Residual Scheduling: A New Reinforcement Learning Approach to Solving Job Shop Scheduling Problem

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Job-shop scheduling problem (JSP) is a mathematical optimization problem widely used in industries like manufacturing, and flexible JSP (FJSP) is also a common variant. Since they are NP-hard, it is intractable to find the optimal solution for all cases within reasonable times. Thus, it becomes important to develop efficient heuristics to solve JSP/FJSP. A kind of method of solving scheduling problems is construction heuristics, which constructs scheduling solutions via heuristics. Recently, many methods for construction heuristics leverage deep reinforcement learning (DRL) with graph neural networks (GNN). In this paper, we propose a new approach, named residual scheduling, to solving JSP/FJSP. In this new approach, we remove irrelevant machines and jobs such as those finished, such that the states include the remaining (or relevant) machines and jobs only. Our experiments show that our approach reaches state-of-the-art (SOTA) among all known construction heuristics on most well-known open JSP and FJSP benchmarks. In addition, we also observe that even though our model is trained for scheduling problems of smaller sizes, our method still performs well for scheduling problems of large sizes. Interestingly in our experiments, our approach even reaches zero gap for 49 among 50 JSP instances whose job numbers are more than 150 on 20 machines.

1 Introduction

The *job-shop scheduling problem* (*JSP*) is a mathematical optimization (MO) problem widely used in many industries, like manufacturing (Zhang et al., 2020; Waschneck et al., 2016). For example, a semiconductor manufacturing process can be viewed as a complex JSP problem (Waschneck et al., 2016), where a set of given jobs are assigned to a set of machines under some constraints to achieve some expected goals such as minimizing makespan which is focused on in this paper. While there are many variants of JSP (Abdolrazzagh-Nezhad and Abdullah, 2017), we also consider an extension called *flexible JSP* (*FJSP*) where job operations can be done on designated machines.

A generic approach to solving MO problems is to use mathematical programming, such as mixed integer linear programming (MILP) and constraint satisfaction problem (CSP). Two popular generic MO solvers for solving MO are *OR-Tools* (Perron and Furnon, 2019) and *IBM ILOG CPLEX Optimizer* (abbr. *CPLEX*) (Cplex, 2009). However, both JSP and FJSP, as well as many other MO problems, have been shown to be NP-hard (Garey and Johnson, 1979; Lageweg et al., 1977). That said, it is unrealistic and intractable to find the optimal solution for all cases within reasonable times. These tools can obtain the optimal solutions if sufficient time (or unlimited time) is given; otherwise, return best-effort solutions during the limited time, which usually have gaps to the optimum. When problems are scaled up, the gaps usually grow significantly.

In practice, some heuristics (Gupta and Sivakumar, 2006; Haupt, 1989) or approximate methods (Jansen et al., 2000) were used to cope with the issue of intractability. A simple greedy approach is to

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

use the heuristics following the so-called *priority dispatching rule* (*PDR*) (Haupt, 1989) to construct solutions. These can also be viewed as a kind of *solution construction heuristics* or *construction heuristics*. Some of PDR examples are *First In First Out* (*FIFO*), *Shortest Processing Time* (*SPT*), *Most WorK Remaining* (*MWKR*), and *Most Operation Remaining* (*MOR*). Although these heuristics are usually computationally fast, it is hard to design generally effective rules to minimize the gap to the optimum, and the derived results are usually far from the optimum.

Furthermore, a generic approach to automating the design of heuristics is called *metaheuristics*, such as tabu search (Dell'Amico and Trubian, 1993; Saidi-Mehrabad and Fattahi, 2007) , genetic algorithm (GA) (Pezzella et al., 2008; Ren and Wang, 2012), and PSO algorithms (Lian et al., 2006; Liu et al., 2011). However, metaheuristics still take a high computation time, and it is not ensured to obtain the optimal solution either.

Recently, deep reinforcement learning (DRL) has made several significant successes for some applications, such as AlphaGo (Silver et al., 2016), AlphaStar (Vinyals et al., 2019), AlphaTensor (Fawzi et al., 2022), and thus it also attracted much attention in the MO problems, including chip design (Mirhoseini et al., 2021) and scheduling problems (Zhang et al., 2023). In the past, several researchers used DRL methods as construction heuristics, and their methods did improve scheduling performance, illustrated as follows. Park et al. (2020) proposed a method based on DQN (Mnih et al., 2015) for JSP in semiconductor manufacturing and showed that their DQN model outperformed GA in terms of both scheduling performance (namely gap to the optimum on makespan) and computation time. Lin et al. (2019) and Luo (2020) proposed different DQN models to decide the scheduling action among the heuristic rules and improved the makespan and the tardiness over PDRs, respectively.

A recent DRL-based approach to solving JSP/FJSP problems is to leverage graph neural networks (GNN) to design a size-agnostic representation (Zhang et al., 2020; Park et al., 2021b,a; Song et al., 2023). In this approach, graph representation has better generalization ability in larger instances and provides a holistic view of scheduling states. Zhang et al. (2020) proposed a DRL method with disjunctive graph representation for JSP, called *L2D* (*Learning to Dispatch*), and used GNN to encode the graph for scheduling decision. Besides, Song et al. (2023) extended their methods to FJSP. Park et al. (2021b) used a similar strategy of (Zhang et al., 2020) but with different state features and model structure. Park et al. (2021a) proposed a new approach to solving JSP, called *ScheduleNet*, by using a different graph representation and a DRL model with the graph attention for scheduling decision. Most of the experiments above showed that their models trained from small instances still worked reasonably well for large test instances, and generally better than PDRs. Among these methods, ScheduleNet achieved state-of-the-art (SOTA) performance. There are still other DRL-based approaches to solving JSP/FJSP problems, but not construction heuristics. Zhang et al. (2022) proposes another approach, called Learning to Search (L2S), a kind of search-based heuristics.

In this paper, we propose a new approach to solving JSP/FJSP, a kind of construction heuristics, also based on GNN. In this new approach, we remove irrelevant machines and jobs, such as those finished, such that the states include the remaining machines and jobs only. This approach is named *residual scheduling* in this paper to indicate to work on the remaining graph.

Without irrelevant information, our experiments show that our approach reaches SOTA by outperforming the above mentioned construction methods on some well-known open benchmarks, seven for JSP and two for FJSP, as described in Section 4. We also observe that even though our model is trained for scheduling problems of smaller sizes, our method still performs well for scheduling problems of large sizes. Interestingly in our experiments, our approach even reaches zero gap for 49 among 50 JSP instances whose job numbers are more than 150 on 20 machines.

2 Problem Formulation

2.1 JSP and FJSP

A $n \times m$ JSP instance contains n jobs and m machines. Each job J_j consists of a sequence of k_j operations $\{O_{j,1}, \ldots, O_{j,k_j}\}$, where operation $O_{j,i}$ must be started after $O_{j,i-1}$ is finished. One machine can process at most one operation at a time, and preemption is not allowed upon processing operations. In JSP, one operation $O_{j,i}$ is allowed to be processed on one designated machine, denoted by $M_{j,i}$, with a processing time, denoted by $T^{(op)}_{j,i}$. Table 1 (a) illustrates a 3×3 JSP instance, where the three jobs have 3, 3, 2 operations respectively, each of which is designated to be processed on one of the three machines $\{M_1, M_2, M_3\}$ in the table. A solution of a JSP instance is to dispatch all operations $O_{j,i}$ to the corresponding machine $M_{j,i}$ at time $\tau_{j,i}^{(s)}$, such that the above constraints are satisfied. Two solutions of the above 3x3 JSP instance are given in Figure 1 (a) and (b).

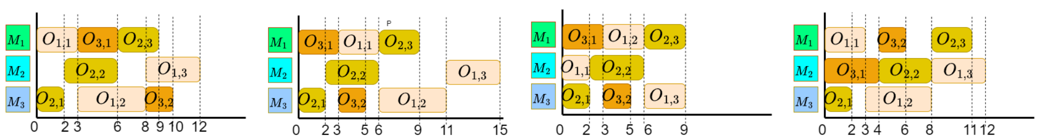
Table 1: JSP and FJSP instances

(a) A 3×3 JSP instance

Job	Operation	M_1	M_2	M_3
Job 1	$O_{1,1}$	3		
	$O_{1,2}$			5
	$O_{1,3}$		4	
Job 2	$O_{2,1}$			2
	$O_{2,2}$		4	
	$O_{2,3}$	3		
Job 3	$O_{3,1}$	3		
	$O_{3,2}$			2

(b) A 3×3 FJSP instance

Job	Operation	M_1	M_2	M_3
Job 1	$O_{1,1}$	3	2	
	$O_{1,2}$	3		5
	$O_{1,3}$		4	3
Job 2	$O_{2,1}$			2
	$O_{2,2}$		4	
	$O_{2,3}$	3		
Job 3	$O_{3,1}$	3	4	
	$O_{3,2}$	2		2



(a) (b) (c) (d)

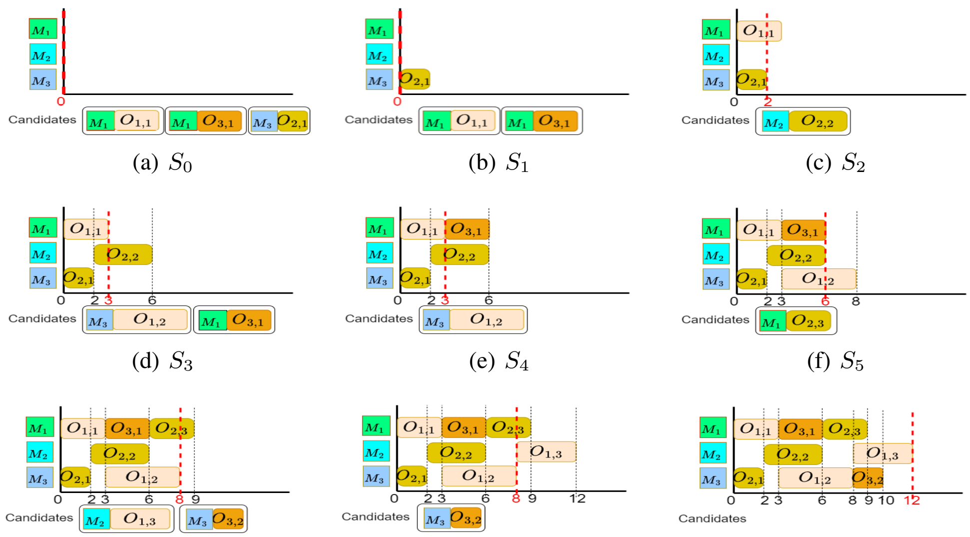
Figure 1: Both (a) and (b) are solutions of the 3x3 JSP instance in Table 1 (a), and the former has the minimal makespan, 12. Both (c) and (d) are solutions of the 3x3 FJSP instance in Table 1 (b), and the former has the minimal makespan, 9.

While there are different expected goals, such as makespan, tardiness, etc., this paper focuses on makespan. Let the first operation start at time $\tau = 0$ in a JSP solution initially. The makespan of the solution is defined to be $T^{(mksp)} = \max(\tau_{j,i}^{(c)})$ for all operations $O_{j,i}$, where $\tau_{j,i}^{(c)} = \tau_{j,i}^{(s)} + T_{j,i}^{(op)}$ denotes the completion time of $O_{j,i}$. The makespans for the two solutions illustrated in Figure 1 (a) and (b) are 12 and 15 respectively. The objective is to derive a solution that minimizes the makespan $T^{(mksp)}$, and the solution of Figure 1 (a) reaches the optimal.

A $n \times m$ FJSP instance is also a $n \times m$ JSP instance with the following difference. In FJSP, all operations $O_{j,i}$ are allowed to be dispatched to multiple designated machines with designated processing times. Table 1 (b) illustrates a 3×3 FJSP instance, where multiple machines can be designated to be processed for one operation. Figure 1 (c) illustrates a solution of an FJSP instance, which takes a shorter time than that in Figure 1 (d).

2.2 Construction Heuristics

An approach to solving these scheduling problems is to construct solutions step by step in a greedy manner, and the heuristics based on this approach is called *construction heuristics* in this paper. In the approach of construction heuristics, a scheduling solution is constructed through a sequence of partial solutions in a chronicle order of dispatching operations step by step, defined as follows. The t-th partial solution S_t associates with a *dispatching time* τ_t and includes a partial set of operations that have been dispatched by τ_t (inclusive) while satisfying the above JSP constraints, and all the remaining operations must be dispatched after τ_t (inclusive). The whole construction starts with S_0 where none of operations have been dispatched and the dispatching time is $\tau_0 = 0$. For each S_t, a set of operations to be chosen for dispatching form a set of pairs of (M, O), called *candidates* C_t, where operations O are allowed to be dispatched on machines M at τ_t. An agent (or a heuristic algorithm) chooses one from candidates C_t for dispatching, and transits the partial solution to the next S_{t+1}. If there exists no operations for dispatching, the whole solution construction process is done and the partial solution is a solution, since no further operations are to be dispatched.



(a) S_0 (b) S_1 (c) S_2

(d) S_3 (e) S_4 (f) S_5

(g) S_6 (h) S_7 (i) S_8

Figure 2: Solution construction, a sequence of partial solutions from S_0 to S_8.

Figure 2 illustrates a solution construction process for the 3x3 JSP instance in Table 1(a), constructed through nine partial solutions step by step. The initial partial solution S_0 starts without any operations dispatched as in Figure 2 (a). The initial candidates C_0 are $\{(M_1, O_{1,1}), (M_3, O_{2,1}), (M_1, O_{3,1})\}$. Following some heuristic, construct a solution from partial solution S_0 to S_9 step by step as in the Figure, where the dashed line in red indicate the time τ_t. The last one S_9, the same as the one in Figure 1 (a), is a solution, since all operations have been dispatched, and the last operation ends at time 12, the makespan of the solution.

For FJSP, the process of solution construction is almost the same except for that one operation have multiple choices from candidates. Besides, an approach based on solution construction can be also viewed as the so-called *Markov decision process* (*MDP*), and the MDP formulation for solution construction is described in more detail in the appendix.

3 Our Approach

In this section, we present a new approach, called *residual scheduling*, to solving scheduling problems. We introduce the residual scheduling in Subsection 3.1, describe the design of the graph representation in Subsection 3.2, propose a model architecture based on graph neural network in Subsection 3.3 and present a method to train this model in Subsection 3.4;

3.1 Residual Scheduling

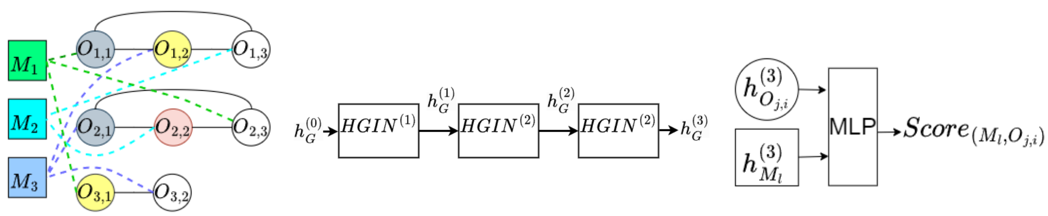
In our approach, the key is to remove irrelevant information, particularly for operations, from states (including partial solutions). An important benefit from this is that we do not need to include all irrelevant information while training to minimize the makespan. Let us illustrate by the state for the partial solution S_3 at time $\tau_3 = 3$ in Figure 2 (d). All processing by τ_3 are irrelevant to the remaining scheduling. Since operations $O_{1,1}$ and $O_{2,1}$ are both finished and irrelevant the rest of scheduling, they can be removed from the state of S_3. In addition, operation $O_{2,2}$ is dispatched at time 2 (before $\tau_3 = 3$) and its processing time is $T^{(op)}_{2,1} = 4$, so the operation is marked as *ongoing*. Thus, the operation can be modified to start at $\tau_3 = 3$ with a processing time $4 - (3 - 2)$. Thus, the modified state for S_3 do not contain both $O_{1,1}$ and $O_{2,1}$, and modify $O_{2,2}$ as above. Let us consider two more examples. For S_4, one more operation $O_{2,2}$ is dispatched and thus marked as ongoing, however, the time τ_4 remains unchanged and no more operations are removed. In this case, the state is almost the same except for including one more ongoing operation $O_{2,2}$. Then, for S_5, two more operations $O_{3,1}$

and $O_{2,2}$ are removed and the ongoing operation $O_{1,2}$ changes its processing time to the remaining time (5-3).

For residual scheduling, we also reset the dispatching time $\tau = 0$ for all states with partial solutions modified as above, so we derive makespans which is also irrelevant to the earlier operations. Given a scheduling policy π, $T_\pi^{(mksp)}(S)$ is defined to be the makespan derived from an episode starting from states S by following π, and $T_\pi^{(mksp)}(S, a)$ the makespan by taking action a on S.

3.2 Residual Graph Representation

In this paper, our model design is based on graph neural network (GNN), and leverage GNN to extract the scheduling decision from the relationship in graph. In this subsection, we present the graph representation. Like many other researchers such as Park et al. (2021a), we formulate a partial solution into a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of edges. A node is either a machine node M or an operation node O. An edge connects two nodes to represent the relationship between two nodes, basically including three kinds of edges, namely operation-to-operation ($O \to O$), machine-to-operation ($M \to O$) and operation-to-machine ($O \to M$). All operations in the same job are fully connected as $O \to O$ edges. If an operation O is able to be performed on a machine M, there exists both $O \to M$ and $M \to O$ directed edges. In (Park et al., 2021a), they also let all machines be fully connected as $M \to M$ edges. However, our experiments in section 4 show that mutual $M \to M$ edges do not help much based on our Residual Scheduling. An illustration for graph representation of S_3 is depicted in Figure 3 (a).



(a) Graph for S_3 (b) Graph embedding (c) Score function

Figure 3: Graph representation and networks.

In the graph representation, all nodes need to include some attributes so that a partial solution S at the dispatching time τ can be supported in the MDP formulation (in the appendix). Note that many of the attributes below are normalized to reduce variance. For nodes corresponding to operations $O_{j,i}$, we have the following attributes:

Status $\phi_{j,i}$: The operation status $\phi_{j,i}$ is *completed* if the operation has been finished by τ, *ongoing* if the operation is ongoing (i.e., has been dispatched to some machine by τ and is still being processed at τ), *ready* if the operation designated to the machine which is idle has not been dispatched yet and its precedent operation has been finished, and *unready* otherwise. For example, in Figure 3 (a), the gray nodes are *completed*, the red *ongoing*, the yellow *ready* and the white *unready*. In our residual scheduling, there exists no completed operations in all partial solutions, since they are removes for irrelevance of the rest of scheduling. The attribute is a one-hot vector to represent the current status of the operation, which is one of *ongoing*, *ready* and *unready*. Illustration for all states S_0 to S_8 are shown in the appendix.

Normalized processing time $\bar{T}^{(op)}_{j,i}$: Let the maximal processing time be $T^{(op)}_{max} = \max_{\forall j,i}(T^{(op)}_{j,i})$. Then, $\bar{T}^{(op)}_{j,i} = T^{(op)}_{j,i} / T^{(op)}_{max}$. In our residual scheduling, the operations that have been finished are removed in partial solutions and therefore their processing time can be ignored; the operations that has not been dispatched yet still keep their processing times the same; the operations that are *ongoing* change their processing times to the remaining times after the dispatching time τ_t. As for FJSP, the operations that has not been dispatched yet may have several processing times on different machines, and thus we can simply choose the average of these processing times.

Normalized job remaining time $\bar{T}^{(job)}_{j,i}$: Let the rest of processing time for job J_j be $T^{(job)}_{j,i} = \sum_{\forall i' \geq i} T^{(op)}_{j,i'}$, and let the processing time for the whole job j be $T^{(job)}_j = \sum_{\forall i'} T^{(op)}_{j,i'}$. In practice, $T^{(job)}_j$ is replaced by the processing time for the original job j. Thus, $\bar{T}^{(job)}_{j,i} = T^{(job)}_{j,i} / T^{(job)}_j$. For FJSP, since operations $O_{j,i}$ can be dispatched to different designated machines M_l, say with the processing time $T^{(op)}_{j,i,l}$, we simply let $T^{(op)}_{j,i}$ be the average of $T^{(op)}_{j,i,l}$ for all M_l.

For machine nodes corresponding to machines M_l, we have the following attributes:

Machine status ϕ_l: The machine status ϕ_l is *processing* if some operation has been dispatched to and is being processed by M_l at τ, and *idle* otherwise (no operation is being processed at τ). The attribute is a one-hot vector to represent the current status, which is one of *processing* and *idle*.

Normalized operation processing time $\bar{T}^{(mac)}_l$: On the machine M_l, the processing time $T^{(mac)}_l$ is $T^{(op)}_{j,i}$ (the same as the normalized processing time for node $O_{j,i}$) if the machine status is *processing*, i.e., some ongoing operation $O_{j,i}$ is being processed but not finished yet, is zero if the machine status is *idle*. Then, this attribute is normalized to $T^{(op)}_{max}$ and thus $\bar{T}^{(mac)}_l = T^{(mac)}_l / T^{(op)}_{max}$.

Now, consider edges in a residual scheduling graph. As described above, there exists three relationship sets for edges, $O \to O$, $O \to M$ and $M \to O$. First, for the same job, say J_j, all of its operation nodes for $O_{j,i}$ are fully connected. Note that for residual scheduling the operations finished by the dispatching time τ are removed and thus have no edges to them. Second, a machine node for M_l is connected to an operation node for $O_{j,i}$, if the operation $O_{j,i}$ is designated to be processed on the machine M_l, which forms two edges $O \to M$ and $M \to O$. Both contains the following attribute.

Normalized operation processing time $\bar{T}^{(edge)}_{j,i,l}$: The attribute is $\bar{T}^{(edge)}_{j,i,l} = T^{(op)}_{j,i} / T^{(op)}_{max}$. Here, $T^{(op)}_{j,i} = T^{(op)}_{j,i,l}$ in the case of FJSP. If operation $O_{j,i}$ is ongoing (or being processed), $T^{(op)}_{j,i}$ is the remaining time as described above.

3.3 Graph Neural Network

In this subsection, we present our model based on graph neural network (GNN). GNN are a family of deep neural networks (Battaglia et al., 2018) that can learn representation of graph-structured data, widely used in many applications (Lv et al., 2021; Zhou et al., 2020). A GNN aggregates information from node itself and its neighboring nodes and then update the data itself, which allows the GNN to capture the complex relationships within the data graph. For GNN, we choose *Graph Isomorphism Network* (*GIN*), which was shown to have strong discriminative power (Xu et al., 2019) and summarily reviewed as follows. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and K GNN layers (K iterations), GIN performs the k-th iterations of updating feature embedding $h^{(k)}$ for each node $v \in \mathcal{V}$:

$$h^{(k)}_v = MLP^{(k)}((1 + \epsilon^{(k)})h^{(k-1)}_v + \sum_{u \in N_b(v)} h^{(k-1)}_u), \tag{1}$$

where $h^{(k)}_v$ is the embedding of node v at the k-th layer, $\epsilon^{(k)}$ is an arbitrary number that can be learned, and $N_b(v)$ is the neighbors of v via edges in $\mathcal{E}$. Note that $h^{(0)}_v$ refers to its raw features for input. $MLP^{(k)}$ is a *Multi-Layer Perceptron* (*MLP*) for the k-th layer with a batch normalization (Ioffe and Szegedy, 2015).

Furthermore, we actually use *heterogeneous GIN*, also called *HGIN*, since there are two types of nodes, machine and operation nodes, and three relations, $O \to O$, $O \to M$ and $M \to O$ in the graph representation. Although we do not have cross machine relations $M \to M$ as described above, updating machine nodes requires to include the update from itself as in (1), that is, there is also one more relation $M \to M$. Thus, HGIN encodes graph information between all relations by using the four MLPs as follows,

$$h^{(k+1)}_v = \sum_{\mathcal{R}} MLP^{(k+1)}_{\mathcal{R}}((1 + \epsilon^{(k+1)}_{\mathcal{R}})h^{(k)}_v + \sum_{u \in N_{\mathcal{R}}(v)} h^{(k)}_u) \tag{2}$$

where $\mathcal{R}$ is one of the above four relations and $MLP^{(k)}_{\mathcal{R}}$ is the MLP for $\mathcal{R}$. For example, for S_0 in Figure 2 (a), the embedding of M_1 in the $(k+1)$-st iteration can be derived as follows.

$$h^{(k+1)}_{M_1} = MLP^{(k+1)}_{MM}((1 + \epsilon^{(k+1)}_{MM})h^{(k)}_{M_1}) + MLP^{(k+1)}_{OM}(h^{(k)}_{O_{1,1}} + h^{(k)}_{O_{1,2}} + h^{(k)}_{O_{1,3}}) \tag{3}$$

Similarly, the embedding of $O_{1,1}$ in the $(k+1)$-st iteration is:

$$h_{O_{1,1}}^{(k+1)} = MLP_{OO}^{(k+1)}((1+\epsilon_{OO}^{(k+1)})h_{O_{1,1}}^{(k)} + h_{O_{1,2}}^{(k)} + h_{O_{1,3}}^{(k)}) + MLP_{MO}^{(k+1)}(h_{M_1}^{(k)}) \tag{4}$$

In our approach, an action includes the two phases, graph embedding phase and action selection phase. Let $h_{\mathcal{G}}^{(k)}$ denote the whole embedding of the graphs $\mathcal{G}$, a summation of the embeddings of all nodes, $h_v^{(k+1)}$. In the graph embedding phase, we use an HGIN to encode node and graph embeddings as described above. An example with three HGIN layers is illustrated in Figure 3 (b).

In the action selection phase, we select an action based on a policy, after node and graph embedding are encoded in the graph embedding phase. The policy is described as follows. First, collect all *ready* operations O to be dispatched to machines M. Then, for all pairs (M, O), feed their node embeddings ($h_M^{(k)}$, $h_O^{(k)}$) into a MLP $Score(M, O)$ to calculate their scores as shown in Figure 3 (c). The probability of selecting (M, O) is calculated based on a softmax function of all scores, which also serves as the model policy π for the current state.

3.4 Policy-Based RL Training

In this paper, we propose to use a policy-based RL training mechanism that follows REINFORCE (Sutton and Barto, 2018) to update our model by policy gradient with a normalized advantage makespan with respect to a baseline policy π_b as follows.

$$A_\pi(S,a) = \frac{T_{\pi_b}^{(mksp)}(S,a) - T_{\pi}^{(mksp)}(S,a)}{T_{\pi_b}^{(mksp)}(S,a)} \tag{5}$$

In this paper, we choose a lightweight PDR, MWKR, as baseline π_b, which performed best for makespan among all PDRs reported from the previous work (Zhang et al., 2020). In fact, our experiment also shows that using MWKR is better than the other PDRs shown in the appendix. The model for policy π is parametrized by θ, which is updated by $\nabla_\theta log\pi_\theta A_{\pi_\theta}(S_t, a_t)$. Our algorithm based on REINFORCE is listed in the appendix.

4 Experiments

4.1 Experimental Settings and Evaluation Benchmarks

In our experiments, the settings of our model are described as follows. All embedding and hidden vectors in our model have a dimension of 256. The model contains three HGIN layers for graph embedding, and an MLP for the score function, as shown in Figure 3 (b) and (c). All MLP networks including those in HGIN and for score contain two hidden layers. The parameters of our model, such as MLP, generally follow the default settings in PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey and Lenssen, 2019). More settings are in the appendix.

Each of our models is trained with one million episodes, each with one scheduling instance. Each instance is generated by following the procedure which is used to generate the TA dataset (Taillard, 1993). Given (N, M), we use the procedure to generate an $n \times m$ JSP instance by conforming to the following distribution, $n \sim \mathcal{U}(3, N)$, $m \sim \mathcal{U}(3, n)$, and operation count $k_j = m$, where $\mathcal{U}(x, y)$ represents a distribution that uniformly samples an integer in a close interval $[x, y]$ at random. The details of designation for machines and processing times refer to (Taillard, 1993) and thus are omitted here. We choose (10,10) for all experiments, since (10,10) generally performs better than the other two as described in the appendix. Following the method described in Subsection 3.4, the model is updated from the above randomly generated instances. For testing our models for JSP and FJSP, seven JSP open benchmarks and two FJSP open benchmarks are used, as listed in the appendix.

The performance for a given policy method π on an instance is measured by the makespan gap G defined as

$$G = \frac{T_{\pi}^{(mksp)} - T_{\pi*}^{(mksp)}}{T_{\pi*}^{(mksp)}} \tag{6}$$

where $T_{\pi*}^{(mksp)}$ is the optimal makespan or the best-effort makespan, from a mathematical optimization tool, OR-Tools, serving as $\pi*$. By the best-effort makespan, we mean the makespan derived with a

Table 2: Average makespan gaps for TA benchmarks.

Size	15×15	20×15	20×20	30×15	30×20	50×15	50×20	100×20	$Avg.$
RS	0.148	**0.165**	0.169	**0.144**	**0.177**	**0.067**	**0.100**	**0.026**	**0.125**
RS+op	**0.143**	0.193	**0.159**	0.192	0.213	0.123	0.126	0.050	0.150
MWKR	0.191	0.233	0.218	0.239	0.251	0.168	0.179	0.083	0.195
MOR	0.205	0.235	0.217	0.228	0.249	0.173	0.176	0.091	0.197
SPT	0.258	0.328	0.277	0.352	0.344	0.241	0.255	0.144	0.275
FIFO	0.239	0.314	0.273	0.311	0.311	0.206	0.239	0.135	0.254
L2D	0.259	0.300	0.316	0.329	0.336	0.223	0.265	0.136	0.270
Park	0.201	0.249	0.292	0.246	0.319	0.159	0.212	0.092	0.221
SchN	0.152	0.194	0.172	0.190	0.237	0.138	0.135	0.066	0.161

sufficiently large time limitation, namely half a day with OR-Tools. For comparison in experiments, we use a server with Intel Xeon E5-2683 CPU and a single NVIDIA GeForce GTX 1080 Ti GPU. Our method uses a CPU thread and a GPU to train and evaluate, while OR-Tools uses eight threads to find the solution.

4.2 Experiments for JSP

For JSP, we first train a model based on residual scheduling, named RS. For ablation testing, we also train a model, named RS+op, by following the same training method but without removing irrelevant operations. When using these models to solve testing instances, action selection is based on the greedy policy that simply chooses the action (M, O) with the highest score deterministically, obtained from the score network as in Figure 3 (c).

For comparison, we consider the three DRL construction heuristics, respectively developed in (Zhang et al., 2020) called L2D, (Park et al., 2021b) by Park et al., and (Park et al., 2021a), called ScheduleNet. We directly use the performance results of these methods for open benchmarks from their articles. For simplicity, they are named L2D, Park and SchN respectively in this paper. We also include some construction heuristics based PDR, such as MWKR, MOR, SPT and FIFO. Besides, to derive the gaps to the optimum in all cases, OR-Tools serve as $\pi*$ as described in (6).

Now, let us analyze the performances of RS as follows. Table 2 shows the average makespan gaps for each collection of JSP TA benchmarks with sizes, 15×15, 20×15, 20×20, 30×15, 30×20, 50×15, 50×20 and 100×20, where the best performances (the smallest gaps) are marked in bold. In general, RS performs the best, and generally outperforms the other methods for all collections by large margins, except for that it has slightly higher gaps than RS+op for the two collections, 15×15 and 20×20. In fact, RS+op also generally outperforms the rest of methods, except for that it is very close to SchN for two collections. For the other six open benchmarks, ABZ, FT, ORB, YN, SWV and LA, the performances are similar and thus presented in the appendix. It is concluded that RS generally performs better than other construction heuristics by large margins.

4.3 Experiments for FJSP

Table 3: Average makespan gaps for FJSP open benchmarks

Method	MK	LA(rdata)	LA(edata)	LA(vdata)
RS	**0.232**	**0.099**	**0.146**	0.031
RS+op	0.254	0.113	0.168	**0.029**
DRL-G	0.254	0.111	0.150	0.040
MWKR	0.282	0.125	0.149	0.051
MOR	0.296	0.147	0.179	0.061
SPT	0.457	0.277	0.262	0.182
FIFO	0.307	0.166	0.220	0.075

For FJSP, we also train a model based on residual scheduling, named RS, and a ablation version, named RS+op, without removing irrelevant operations. We compares ours with one DRL construction heuristics developed by (Song et al., 2023), called DRL-G, and four PDR-based heuristics, MOR, MWKR, SPT and FIFO. We directly use the performance results of these methods for open datasets according to the reports from (Song et al., 2023).

Table 3 shows the average makespan gaps in the four open benchmarks, MK, LA(rdata), LA(edata) and LA(vdata). From the table, RS generally outperforms all the other methods for all benchmarks by large margins, except for that RS+op is slightly better for the benchmark LA(vdata).

5 Discussions

In this paper, we propose a new approach, called residual scheduling, to solving JSP an FJSP problems, and the experiments show that our approach reaches SOTA among DRL-based construction heuristics on the above open JSP and FJSP benchmarks. We further discusses three issues: large instances, computation times and further improvement.

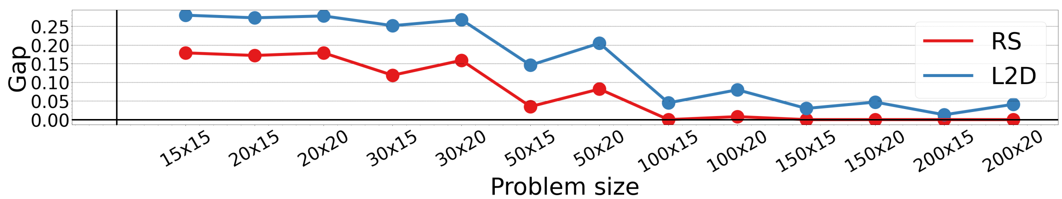


Figure 4: Average makespan gaps of JSP instances with different problem sizes.

First, from the above experiments particularly for TA benchmark for JSP, we observe that the average gaps gets smaller as the number of jobs increases, even if we use the same model trained with $(N, M) = (10, 10)$. In order to investigate size-agnostics, we further generate 13 collections of JSP instances of sizes for testing, from 15 $\times$ 15 to 200 $\times$ 20, and generate 10 instances for each collection by using the procedure above. Figure 4 shows the average gaps for these collections for RS and L2D, and these collections are listed in the order of sizes in the x-axis. Note that we only show the results of L2D in addition to our RS, since L2D is the only open-source among the above DRL heuristics. Interestingly, using RS, the average gaps are nearly zero for the collections with sizes larger than 100 $\times$ 15, namely, 100 $\times$ 15, 100 $\times$ 20, 150 $\times$ 15, 200 $\times$ 15 and 200 $\times$ 20. Among the 50 JSP instances in the five collections, 49 reaches zero gaps. A strong implication is that our RS approach can be scaled up for job sizes and even reach the optimal for sufficient large job count.

Second, the computation times for RS are relatively small and has low variance like most of other construction heuristics. Here, we just use the collection of TA 100x20 for illustration. It takes about 30 seconds on average for both RS and RS+op, about 28 for L2D and about 444 for SchN. In contrast, it takes about 4000 seconds with high variance for OR-tools. The times for other collections are listed in more detail in the appendix.

Table 4: Average makespan gaps for FJSP open benchmark.

Method	MK	LA(rdata)	LA(edata)	LA(vdata)
RS	0.232	0.099	0.146	0.031
RS+100	**0.154**	**0.047**	**0.079**	**0.007**
DRL-G	0.254	0.111	0.150	0.040
DRL+100	0.190	0.058	0.082	0.014

Third, as proposed by Song et al. (2023), construction heuristics can further improve the gap by constructing multiple solutions based on the softmax policy, in addition to the greedy policy. They had a version constructing 100 solutions for FJSP, called DRL+100 in this paper. In this paper, we also implement a RS version for FJSP based on the softmax policy, as described in Subsection 3.3, and then use the version, called RS+100, to constructing 100 solutions. In Table 4, the experimental results show that RS+100 performs the best, much better than RS, DRL-G and DRL+100. An important property for such an improvement is that constructing multiple solutions can be done in parallel. That is, for construction heuristics, the solution quality can be improved by adding more computation powers.

References

Majid Abdolrazzagh-Nezhad and Salwani Abdullah. 2017. Job Shop Scheduling: Classification, Constraints and Objective Functions. *International Journal of Computer and Information Engineering* 11, 4 (2017), 429–434.

Joseph William Adams, Egon Balas, and Daniel J. Zawack. 1988. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management science* 34, 3 (1988), 391–401.

David L. Applegate and William J. Cook. 1991. A Computational Study of the Job-Shop Scheduling Problem. *INFORMS Journal on Computing* 3, 2 (1991), 149–156. https://doi.org/10.1287/ijoc.3.2.149

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew M. Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks. *CoRR* abs/1806.01261 (2018). arXiv:1806.01261 http://arxiv.org/abs/1806.01261

Dennis Behnke and Martin Josef Geiger. 2012. Test instances for the flexible job shop scheduling problem with work centers. *Arbeitspapier/Research Paper/Helmut-Schmidt-Universität, Lehrstuhl für Betriebswirtschaftslehre, insbes. Logistik-Management* (2012).

Paolo Brandimarte. 1993. Routing and scheduling in a flexible job shop by tabu search. *Ann. Oper. Res.* 41, 3 (1993), 157–183. https://doi.org/10.1007/BF02023073

IBM ILOG Cplex. 2009. V12. 1: User's Manual for CPLEX. *International Business Machines Corporation* 46, 53 (2009), 157.

Mauro Dell'Amico and Marco Trubian. 1993. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research* 41, 3 (1993), 231–252. https://doi.org/10.1007/BF02023076

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. 2022. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature* 610, 7930 (2022), 47–53.

Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric, In ICLR Workshop on Representation Learning on Graphs and Manifolds. *CoRR* abs/1903.02428. arXiv:1903.02428 http://arxiv.org/abs/1903.02428

M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, USA.

Amit Kumar Gupta and Appa Iyer Sivakumar. 2006. Job shop scheduling techniques in semiconductor manufacturing. *The International Journal of Advanced Manufacturing Technology* 27, 11 (2006), 1163–1169.

Reinhard Haupt. 1989. A survey of priority rule-based scheduling. *Operations-Research-Spektrum* 11, 1 (1989), 3–16.

Johann Hurink, Bernd Jurisch, and Monika Thole. 1994. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum* 15 (1994), 205–215.

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning (ICML) (JMLR Workshop and Conference Proceedings, Vol. 37)*, Francis R. Bach and David M. Blei (Eds.). JMLR.org, 448–456. http://proceedings.mlr.press/v37/ioffe15.html

Klaus Jansen, Monaldo Mastrolilli, and Roberto Solis-Oba. 2000. Approximation Algorithms for Flexible Job Shop Problems. In *Latin American Symposium on Theoretical Informatics (Lecture Notes in Computer Science, Vol. 1776)*, Gaston H. Gonnet, Daniel Panario, and Alfredo Viola (Eds.). Springer, 68–77. https://doi.org/10.1007/10719839_7

BJ Lageweg, JK Lenstra, and AHG Rinnooy Kan. 1977. Job-shop scheduling by implicit enumeration. *Management Science* 24, 4 (1977), 441–450.

Stephen Lawrence. 1984. Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University* (1984).

Zhigang Lian, Bin Jiao, and Xingsheng Gu. 2006. A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. *Appl. Math. Comput.* 183, 2 (2006), 1008–1017. https://doi.org/10.1016/j.amc.2006.05.168

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 Trans. Ind. Informatics* 15, 7 (2019), 4276–4284. https://doi.org/10.1109/TII.2019.2908210

Min Liu, Zhi-jiang Sun, Junwei Yan, and Jing-song Kang. 2011. An adaptive annealing genetic algorithm for the job-shop planning and scheduling problem. *Expert Systems with Applications* 38, 8 (2011), 9248–9255. https://doi.org/10.1016/j.eswa.2011.01.136

Shu Luo. 2020. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. *Applied Soft Computing* 91 (2020), 106208. https://doi.org/10.1016/j.asoc.2020.106208

Mingqi Lv, Zhaoxiong Hong, Ling Chen, Tieming Chen, Tiantian Zhu, and Shouling Ji. 2021. Temporal Multi-Graph Convolutional Network for Traffic Flow Prediction. *IEEE Transactions on Intelligent Transportation Systems* 22, 6 (2021), 3337–3348. https://doi.org/10.1109/TITS.2020.2983763

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim M. Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, Will Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (2021), 207–212.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. https://doi.org/10.1038/nature14236

J.F. Muth and G.L. Thompson. 1963. *Industrial Scheduling*. Prentice-Hall.

In-Beom Park, Jaeseok Huh, Joongkyun Kim, and Jonghun Park. 2020. A Reinforcement Learning Approach to Robust Scheduling of Semiconductor Manufacturing Facilities. *IEEE Transactions on Automation Science and Engineering* 17, 3 (2020), 1420–1431. https://doi.org/10.1109/TASE.2019.2956762

Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. 2021a. ScheduleNet: Learn to solve multi-agent scheduling problems with reinforcement learning. *CoRR* abs/2106.03051 (2021). arXiv:2106.03051 https://arxiv.org/abs/2106.03051

Junyoung Park, Jaehyeong Chun, Sang Hun Kim, Youngkook Kim, and Jinkyoo Park. 2021b. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research* 59, 11 (2021), 3360–3377. https://doi.org/10.1080/00207543.2020.1870013

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Neural Information Processing Systems (NeurIPS)*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 8024–8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

Laurent Perron and Vincent Furnon. 2019. *OR-Tools*. Google. https://developers.google.com/optimization/

Ferdinando Pezzella, Gianluca Morganti, and Giampiero Ciaschetti. 2008. A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers and Operations Research* 35, 10 (2008), 3202–3212. https://doi.org/10.1016/j.cor.2007.02.014

Qing-dao-er-ji Ren and Yuping Wang. 2012. A new hybrid genetic algorithm for job shop scheduling problem. *Computers and Operations Research* 39, 10 (2012), 2291–2299. https://doi.org/10.1016/j.cor.2011.12.005

Mohammad Saidi-Mehrabad and Parviz Fattahi. 2007. Flexible job shop scheduling with tabu search algorithms. *The International Journal of Advanced Manufacturing Technology* 32, 5 (2007), 563–570.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489. https://doi.org/10.1038/nature16961

Wen Song, Xinyang Chen, Qiqiang Li, and Zhiguang Cao. 2023. Flexible Job-Shop Scheduling via Graph Neural Network and Deep Reinforcement Learning. *IEEE Trans. Ind. Informatics* 19, 2 (2023), 1600–1610. https://doi.org/10.1109/TII.2022.3189725

Robert H. Storer, S. David Wu, and Renzo Vaccari. 1992. New search spaces for sequencing problems with application to job shop scheduling. *Management science* 38, 10 (1992), 1495–1509.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press. http://incompleteideas.net/book/the-book-2nd.html

Éric D. Taillard. 1993. Benchmarks for basic scheduling problems. *european journal of operational research* 64, 2 (1993), 278–285.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354. https://doi.org/10.1038/s41586-019-1724-z

Bernd Waschneck, Thomas Altenmüller, Thomas Bauernhansl, and Andreas Kyek. 2016. Production Scheduling in Complex Job Shops from an Industry 4.0 Perspective: A Review and Challenges in the Semiconductor Industry. In *Proceedings of the 1st International Workshop on Science, Application and Methods in Industry 4.0 (i-KNOW) (CEUR Workshop Proceedings, Vol. 1793)*, Roman Kern, Gerald Reiner, and Olivia Bluder (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-1793/paper3.pdf

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks? (2019). https://openreview.net/forum?id=ryGs6iA5Km

Takeshi Yamada and Ryohei Nakano. 1992. A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems. In *Parallel Problem Solving from Nature 2, (PPSN-II)*, Reinhard Männer and Bernard Manderick (Eds.). Elsevier, 283–292.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. 2020. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. In *Neural Information Processing Systems (NeurIPS)*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/11958dfee29b6709f48a9ba0387a2431-Abstract.html

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. 2022. Learning to Search for Job Shop Scheduling via Deep Reinforcement Learning. *CoRR* abs/2211.10936 (2022). https://doi.org/10.48550/arXiv.2211.10936 arXiv:2211.10936

Cong Zhang, Yaoxin Wu, Yining Ma, Wen Song, Zhang Le, Zhiguang Cao, and Jie Zhang. 2023. A review on learning to solve combinatorial optimisation problems in manufacturing. *IET Collaborative Intelligent Manufacturing* 5, 1 (2023), e12072. https://doi.org/10.1049/cim2.12072 arXiv:https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/cim2.12072

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81. https://doi.org/10.1016/j.aiopen.2021.01.001