the current partial solution. Thirdly, repeat the above with the next job until no job in the selected list is unconsidered.

Note that the updated solution is regarded as the next state regardless of whether it becomes better. The newly generated solution may become much worse and have to improve from a poor solution again if *ss* is large, but the improvement may not appear if *ss* is too small. The different choice of action is helpful to balance exploration and exploitation. It is worth mentioning that the same operator may have a huge effect varies with different instances or even in different solutions of the same instance. There exists no operator that can perform well in all instances. With expert knowledge or grid search, traditional IG framework can change the operator on different instances but still keeps the same operator during the entire iterative improvement process, causing low search efficiency.

C SETTINGS FOR COMPARISONS

If allowing a sufficiently long calculation time and a significantly large search cost, both IG and MGRO can obtain very high-quality solutions. But this comparison is of little significance because it is impossible to have such conditions to improve the quality of the solution in large-scale actual scheduling problems. Therefore, we reduce the search cost and calculation time limit of IG and MGRO to compare their effects. The scale and difficulty of the standard dataset and real industrial dataset are different, which makes the computational time varies. The computational time cost of a search operator executed takes just 0.03s on a standard dataset but the time reaches at least 60s on the real industrial dataset with the acceleration of parallel processing. We make some adjustments for the reasons mentioned above but not harm fairness. For the standard dataset with relatively fast iterated speed, we set a larger number of iteration steps to demonstrate the potential of baselines. For the

real industrial dataset, the terminal criterion is set as 30 to meet the real-time requirements. As mentioned above, the complexity of NEH reaches $O(n^3m)$. This scale of data is hard to be initialized by NEH and NEH completely failed at instances with order scale larger than 400. So we choose a random initialization and perform MGRO to improve the initial solution and maintain MGRO and IG the same number of iterated steps for fairness.

D COMPLETE RESULTS ON STANDARD DATASET

Table 5 and Table 6 display the comparisons between methods on small size and large size standard dataset respectively. Each job scale contains four sets of stages, which are 5, 10, 15, 20, respectively. Each scale contains ten instances for generality.

E DETAILS OF FEATURES

Here we introduce the specific meaning of selected features. The arriving time is the time job finishes at the previous stage and arrive at the current stage. The starting time is the time job starts to be processed at the current stage. The completion time is the time job finishes at the current stage. The arriving time of all jobs on the first stage is set at 0. For the same job, the completion time of previous time equals to the arriving time of the current stage regardless of the transporting time or manual intervention. The processing time is given with the instance and keeps fixed with different solutions.

As for the idle time I_{ik}, it means the idle time of the assigned machine at stage k before it starts processing job i. The waiting time W_{ik} means the time that job i arrives at stage k and waits for the assigned machine to finish the job currently being processed. To make it clear, we still take Figure 3(a) as an example. Job 2 is assigned to the first machine in stage 2 and the machine has finished job 5 already and has no job to process until job 2 leaves stage 1 and arrives in stage 2. The idle time for the machine is recorded as the idle time I_{22} caused by job 2 in stage 2. Similarly, job 4 is assigned at the third machine in stage 2 and the machine is still processing job 3 when job 4 leaves stage 1 and arrives in stage 2. Job 4 has to wait for the completion of job 3 before it starts and the waiting time is recorded as W_{42} of job 4 in stage 2. Note that W_{ik} and I_{ik} cannot be larger than 0 at the same time because the machine waiting for unfinished job and job waiting for the non-idle machine cannot happen at the same time for a pair of job and its assigned machine.

Table 5: Results on small size dataset.

Size Job	Stage	NEH	BDS	IG	**MGRO**	Best Known
10	5	501.4	449.9	427.1	**412.8**	411.9
	10	974.2	846.7	811.2	**793.2**	793
	15	1112.4	975.3	953.6	**928.5**	925.3
	20	1549.1	1425.8	1374.6	**1327.4**	1325.2
15	5	622.6	513.1	493.0	**473.6**	469.6
	10	1128.6	953.6	901.6	**881.2**	875.5
	15	1333.8	1211.5	1150.1	**1114.5**	1106.8
	20	1768.7	1554.7	1501.4	**1451.8**	1439.2
20	5	804.4	652.1	604.6	**587.5**	579.8
	10	1184.1	968.9	925.4	**899.9**	882.3
	15	1549.7	1378.6	1288.2	**1248.2**	1226.8
	20	1721.8	1574.3	1485.1	**1438.9**	1414.8
25	5	913.3	776.6	725.7	**706.0**	697.4
	10	1180.4	1074.3	986.4	**956.8**	937.3
	15	1520.2	1377.8	1273.6	**1226.1**	1190.9
	20	1740.6	1620.1	1482.6	**1438.7**	1397.6
30	5	982.2	805.9	759.8	**736.8**	723.3
	10	1290.2	1132.7	1091.9	**1039.0**	1011.8
	15	1774.6	1590.4	1495.3	**1434.0**	1395.9
	20	2197.3	1984.6	1863.7	**1794.9**	1743.4
35	5	1142.5	995.6	917.0	**897.4**	886.5
	10	1553.6	1393.5	1287.3	**1242.7**	1218.2
	15	1871.3	1734.8	1597.5	**1542.3**	1502
	20	2206.7	1985.8	1873.0	**1797.1**	1737

Table 6: Results on big size dataset.

Size Job	Stage	NEH	BDS	IG	**MGRO**	Best Known
40	5	1150.4	1002	944.17	**932.17**	909
	10	1615.3	1416.2	1317.00	**1315.40**	1257.8
	15	1880.9	1651.6	1585.13	**1558.37**	1474.8
	20	2302.8	2129.4	1989.27	**1964.90**	1851.3
80	5	2231.5	2109	2057.63	**2040.43**	2037.1
	10	2317.6	2065.7	1984.43	**1950.13**	1855.9
	15	2859.9	2684.4	2513.27	**2457.10**	2342.8
	20	3050.3	2899.1	2717.90	**2660.33**	2510.7
120	5	3138.9	3047.3	2966.70	**2953.07**	2950.8
	10	3509	3391.4	3203.40	**3171.70**	3143.2
	15	3862.8	3569.7	3419.63	**3350.57**	3180.4
	20	4227.7	3918.5	3823.43	**3756.13**	3543.5
160	5	4351.2	4234.4	4171.47	**4153.33**	4151.7
	10	4659.5	4450.8	4384.53	**4316.23**	4303
	15	4853.1	4448.3	4265.00	**4213.70**	4007.4
	20	4832.5	4490.8	4372.33	**4291.23**	4058.2
200	5	5286.7	5163	5096.97	**5082.27**	5079.4
	10	5570.2	5356.8	5225.23	**5172.87**	5163.6
	15	5793.9	5522.9	5369.37	**5289.23**	5232.6
	20	5802.8	5386.6	5252.60	**5190.27**	4931.1
240	5	6326.4	6182.9	6113.13	**6099.37**	6096.7
	10	7173.8	7018.1	6858.47	**6840.21**	6814
	15	6754.6	6691	6420.93	**6360.17**	6299.8
	20	6945.2	6589.7	6414.27	**6379.53**	5991.5

F LOWER BOUND

We choose to calculate the lower bound because of the lack of public best-known solution on the real industrial dataset used in the paper. The lower bound is denoted as

$$LB = max_{l \in 1,\dots,L}\{\alpha_l + T_l + \beta_l\} \tag{17}$$

where α_l denotes the average idle time of the machines at stage l before they start processing a job, T_l denotes the average workload of each machine at stage l, and β_l denotes the average idle time of machines at stage l after they complete the last work. For details, please refer [25].