IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.
Digital Object Identifier 10.1109/ACCESS.2023.0322000

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

KUO-HAO HO[1], Jui-Yu Cheng[1], Ji-Han Wu[1], Fan Chiang[1], Yen-Chi Chen[2], Yuan-Yu Wu[1], and I-Chen Wu[1 2],
[1]Department of Computer Science, National Yang Ming Chiao Tung University, No. 1001, Daxue Rd., Hsinchu City, Taiwan
[2]Research Center for Information Technology Innovation, Academia Sinica, 128 Academia Road, Taipei, Taiwan
Corresponding author: I-Chen Wu (e-mail: icwu@cs.nctu.edu.tw).

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 in terms of makespan. Interestingly in our experiments, our approach even reaches zero makespan gap for 49 among 60 JSP instances whose job numbers are more than 100 on 15 machines.

INDEX TERMS Deep reinforcement learning (DRL), flexible job-shop scheduling problem (FJSP), graph neural network (GNN), job-shop scheduling problem (JSP).

I. INTRODUCTION

THE *job-shop scheduling problem* (*JSP*) is a combinatorial optimization (CO) problem widely used in many industries, like manufacturing [1, 2]. For example, a semiconductor manufacturing process can be viewed as a complex JSP problem [2], 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 [3], we also consider an extension called *flexible JSP* (*FJSP*) where job operations can be done on designated machines.

A generic approach to solving CO problems is to use mathematical programming, such as mixed integer linear programming (MILP), or constraint programming (CP). Two popular generic CO solvers for solving CO are *OR-Tools* [4] and *IBM ILOG CPLEX Optimizer* (abbr. *CPLEX*) [5]. However, both JSP and FJSP, as well as many other CO problems, have been shown to be NP-hard [6, 7]. 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 [8, 9] or approximate methods [10] were used to cope with the issue of intractability. A simple greedy approach is to use the heuristics following the so-called *priority dispatching rule* (*PDR*) [9] 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 finding near–optimal solutions within limited computation capacity is called *metaheuristics*, such as tabu search, genetic algorithm (GA) [11, 12], and PSO algorithms [13, 14]. 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 [15], AlphaStar [16], AlphaTensor [17], and thus it also attracted much attention in the CO problems, including chip design [18] and scheduling problems [19]. In the past, several researchers used DRL methods as construction heuristics, and their methods did improve scheduling performance, illustrated as follows. Park et al. [20] proposed a method based on DQN [21] 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. [22] and Luo [23] 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 [1, 24, 25, 26]. In this approach, graph representation has better generalization ability in larger instances and provides a holistic view of scheduling states. Zhang et al. [1] 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. [26] extended their methods to FJSP. Park et al. [24] used a similar strategy of [1] but with different state features and model structure. Park et al. [25] 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. [27] 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 IV. 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 makespan gap for 49 among 60 JSP instances whose job numbers are more than 100 on 15 machines.

II. PROBLEM FORMULATION

A. JSP AND FJSP

An $n \times m$ 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 $\tau^{(s)}_{j,i}$, 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

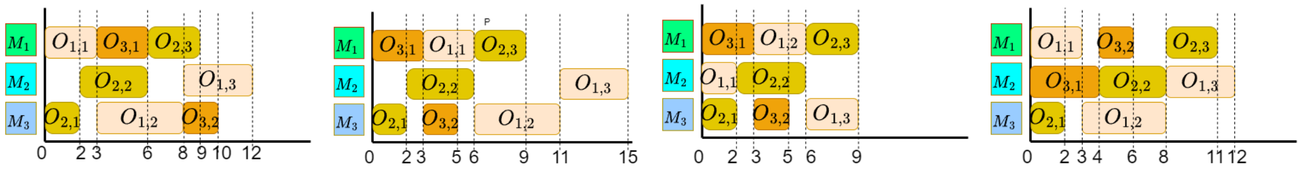
Job	Operation	M_1	M_2	M_3
Job 1	$O_{1,1}$	3		
	$O_{1,2}$			5
	$O_{1,3}$		4	
Job 2	$O_{2,1}$			2
	$O_{2,2}$		4	
	$O_{2,3}$	3		
Job 3	$O_{3,1}$	3		
	$O_{3,2}$			2

(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

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 $T^{(mksp)} = \max(\tau^{(c)}_{j,i})$ for all operations $O_{j,i}$, where $\tau^{(c)}_{j,i} = \tau^{(s)}_{j,i} + T^{(op)}_{j,i}$ 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.

An $n \times m$ FJSP instance is also an $n \times m$ JSP instance with the following difference. In FJSP, all operations $O_{j,i}$ are allowed to be dispatched to multiple designated machines



(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.

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).

B. 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.

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 2, 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 next section.

C. FORMULATION OF MARKOV DECISION PROCESS FOR SOLUTION CONSTRUCTION

In Subsection II-B, an approach based on solution construction can be also viewed as the so-called *Markov decision process* (*MDP*). An MDP is a stochastic decision-making process widely used in reinforcement learning, generally defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$, where $\mathcal{S}$ is the finite set of the states, $\mathcal{A}$ is the set of the available actions, $\mathcal{R}$ is the reward function and $\mathcal{P}$ is the transition probability function. The objective is to find a policy that maximizes the agent's cumulative rewards[1].

Following the above MDP definition, we formulate the process of solution construction for JSP and FJSP problems as follows.

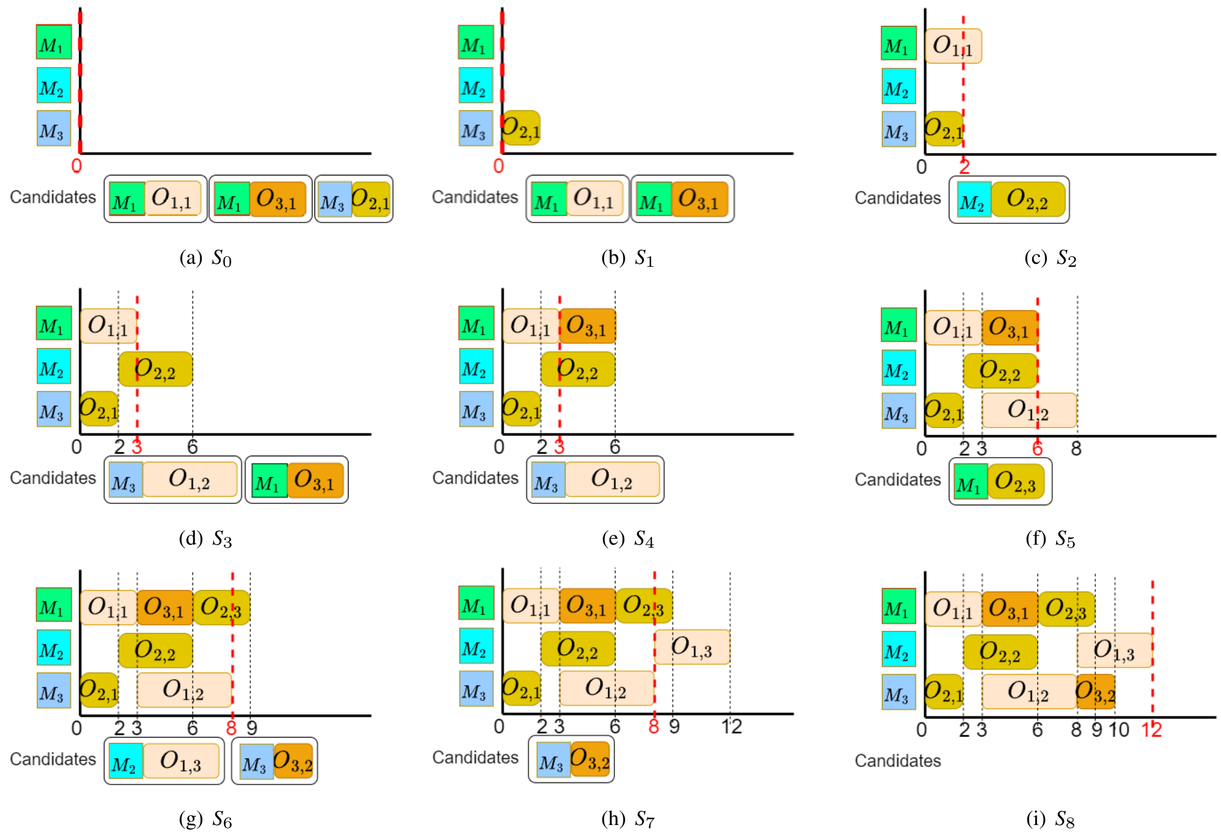
- $\mathcal{S}$ specifies states each representing an instance associated with the information of partial solutions S like those in Figure 2.
- $\mathcal{A}$ specifies actions each of which selects machine-operation pairs (M, O) to dispatch the operation O on the machine M on given states.
- $\mathcal{R}$ specifies a reward to indicate a negative of the additional processing time for the dispatched action.
- $\mathcal{P}$ specifies how states are transited from one partial solution S to the next S' after an action, as described in Subsection II-B accordingly.

An episode starts from the initial partial solution and repeats transitions till the end when the partial solution is a solution. The cumulative reward is the negative of makespan $T^{(mksp)}$, i.e., the total complete time. The objective is to maximize the cumulative reward, i.e., minimize the makespan. In the past, many methods for JSP and FJSP based on reinforcement learning such as [1, 20, 23, 24, 25] followed this MDP formulation.

III. OUR APPROACH

In this section, we present a new approach, called *residual scheduling*, to solving scheduling problems. We introduce the residual scheduling in Subsection III-A, describe the design of the graph representation in Subsection III-B, propose a model architecture based on graph neural network in Sub-

[1]Discount factor γ is not included in the tuple, since it is always one and thus not used in this paper.



(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.

section III-C and present a method to train this model in Subsection III-D;

A. 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, $T^{(mksp)}_{\pi}(S)$ is defined to be the makespan derived from an episode starting from states S by following π, and $T^{(mksp)}_{\pi}(S, a)$ the makespan by taking action a on S.

B. 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. [25], 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 [25], they also let all machines be fully connected as $M \to M$ edges. Figure 3 illustrates the step-by-step graph representations respectively corresponding to all the partial

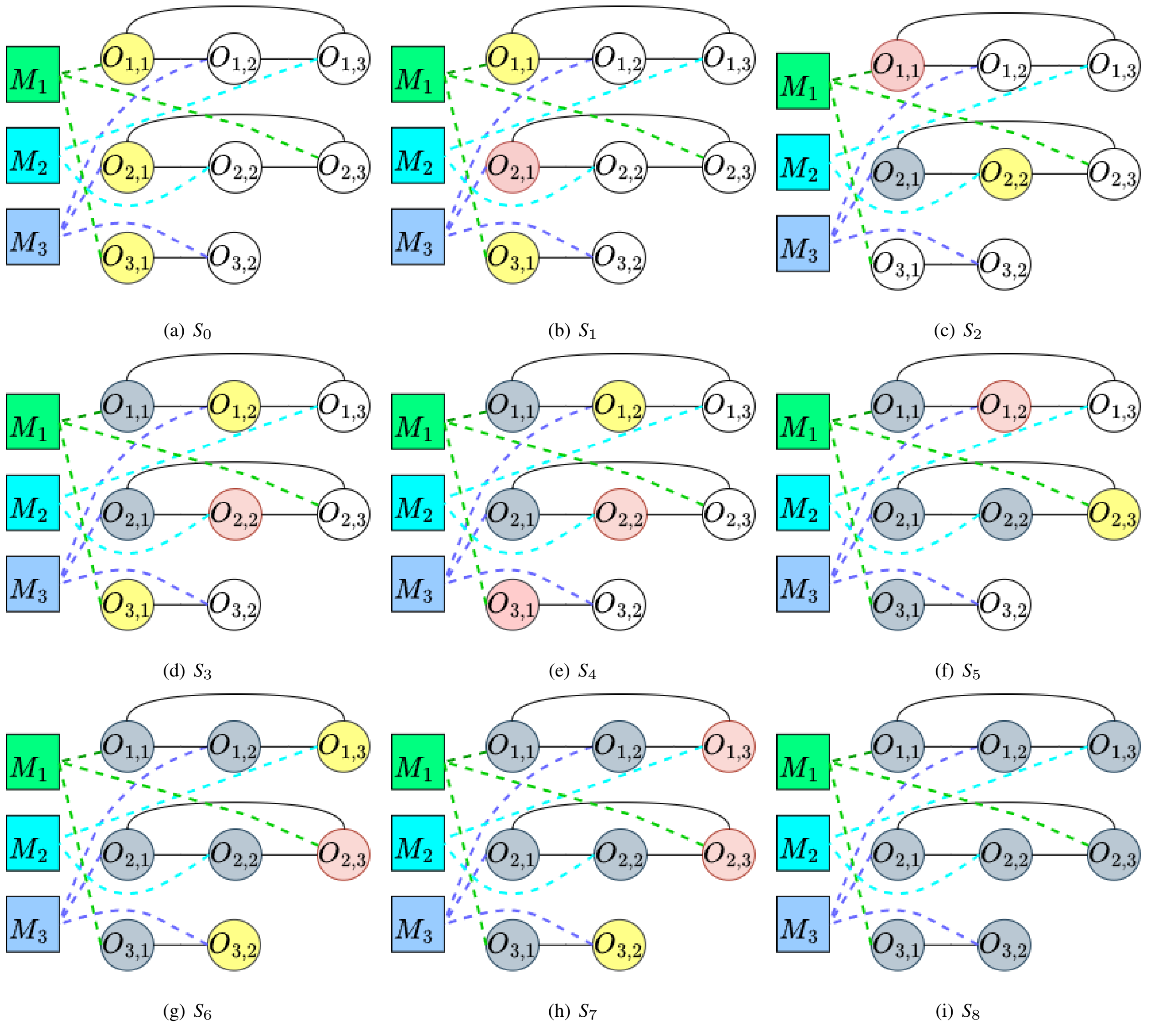


FIGURE 3. (a)-(i) are corresponding to partial solutions in Figure 2 (a)-(i), respectively. The gray nodes are *completed*, the red *ongoing*, the yellow *ready* and the white *unready*.

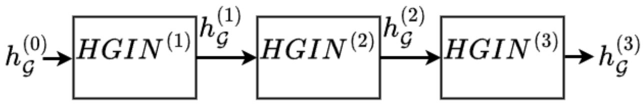
solutions in Figure 2.

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 mentioned in Section II-C. 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:

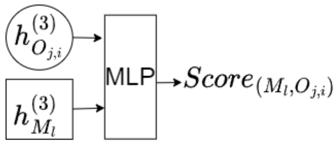
Status $\phi_{j,i}$: The operation status $\phi_{j,i}$ is *completed* if the operation has been finished by τ, *ongoing* if the operation is ongoing (i.e., has been dispatched to some machine by τ and is still being processed at τ), *ready* if the operation designated to the machine which is idle has not been dispatched yet and its precedent operation has been finished, and *unready* otherwise. For example, in Figure 3, the gray nodes are *completed*, the red *ongoing*, the yellow *ready* and the white *unready*. The attribute is a one-hot vector to represent the current status of the operation, which is one of *completed*, *ongoing*, *ready* and *unready*. Illustration for all states S_0 to S_8 are shown in Figure3. In our residual scheduling, all completed operations become irrelevant and thus can be removed. In fact, ongoing operations can also be removed, but machines executing the operations need to record the remaining time to complete these operations. This will be described in more detail later for machine nodes.

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, $\bar{T}^{(op)}_{j,i} = T^{(op)}_{j,i}/T^{(op)}_{max}$ for JSP. In 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



(a) Graph embedding



(b) Score function

FIGURE 4. Graph representation and networks.

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

Normalized waiting time $\bar{W}_{j,i}^{(op)}$: For the *ready* operation, we define the waiting time as the difference between the current time and the time when this node first becomes *ready*, and then the waiting time is normalized by $T_{max}^{(op)}$. For *unready* operations, the (normalized) waiting time is designated as 0.

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 remaining processing time $\bar{T}_l^{(mac)}$: On machine M_l, the remaining processing time $T_l^{(mac)}$ is the difference between the finish time of processing operation on M_l and current time, 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 $T_{max}^{(op)}$ and thus $\bar{T}_l^{(mac)} = T_l^{(mac)}/T_{max}^{(op)}$.

Normalized idle time $\bar{W}_l^{(mac)}$: For an *idle* machine, we define its idle time as the difference between the current time and the time at which the machine become *idle*, and the time is normalized by $T_{max}^{(op)}$. For the *processing* machine, the (normalized) idle time is designated as 0.

Now, consider edges in a residual scheduling graph. As described above, there exists three relationship sets for edges, $O \rightarrow O$, $O \rightarrow M$ and $M \rightarrow O$. 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 $O \rightarrow M$ and $M \rightarrow O$.

C. GRAPH NEURAL NETWORK

In this subsection, we present our model based on graph neural network (GNN). GNN are a family of deep neural networks [28] that can learn representation of graph-structured data, widely used in many applications [29, 30]. 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 as in [31] 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 $v \in \mathcal{V}$:

$$h_v^{(k)} = MLP^{(k)}((1+\epsilon^{(k)})h_v^{(k-1)} + \sum_{u \in N_b(v)} h_u^{(k-1)}), \quad (1)$$

where $h_v^{(k)}$ 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 $h_v^{(0)}$ refers to its raw features for input. $MLP^{(k)}$ is a *Multi-Layer Perceptron* (*MLP*) for the k-th layer with a batch normalization [32].

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

$$h_v^{(k+1)} = \sum_{\mathcal{R}} MLP_{\mathcal{R}}^{(k+1)}((1+\epsilon_{\mathcal{R}}^{(k+1)})h_v^{(k)} + \sum_{u \in N_{\mathcal{R}}(v)} h_u^{(k)}) \quad (2)$$

where $\mathcal{R}$ is one of the above four relations and $MLP_{\mathcal{R}}^{(k)}$ 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.

$$\begin{aligned} h_{M_1}^{(k+1)} &= MLP_{MM}^{(k+1)}((1+\epsilon_{MM}^{(k+1)})h_{M_1}^{(k)}) \\ &+ MLP_{OM}^{(k+1)}(h_{O_{1,1}}^{(k)} + h_{O_{2,3}}^{(k)} + h_{O_{3,1}}^{(k)}) \end{aligned} \quad (3)$$

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

$$\begin{aligned} h_{O_{1,1}}^{(k+1)} &= MLP_{OO}^{(k+1)}((1+\epsilon_{OO}^{(k+1)})h_{O_{1,1}}^{(k)} + h_{O_{1,2}}^{(k)} + h_{O_{1,3}}^{(k)}) \\ &+ MLP_{MO}^{(k+1)}(h_{M_1}^{(k)}) \end{aligned} \quad (4)$$

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 $\mathcal{G}$, including all $h_{O \in \mathcal{G}}^{(k)}$ and $h_{M \in \mathcal{G}}^{(k)}$. In the graph embedding phase, we use an HGIN to encode node embeddings as described above. An example with three HGIN layers is illustrated in Figure 4 (a).

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 ($h^{(k)}_{M_l}$, $h^{(k)}_{O_{j,i}}$, $\bar{T}^{(op)}_{j,i,l}$) into a MLP $Score(M, O)$ to calculate their scores as shown in Figure 4 (b). 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.

D. POLICY-BASED RL TRAINING

In this paper, we propose to use a policy-based RL training mechanism that follows REINFORCE [33] to update our model by policy gradient with a normalized advantage makespan with respect to a baseline policy π_b as follows.

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

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 [1]. 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 $\nabla_\theta log\pi_\theta A_{\pi_\theta}(S_t, a_t)$.

Algorithm 1 (below) shows our REINFORCE training process with the normalized advantage makespan as Equation 5 and an entropy bonus to ensure sufficient exploration like PPO. The model policy π_θ is parameterized by θ initialized at random, and a baseline policy π_b is given, which is a lightweight PDR, MWKR, in this paper.

Algorithm 1 REINFORCE with a normalized advantage for JSP and FJSP

1: **Input:** Initial policy π_θ, learning rate $\alpha \in (0, 1)$, entropy bonus coefficient c, a baseline policy π_b;
2: **for** Episode $e = 0, 1, \cdots$ **do**
3: Use policy π_θ to rollout an episode: $\{S_1, a_1, S_2, a_2, \cdots, S_T\}$
4: **for** $t = 0, 1, 2, \cdots T$ **do**
5: Use the baseline policy π_b to rollout and obtain $T^{(mksp)}_{\pi_b}(S_t, a_t)$
6: Calculate $T^{(mksp)}_{\pi_\theta}(S_t, a_t)$
7: Calculate normalized advantage $A_{\pi_\theta}(S_t, a_t)$ following Equation (5)
8: Update policy $\theta = \theta + \alpha\nabla_\theta(log\pi_\theta A_{\pi_\theta}(S_t, a_t) + cH_{\pi_\theta}(S_t))$
9: **end for**
10: Update learning rate every $P = 1000$ episodes
11: **end for**

IV. EXPERIMENTS

A. 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 4 (a) and (b). 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 [34] and PyTorch Geometric [35]. Table 2 lists the settings of training our model. Particularly, the entropy bonus coefficient c is used for the weight of entropy in line 8 of Algorithm 1, and the initial learning rate α is used in lines 1 and 8 of Algorithm 1.

Each of our models is trained with 300,000 episodes, each with one scheduling instance. Each instance is generated by following the procedure which is used to generate the TA dataset [36]. Given (N, M), we use the procedure to generate an $n \times m$ JSP instance by conforming to the following distribution, $n \sim \mathcal{U}(3, N)$, $m \sim \mathcal{U}(3, M)$, 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 reason for choosing three as a lower bound for the number of jobs or machines is simply because it is very likely to be a trivial case of no more than two jobs and two machines. The details of designation for machines and processing times refer to [36] 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 III-D, 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 Table 3. For JSP, TA benchmark has most instances, including 8 categories, 15×15, 20×15, 20×20, 30×15, 30×20, 50×15, 50×20 and 100×20. For each category, there are 10 instances generated by their procedure. For other JSP benchmarks, they cover different problem sizes and data distributions as in the table.

For FJSP, two popular open benchmarks are MK, also known as Brandimarte instances, and LA, also known as Hurink instances (to distinguish LA benchmark for JSP). Hurink instances are categories into three types, edata, rdata and vdata, according to different distribution for assignable machines, denoted by LA(edata), LA(rdata) and LA(vdata) respectively, as described in [37].

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

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

where $T^{(mksp)}_{\pi*}$ 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 sufficiently large time limitation,

TABLE 2. Model hyperparameters.

Item	Value	Description
HGIN layers	3	Layer number of GNN.
FC layers of MLP	2	Layer number of MLP.
Hidden dimension	256	Hidden vector dimension for GNN and policy network.
Episodes	300,000	Training episodes
Entropy bonus coefficient (c)	10^{-2}	The weight of entropy bonus in the loss function.
Initial learning rate (α)	10^{-4}	Initialized with 10^{-4} and then updated with a decay of 0.99 rate for every 1000 episodes
Optimizer	Adam	Adam with $\beta_1 = 0.9$ and $\beta_2 = 0.999$

TABLE 3. Open benchmarks.

Type	Benchmark	Instances	Min size	Max size
JSP	TA [36]	80	15×15	100×20
	ABZ [38]	5	10×10	20×15
	FT [39]	3	6×6	20×5
	ORB [40]	10	10×10	10×10
	YN [41]	4	20×20	20×20
	SWV [42]	20	20×10	50×10
	LA [43]	40	10×5	30×10
FJSP	MK [44]	10	10×6	20×15
	LA [37]	120	10×5	30×10

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.

B. 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 4 (b).

For comparison, we consider the three DRL construction heuristics, respectively developed in [1] called L2D, [24] by Park et al., and [25], 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 4 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. RS+op also generally outperforms the rest of methods, except for that it is very close to SchN for the smallest collection, 15 × 15.

Table 5 shows the average makespan gaps for other six JSP open benchmarks, where the best performances (the smallest gaps) are marked in bold. Similar to the results in TA benchmark, in general, RS still performs the best, and generally outperforms the other methods except for that RS has slightly higher gaps than RS+op for the FT benchmark. In fact, RS+op also generally outperforms the rest of methods. It is concluded that without irrelevant information the method RS generally performs better than other construction heuristics by large margins.

C. EXPERIMENTS FOR FJSP

For FJSP, we also train a model based on residual scheduling, named RS, and an ablation version, named RS+op, without removing irrelevant operations. We compares ours with one DRL construction heuristics developed by [26], 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 [26].

Table 6 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, especially for the benchmark MK.

Song et al. [26] proposed a version constructing 100 solutions for FJSP, called DRL+100 in this paper, which can further improve the gap by constructing multiple solutions based on the softmax policy. We also implement a RS version for FJSP based on the softmax policy, as described in Subsection III-C, and then use the version, called RS+100, to constructing 100 solutions. From Table 7, RS+100 clearly outperforms DRL+100, DRL-G and RS by significant margins.

D. COMPUTATION TIME

Table 8 and Table 9 show the computation time taken by methods for TA JSP benchmark and FJSP benchmarks. The four heuristics, MWKR, MOR, SPT and FIFO, take almost the same time to obtain the scheduling solutions, so they are merged into the PDRs category. For TA benchmark, the computation times for L2D and SchN were reported by their works [1] and [25], respectively, which also reported to use a machine with AMD Ryzen 3600 CPU and a single Nvidia GeForce 2070S GPU. To our knowledge, there were no time records available for the work [24], marked as "-". We observe that the time for RS is much less than SchN, and comparable to other DRL-based construction heuristics.

TABLE 4. 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	**0.137**	**0.180**	**0.165**	**0.173**	**0.181**	**0.084**	**0.114**	**0.040**	**0.134**
RS+op	0.156	0.194	0.170	0.188	0.210	0.116	0.128	0.047	0.151
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

TABLE 5. Average makespan gaps for other benchmarks.

Dataset	ABZ	FT	ORB	YN	SWV	LA
RS	**0.111**	0.102	**0.164**	**0.159**	**0.159**	**0.081**
RS+op	0.140	**0.085**	**0.164**	0.173	0.164	0.083
MWKR	0.166	0.196	0.251	0.197	0.284	0.126
MOR	0.180	0.232	0.290	0.228	0.358	0.138
SPT	0.251	0.280	0.262	0.306	0.230	0.199
FIFO	0.247	0.288	0.297	0.258	0.373	0.188
Park	0.214	0.222	0.218	0.247	0.228	0.141
SchN	0.147	0.184	0.199	0.184	0.288	0.099

TABLE 6. Average makespan gaps for FJSP open benchmarks

Method	MK	LA(rdata)	LA(edata)	LA(vdata)
RS	**0.098**	**0.096**	**0.132**	**0.038**
RS+op	0.134	0.120	0.145	0.048
DRL-G	0.278	0.111	0.155	0.042
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

TABLE 7. Average makespan gaps for FJSP open benchmark.

Method	MK	LA(rdata)	LA(edata)	LA(vdata)
RS	0.098	0.096	0.132	0.038
RS+100	**0.071**	**0.047**	**0.069**	**0.008**
DRL-G	0.278	0.111	0.155	0.042
DRL+100	0.190	0.058	0.082	0.014

TABLE 8. Average computation times for TA JSP instances.

Size	15×15	20×15	20×20	30×15	30×20	50×15	50×20	100×20	*Avg.*
RS	0.50s	0.89s	0.97s	1.93s	2.27s	4.75s	5.93s	19.76s	4.63s
PDRs	0.05s	0.08s	0.12s	0.16s	0.24s	0.41s	0.60s	2.20s	0.48s
L2D	0.40s	0.60s	1.10s	1.30s	1.50s	2.20s	3.6s	28.20s	4.86s
Park	-	-	-	-	-	-	-	-	-
SchN	3.50s	6.60s	11.00s	17.10s	28.30s	52.50s	96.00s	444.0s	82.38s

TABLE 9. Average computation times for FJSP open benchmarks.

Method	MK	LA(rdata)	LA(edata)	LA(vdata)
RS	0.85s	0.63s	0.80s	0.98s
PDRs	0.41s	0.47s	0.47s	0.46s
DRL-G	0.90s	0.97s	1.04s	0.96s

To make a fairness comparison with OR-tools, we reran our program with CPU. Table 10 shows that our version with CPU (with 20 threads) takes roughly twice of computation time for our version with GPU. Let OR-tools use 20 threads run within the same times (T) as above for each group of dataset (in the row of CPU time). In this way, the obtained makespans

TABLE 10. Average computation times for TA JSP instances for CPU and GPU.

Size	15×15	20×15	20×20	30×15	30×20	50×15	50×20	100×20
CPU times	1.30s	1.55s	1.45s	3.67s	4.65s	10.04s	13.10s	51.47s
GPU times	0.50s	0.89s	0.97s	1.93s	2.27s	4.75s	5.93s	19.76s

TABLE 11. Makespan gap for TA JSP instances with different computation time T.

Size	15×15	20×15	20×20	30×15	30×20	50×15	50×20	100×20
T (by RS)	0.137	0.180	0.165	0.173	0.181	0.084	0.114	0.040
T (by OR-Tools)	0.081	0.129	0.136	0.199	0.222	0.138	0.172	0.109
$2T$ (by OR-Tools)	0.040	0.085	0.087	0.147	0.169	0.101	0.145	0.092
$4T$ (by OR-Tools)	0.011	0.049	0.055	0.099	0.124	0.076	0.104	0.078

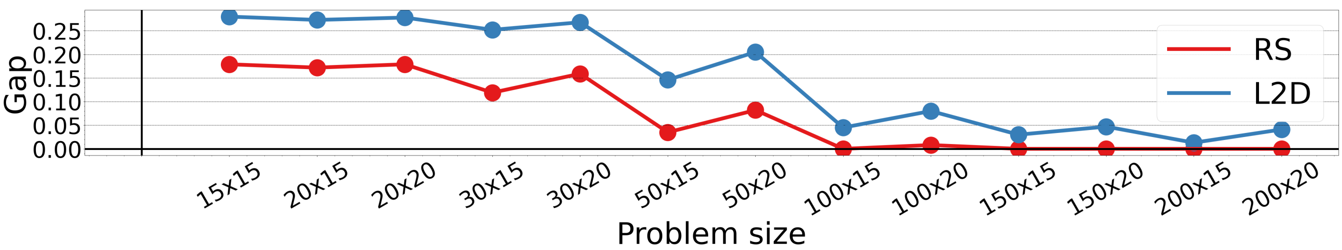


FIGURE 5. Average makespan gaps of JSP instances with different problem sizes.

have the gaps as shown in Table 11. From the table, ours outperformed OR-tools' except for some small cases when limiting the running time for OR-tools, even for $2T$ and $4T$. For some large instances like 100×20, 50×20, 50×15, the table shows that longer times do not help improve OR-tools' much.

V. 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.

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 × 15 to 200 × 20, and generate 10 instances for each collection by using the procedure above. Given half-a-day computation, OR tools returned optimal solutions for most instances, interestingly especially for the problem sizes larger than 100 × 15. Figure 5 shows the average gaps for these collections for RS and L2D (where RS is clearly better than L2D in general), 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 × 15, namely, 100 × 15, 100 × 20, 150 × 15, 150 × 20, 200 × 15 and 200 × 20. Among the 60 JSP instances in the six collections, 49 reaches zero makespan gaps and the details are listed in the appendix. 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 20 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 Table 8 and Table 9.

Third, as proposed by Song et al. [26], 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 III-C, and then use the version, called RS+100, to constructing 100 solutions. In Table 7 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.

VI. ACKNOWLEDGEMENTS

This research is partially supported by the National Science and Technology Council (NSTC) of Taiwan under Grant Numbers 112-2622-8-A49-021, 111-2221-E-A49-101-MY2 and 110-2221-E-A49-067-MY3. The authors would also like to thank anonymous reviewers for their valuable comments.

REFERENCES

[1] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and C. Xu, "Learning to dispatch for job shop scheduling via deep reinforcement learning," in *Neural Information Processing Systems (NeurIPS)*, 2020.

[2] B. Waschneck, T. Altenmüller, T. Bauernhansl, and A. Kyek, "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)*, vol. 1793, 2016.

[3] M. Abdolrazzagh-Nezhad and S. Abdullah, "Job shop scheduling: Classification, constraints and objective functions," *International Journal of Computer and Information Engineering*, vol. 11, no. 4, pp. 429–434, 2017.

[4] L. Perron and V. Furnon. (2019) Or-tools. Google. [Online]. Available: https://developers.google.com/optimization/

[5] I. I. Cplex, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

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

[7] B. Lageweg, J. Lenstra, and A. Rinnooy Kan, "Job-shop scheduling by implicit enumeration," *Management Science*, vol. 24, no. 4, pp. 441–450, 1977.

[8] A. K. Gupta and A. I. Sivakumar, "Job shop scheduling techniques in semiconductor manufacturing," *The International Journal of Advanced Manufacturing Technology*, vol. 27, no. 11, pp. 1163–1169, 2006.

[9] R. Haupt, "A survey of priority rule-based scheduling," *Operations-Research-Spektrum*, vol. 11, no. 1, pp. 3–16, 1989.

[10] K. Jansen, M. Mastrolilli, and R. Solis-Oba, "Approximation algorithms for flexible job shop problems," in *Latin American Symposium on Theoretical Informatics*, vol. 1776, 2000, pp. 68–77.

[11] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers and Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008.

[12] Q. Ren and Y. Wang, "A new hybrid genetic algorithm for job shop scheduling problem," *Computers and Operations Research*, vol. 39, no. 10, pp. 2291–2299, 2012.

[13] Z. Lian, B. Jiao, and X. Gu, "A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan," *Applied Mathematics and Computation*, vol. 183, no. 2, pp. 1008–1017, 2006.

[14] M. Liu, Z. Sun, J. Yan, and J. Kang, "An adaptive annealing genetic algorithm for the job-shop planning and scheduling problem," *Expert Systems with Applications*, vol. 38, no. 8, pp. 9248–9255, 2011.

[15] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[16] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, Ç. Gülçehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in starcraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[17] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz *et al.*, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.

[18] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. M. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. V. Le, J. Laudon, R. Ho, R. Carpenter, and J. Dean, "A graph placement methodology for fast chip design." *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[19] C. Zhang, Y. Wu, Y. Ma, W. Song, Z. Le, Z. Cao, and J. Zhang, "A review on learning to solve combinatorial optimisation problems in manufacturing," *IET Collaborative Intelligent Manufacturing*, vol. 5, no. 1, p. e12072, 2023.

[20] I. Park, J. Huh, J. Kim, and J. Park, "A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 3, pp. 1420–1431, 2020.

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

[22] C. Lin, D. Deng, Y. Chih, and H. Chiu, "Smart manufacturing scheduling with edge computing using multiclass deep Q network," *IEEE Trans. Ind. Informatics*, vol. 15, no. 7, pp. 4276–4284, 2019.

[23] S. Luo, "Dynamic scheduling for flexible job shop with

new job insertions by deep reinforcement learning," *Applied Soft Computing*, vol. 91, p. 106208, 2020.
[24] J. Park, J. Chun, S. H. Kim, Y. Kim, and J. Park, "Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning," *International Journal of Production Research*, vol. 59, no. 11, pp. 3360–3377, 2021.
[25] J. Park, S. Bakhtiyar, and J. Park, "Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning," *CoRR*, vol. abs/2106.03051, 2021.
[26] W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible job-shop scheduling via graph neural network and deep reinforcement learning," *IEEE Trans. Ind. Informatics*, vol. 19, no. 2, pp. 1600–1610, 2023.
[27] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and C. Xu, "Learning to search for job shop scheduling via deep reinforcement learning," *CoRR*, vol. abs/2211.10936, 2022.
[28] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, 2018.
[29] M. Lv, Z. Hong, L. Chen, T. Chen, T. Zhu, and S. Ji, "Temporal multi-graph convolutional network for traffic flow prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 6, pp. 3337–3348, 2021.
[30] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.
[31] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *International Conference on Learning Representations (ICLR)*, 2019.
[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning (ICML)*, ser. JMLR Workshop and Conference Proceedings, vol. 37, 2015, pp. 448–456.
[33] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, 2nd ed. MIT press, 2018.
[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Neural Information Processing Systems (NeurIPS)*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035.
[35] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, vol. abs/1903.02428, 2019.
[36] É. D. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.
[37] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines," *Operations-Research-Spektrum*, vol. 15, pp. 205–215, 1994.
[38] J. W. Adams, E. Balas, and D. J. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management science*, vol. 34, no. 3, pp. 391–401, 1988.
[39] J. Muth and G. Thompson, *Industrial Scheduling*, ser. Prentice-Hall International series in management. Prentice-Hall, 1963.
[40] D. L. Applegate and W. J. Cook, "A computational study of the job-shop scheduling problem," *INFORMS Journal on Computing*, vol. 3, no. 2, pp. 149–156, 1991.
[41] T. Yamada and R. Nakano, "A genetic algorithm applicable to large-scale job-shop problems," in *Parallel Problem Solving from Nature 2, (PPSN-II)*, R. Männer and B. Manderick, Eds. Elsevier, 1992, pp. 283–292.
[42] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Management science*, vol. 38, no. 10, pp. 1495–1509, 1992.
[43] S. Lawrence, "Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement)," *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.
[44] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Ann. Oper. Res.*, vol. 41, no. 3, pp. 157–183, 1993.
[45] D. Behnke and M. J. Geiger, "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.

KUO-HAO HO (M'02) received the B.S. and M.S. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan in 2000 and 2002, respectively. He is currently pursuing the Ph.D. degree in computer science at National Yang Ming Chiao Tung University, Hsinchu, Taiwan.

JUI-YU CHENG received the B.S. degrees in applied mathematics from National Yang Ming Chiao Tung University, Hsinchu, Taiwan in 2022. He is currently pursuing the M.S degree in Institute of Artificial Intelligence Innovation at National Yang Ming Chiao Tung University, Hsinchu, Taiwan.

JI-HAN WU (M'22) received the B.S. in science and technology from National Yunlin University, Yunlin, Taiwan in 2019, and the M.S. degrees in multimedia engineering from National Yang Ming Chiao Tung University, Hsinchu, Taiwan in 2022. He was a research assistant in Computer Games and Intelligence Lab in 2023.

FAN CHIANG (M'22) received the B.S. and M.S. degrees in computer science from National Yang Ming Chiao Tung University, Hsinchu, Taiwan in 2020 and 2022, respectively. He is currently an Research Assistan in Computer Games and Intelligence Lab.

YEN-CHI CHEN received the B.S. degree in mathematics and M.S. degree in computer science from National Taiwan Normal University, Taipei, Taiwan in 2017 and 2019, respectively. He is currently an Research Assistan in Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan.

YUAN-YU WU (M'20) received the B.S. and M.S. degrees in computer science from National Yang Ming Chiao Tung University, Hsinchu, Taiwan in 2018 and 2020, respectively.

I-CHEN WU (M'05-SM'15) is with the Department of Computer Science, at National Yang Ming Chiao Tung University. He received his B.S. in Electronic Engineering from National Taiwan University (NTU), M.S. in Computer Science from NTU, and Ph.D. in Computer Science from Carnegie-Mellon University, in 1982, 1984 and 1993, respectively. He serves in the editorial board of the IEEE Transactions on Games and ICGA Journal, and served as chairs and committee in over 50 academic conferences and organizations, including the conference chair of IEEE CIG conference 2015.

His research interests include computer games and deep reinforcement learning. Dr. Wu introduced the new game, Connect6, a kind of six-in-a-row game. He led a team developing various game playing programs and reinforcement learning systems, winning over 60 gold medals in international tournaments, including Computer Olympiad and AWS DeepRacer World Championships (2020 and 2022). He published over 170 papers, including top-tier conferences such as AAAI, IJCAI, NeurIPS, ICLR, UAI and ICRA.

APPENDIX A CODE, DATASETS AND MODEL WEIGHTS

Our code, datasets and pre-trained weights are available in `ResidualScheduling` project[2]. The code includes the following: setting up the virtual environment, the installation of required packages, and the code with the commands in the `README` file. All hyperparameters are also listed in `README` file.

We collect the following dataset for evaluation. For JSP dataset, we collect 13 collections generated for the experiments shown in Figure 5, and other seven JSP benchmarks, TA, ABZ, FT, ORB, YN, SWV and LA. For FJSP dataset, we collect two FJSP benchmarks, MK and LA.

There are four weights according to the final models for RS and RS+op with JSP and FJSP cases, denoted as `RS_JSP`, `RS+op_JSP`, `RS_FJSP`, and `RS+op_FJSP`.

APPENDIX B MODEL ABLATION

In practice, one model is trained for each of (N, M), (6,6), (10,10) and (15,15), and each of baseline policies, MWKR, MOR, SPT and FIFO. Each of models is trained with 1000 iterations, for each of which model parameters are updated with 1000 episodes (300 thousands of episodes in total). It takes about one day to train with 200,000 episodes. For each episode, one scheduling instance is generated for training by the procedure subject to (N, M) as in Subsection IV-A.

For JSP, we also generate a validation dataset of the three collections, 20 × 20, 50 × 20 and 100 × 20, each with 10 instances. Among the 1000 sets of model parameters, the one performing best for the validation dataset is chosen for testing in the rest of experiments. Table 12 shows the average makespans of the four models trained with different baselines respectively and with (10,10) on the validation dataset. The one with MWKR clearly outperforms other baselines on the validation dataset. Thus, MWKR is chosen as the baseline in this paper.

Similarly, Table 13 shows the average makespans of three models trained with different training sizes (6,6), (10,10) and (15,15) respectively and with MWKR on the validation dataset. The one with (10,10) performs best, except for that the one with (15,15) performs slightly better for the collection of 100 × 20. Thus, this paper still chooses (10,10) since it performs well in general and has less training costs when compared to (15,15).

For FJSP, we simply use the validation set used by [26]. Table 14 shows the average makespans of three models trained with different training sizes (6,6), (10,10) and (15,15) respectively and with MWKR on the validation dataset. The one with (10,10) outperforms others for all the collections in [26]. Hence, (10,10) is also chosen in FJSP experiments in this paper.

APPENDIX C EXPERIMENT RESULTS IN MORE DETAIL

This section presents experiment results in more detail by listing makespans for all instances. First, Table 15 shows the average JSP makespan gaps in Figure 5, and Tables 16 and 17 show in more detail the makespans of all individual JSP instances obtained by RS, L2D and OR-tools. For OR-tools, the time limitation is set to half a day, i.e., it is set to the best-effort if OR-tools exceed the time limitation, where the cases are marked with †. For all 60 JSP instances larger than and equal to 100 × 15, RS reaches all optimum (marked in bold) except for four 100 × 20 instances and one 150 × 20 instance in Table 17.

Second, Tables 18 and 19 show the makespans of FJSP MK instances and LA instances, respectively. OPT indicates the best-known results from [45]. From the tables, RS+100 obtains makespans the same as (marked in bold) or close to those in OPT for many instances.

Third, Table 20 and Table 21 show the makespans of all TA instances for Table 4. Park and SchN represent the work [24] and ScheduleNet; and OPT is the best-effort results for OR-tools in 6000 seconds. To indicate the best performance of the construction heuristics, we mark in bold the makespan

[2]https://github.com/Raydiation/ResidualScheduling_IEEE_access

TABLE 12. Average makespans of models with different baseline policies π_b on JSP validation dataset.

π_b	20×20	50×20	100×20
MWKR	1804.2	3197.1	5698.8
MOR	1810.9	3214.1	5733.3
SPT	1806.4	3191.9	5744.6
FIFO	1806.5	3203.5	5733.3

TABLE 13. Average makespans of models with different (N, M) on JSP validation dataset.

Method	20×20	50×20	100×20
RS (6,6)	1824.0	3183.2	5766.7
RS (10,10)	1804.2	3197.1	5698.8
RS (15,15)	1807.4	3214.7	5734.4

TABLE 14. Average makespans of models with different (N, M) on FJSP validation dataset.

Method	10 × 5	15 × 10	20 × 5	20 × 10
RS (6,6)	106.60	161.64	205.74	206.21
RS (10,10)	105.79	160.99	202.84	203.65
RS (15,15)	105.93	161.97	204.76	203.55

TABLE 15. Average makespan gaps in Figure 5.

Method	OR-Tools	RS	L2D
15 × 15	0	0.165	0.280
20 × 15	0†	0.171	0.273
20 × 20	0†	0.166	0.278
30 × 15	0†	0.121	0.252
30 × 20	0†	0.154	0.268
50 × 15	0	0.032	0.146
50 × 20	0†	0.091	0.205
100 × 15	0	0.000	0.045
100 × 20	0	0.008	0.080
150 × 15	0	0.0003	0.030
150 × 20	0	0.003	0.047
200 × 15	0	0.000	0.013
200 × 20	0	0.001	0.041

†means the best-effort solution from OR-Tools with a half day computation (43,200 seconds).

that is closest to that of OPT. We observe that RS performs best for most instances from Ta30 to Ta80, and outperforms others for nearly all the instances with sizes larger than and equal to 30×15.

TABLE 16. Part 1 of makespans of all JSP instances of Table 15.

Instance	$n \times m$	L2D	RS	OR-Tools	$n \times m$	L2D	RS	OR-Tools
01	15×15	1528	1383	1181	-	-	-	-
02	15×15	1422	1353	1172	-	-	-	-
03	15×15	1500	1457	1243	-	-	-	-
04	15×15	1651	1405	1230	-	-	-	-
05	15×15	1663	1583	1302	-	-	-	-
06	15×15	1479	1407	1237	-	-	-	-
07	15×15	1590	1349	1139	-	-	-	-
08	15×15	1432	1365	1148	-	-	-	-
09	15×15	1591	1301	1170	-	-	-	-
10	15×15	1492	1371	1168	-	-	-	-
01	20×15	1754	1537	1290	20×20	1864	1711	1483
02	20×15	1789	1556	1386	20×20	2116	2008	1669
03	20×15	1868	1708	1424	20×20	2225	2009	1697
04	20×15	1896	1702	1437	20×20	1989	1750	1510
05	20×15	1767	1638	1383	20×20	1749	1803	1539
06	20×15	1564	1513	1264	20×20	1950	1790	1524
07	20×15	1603	1493	1315	20×20	2065	1841	1614
08	20×15	1652	1561	1316	20×20	2103	1752	1512
09	20×15	1611	1563	1319	20×20	1914	1647	1482
10	20×15	1660	1508	1335	20×20	2001	1936	1597
01	30×15	2154	1965	1691	30×20	2390	2171	1939
02	30×15	2473	2057	1822	30×20	2501	2418	2056
03	30×15	2430	2209	2078	30×20	2269	2119	1835
04	30×15	2099	1822	1600	30×20	2470	2177	1936
05	30×15	2178	1981	1828	30×20	2420	2173	1958
06	30×15	2158	1987	1778	30×20	2414	2193	1824
07	30×15	2193	1960	1863	30×20	2639	2197	1921
08	30×15	2219	2072	1801	30×20	2233	2142	1804
09	30×15	2354	2041	1750	30×20	2569	2263	1973
10	30×15	1938	1789	1550	30×20	2491	2358	2001
01	50×15	3238	2852	2781	50×20	3732	3272	3047
02	50×15	3571	3174	2954	50×20	3477	3124	2760
03	50×15	3114	**2787**	2787	50×20	3381	3116	2969
04	50×15	3111	2912	2845	50×20	3446	3070	2847
05	50×15	3074	**2924**	2924	50×20	3267	3036	2775
06	50×15	3113	2780	2588	50×20	3565	3262	2962
07	50×15	3230	3045	3036	50×20	3678	3243	2920
08	50×15	3448	2978	2969	50×20	3723	3260	2919
09	50×15	3437	3102	2810	50×20	3538	3177	2873
10	50×15	3397	2926	2873	50×20	3456	**3411**	3241
01	100×15	5945	**5493**	5493	100×20	6215	5544	5505
02	100×15	5683	**5517**	5517	100×20	5978	**5663**	5663
03	100×15	5457	**5212**	5212	100×20	6136	5513	5443
04	100×15	6088	**5802**	5802	100×20	6193	5680	5511
05	100×15	5701	**5568**	5568	100×20	6147	**6066**	6066
06	100×15	6094	**5821**	5821	100×20	6176	**5699**	5699
07	100×15	6019	**5756**	5756	100×20	6054	5617	5574
08	100×15	5898	**5876**	5876	100×20	6040	5631	5505
09	100×15	5545	**5367**	5367	100×20	5981	**5886**	5886
10	100×15	5784	**5303**	5303	100×20	6089	**5689**	5689

TABLE 17. Part 2 of makespans of all JSP instances of Table 15.

Instance	$n \times m$	L2D	RS	OR-Tools	$n \times m$	L2D	RS	OR-Tools
01	150 × 15	8902	**8470**	8470	150 × 20	8630	**8146**	8146
02	150 × 15	8077	**7871**	7871	150 × 20	8258	**8139**	8139
03	150 × 15	8281	**7964**	7964	150 × 20	8693	8194	8178
04	150 × 15	8268	**8215**	8215	150 × 20	8717	**8219**	8219
05	150 × 15	8423	8000	7974	150 × 20	8720	**8663**	8663
06	150 × 15	8544	**8506**	8506	150 × 20	8577	8288	8086
07	150 × 15	8703	**8227**	8227	150 × 20	8622	8013	7962
08	150 × 15	8265	**7940**	7940	150 × 20	8395	**8101**	8101
09	150 × 15	8522	**8472**	8472	150 × 20	8308	**8000**	8000
10	150 × 15	8099	**7937**	7937	150 × 20	8576	**8215**	8215
01	200 × 15	11018	**10696**	10696	200 × 20	10687	**10545**	10545
02	200 × 15	10770	**10576**	10576	200 × 20	11453	**10464**	10464
03	200 × 15	11030	**10952**	10952	200 × 20	12069	11101	11040
04	200 × 15	10881	**10881**	10881	200 × 20	11044	**10644**	10644
05	200 × 15	10706	**10701**	10701	200 × 20	11070	**10793**	10793
06	200 × 15	10540	**10510**	10510	200 × 20	11029	10518	10427
07	200 × 15	10810	**10345**	10345	200 × 20	10972	**10699**	10699
08	200 × 15	10355	**10355**	10355	200 × 20	11033	**10710**	10710
09	200 × 15	10983	**10668**	10668	200 × 20	10988	**10811**	10811
10	200 × 15	11036	**10969**	10969	200 × 20	10787	**10666**	10666

TABLE 18. Makespans of all MK instances (FJSP).

Instance	$n \times m$	RS	RS+100	OPT
mk01	10 × 6	45	42	39
mk02	10 × 6	29	28	26
mk03	15 × 8	**204**	**204**	204
mk04	15 × 8	67	67	60
mk05	15 × 4	185	177	172
mk06	10 × 15	74	71	58
mk07	20 × 5	150	149	139
mk08	20 × 10	**523**	**523**	523
mk09	20 × 10	314	311	307
mk10	20 × 15	225	218	197

TABLE 19. Makespans of all LA instances (FJSP).

		edata			rdata			vdata		
Instance	$n \times m$	RS	RS+100	OPT	RS	RS+100	OPT	RS	RS+100	OPT
la01	10×5	658	621	609	618	588	571	620	576	570
la02	10×5	789	710	655	598	550	530	564	535	529
la03	10×5	611	573	550	516	487	478	492	482	477
la04	10×5	679	604	568	546	517	502	529	506	502
la05	10×5	530	**503**	503	502	466	457	481	467	457
la06	15×5	875	**833**	833	810	804	799	855	804	799
la07	15×5	868	820	762	782	761	750	757	754	749
la08	15×5	969	860	845	789	773	765	778	771	765
la09	15×5	912	900	878	882	864	853	869	855	853
la10	15×5	929	**866**	866	840	816	804	840	811	804
la11	20×5	1157	1111	1103	1094	1078	1071	1086	1073	1071
la12	20×5	979	979	960	987	941	936	974	941	936
la13	20×5	1061	1061	1053	1067	1048	1038	1074	1043	1038
la14	20×5	1195	1147	1123	1085	1082	1070	1098	1074	1070
la15	20×5	1397	1291	1111	1151	1098	1090	1112	1093	1089
la16	10×10	1017	951	892	784	761	717	740	**717**	717
la17	10×10	821	780	707	732	693	646	708	**646**	646
la18	10×10	919	899	842	772	716	666	**663**	**663**	663
la19	10×10	941	909	796	828	772	700	671	**617**	617
la20	10×10	1007	946	857	889	815	756	**756**	**756**	756
la21	15×10	1197	1118	1017	987	909	835	847	816	806
la22	15×10	1017	949	882	894	843	760	765	753	739
la23	15×10	1122	1052	950	983	894	842	858	829	815
la24	15×10	1025	996	909	896	877	808	828	790	777
la25	15×10	1117	1016	941	935	875	791	827	776	756
la26	20×10	1394	1240	1125	1169	1108	1061	1080	1062	1054
la27	20×10	1344	1302	1186	1209	1152	1091	1110	1100	1085
la28	20×10	1331	1307	1149	1196	1122	1080	1095	1084	1070
la29	20×10	1289	1254	1118	1106	1043	998	1031	1003	994
la30	20×10	1371	1329	1204	1191	1143	1078	1119	1081	1069
la31	30×10	1698	1673	1539	1576	1551	1521	1561	1534	1520
la32	30×10	1872	1799	1698	1742	1692	1659	1702	1671	1658
la33	30×10	1664	1622	1547	1529	1516	1499	1527	1512	1497
la34	30×10	1778	1693	1604	1601	1557	1536	1548	1548	1535
la35	30×10	1935	1773	1736	1653	1584	1550	1603	1562	1549
la36	15×15	1347	1263	1162	1192	1136	1030	982	**948**	948
la37	15×15	1588	1492	1397	1208	1169	1077	1044	**986**	986
la38	15×15	1444	1283	1144	1104	1074	962	943	**943**	943
la39	15×15	1414	1286	1184	1222	1116	1024	964	935	922
la40	15×15	1316	1248	1150	1082	1057	970	966	**955**	955

TABLE 20. Part 1 of makespans of all TA instances (JSP).

Instance	$n \times m$	SPT	FIFO	MOR	Park	L2D	SchN	RS	OPT
Ta01	15×15	1462	1486	1438	**1389**	1443	1452	1417	1231
Ta02	15×15	1446	1486	1452	1519	1544	1411	**1378**	1244
Ta03	15×15	1495	1461	1418	1457	1440	1396	**1393**	1218
Ta04	15×15	1708	1575	1457	1465	1637	1348	**1337**	1175
Ta05	15×15	1618	1457	1448	**1352**	1619	1382	1367	1224
Ta06	15×15	1522	1528	1486	1481	1601	1413	**1385**	1238
Ta07	15×15	1434	1497	1456	1554	1568	1380	**1372**	1227
Ta08	15×15	1457	1496	1482	1488	1468	**1374**	1421	1217
Ta09	15×15	1622	1642	1594	1556	1627	1523	**1519**	1274
Ta10	15×15	1697	1600	1582	1501	1527	1493	**1397**	1241
Ta11	20×15	1865	1701	1665	1626	1794	**1612**	1618	1357
Ta12	20×15	1667	1670	1739	1668	1805	**1600**	1646	1367
Ta13	20×15	1802	1862	1642	1715	1932	1625	**1583**	1342
Ta14	20×15	1635	1812	1662	1642	1664	1590	**1570**	1345
Ta15	20×15	1835	1788	1682	1672	1730	1676	**1588**	1339
Ta16	20×15	1965	1825	1638	1700	1710	**1550**	1561	1360
Ta17	20×15	2059	1899	1856	**1678**	1897	1753	1698	1462
Ta18	20×15	1808	1833	1710	1684	1794	1668	**1646**	1396
Ta19	20×15	1789	1716	1651	1900	1682	1622	**1586**	1332
Ta20	20×15	1710	1827	1622	1752	1739	**1604**	1607	1348
Ta21	20×20	2175	2089	1964	2199	2252	1921	**1904**	1642
Ta22	20×20	1965	2146	1905	2049	2102	**1844**	1849	1600
Ta23	20×20	1933	2010	1922	2006	2085	**1879**	1882	1557
Ta24	20×20	2230	1989	1943	2020	2200	1922	**1859**	1644
Ta25	20×20	1950	2160	1957	1981	2201	**1897**	1938	1595
Ta26	20×20	2188	2182	1964	2057	2176	1887	**1860**	1643
Ta27	20×20	2096	2091	2160	2187	2132	2009	**1975**	1680
Ta28	20×20	1968	1980	1952	2054	2146	**1813**	1819	1603
Ta29	20×20	2166	2011	1899	2210	1952	**1875**	1911	1625
Ta30	20×20	1999	1941	2017	2140	2035	1913	**1855**	1584
Ta31	30×15	2335	2277	2143	2251	2565	**2055**	2067	1764
Ta32	30×15	2432	2279	2188	2378	2388	2268	**2076**	1784
Ta33	30×15	2453	2481	2308	2316	2324	2281	**2265**	1791
Ta34	30×15	2434	2546	2193	2319	2332	**2061**	2119	1829
Ta35	30×15	2497	2478	2255	2333	2505	2218	**2131**	2007
Ta36	30×15	2445	2433	2307	2210	2497	2154	**2083**	1819
Ta37	30×15	2664	2382	2190	2201	2325	**2112**	2177	1771
Ta38	30×15	2155	2277	2179	2151	2302	1970	**1941**	1673
Ta39	30×15	2477	2255	2167	2138	2410	2146	**2124**	1795
Ta40	30×15	2301	2069	2028	2007	2140	2030	**1991**	1669
Ta41	30×20	2499	2543	2538	2654	2667	2572	**2417**	2005
Ta42	30×20	2710	2669	2440	2579	2664	2397	**2270**	1937
Ta43	30×20	2434	2506	2432	2737	2431	2310	**2196**	1846
Ta44	30×20	2906	2540	2426	2772	2714	2456	**2344**	1979
Ta45	30×20	2640	2565	2487	2435	2637	2445	**2254**	2000
Ta46	30×20	2667	2582	2490	2681	2776	2541	**2399**	2004
Ta47	30×20	2620	2508	2286	2428	2476	2280	**2232**	1889
Ta48	30×20	2620	2541	2371	2440	2490	2358	**2227**	1941
Ta49	30×20	2666	2550	2397	2446	2556	2301	**2287**	1961
Ta50	30×20	2429	2531	2469	2530	2628	2453	**2387**	1923

TABLE 21. Part 2 of makespans of all TA instances (JSP).

Instance	$n \times m$	SPT	FIFO	MOR	Park	L2D	SchN	RS	OPT
Ta51	50×15	3856	3590	3567	3145	3599	3382	**3085**	2760
Ta52	50×15	3266	3365	3303	3157	3341	3231	**3093**	2756
Ta53	50×15	3507	3169	3115	3103	3186	3083	**2884**	2717
Ta54	50×15	3142	3218	3265	3278	3266	3068	**2924**	2839
Ta55	50×15	3225	3291	3279	3142	3232	3078	**3023**	2679
Ta56	50×15	3530	3329	3100	3258	3378	3065	**2909**	2781
Ta57	50×15	3725	3654	3335	3230	3471	3266	**3145**	2943
Ta58	50×15	3365	3362	3420	3469	3732	3321	**3057**	2885
Ta59	50×15	3294	3357	3117	3108	3381	3044	**2923**	2655
Ta60	50×15	3500	3129	3044	3256	3352	3036	**3026**	2723
Ta61	50×20	3606	3690	3376	3425	3654	**3202**	3259	2868
Ta62	50×20	3639	3657	3417	3626	3722	3339	**3156**	2869
Ta63	50×20	3521	3367	3276	3110	3536	3118	**3033**	2755
Ta64	50×20	3447	3179	3057	3329	3631	2989	**2962**	2702
Ta65	50×20	3332	3273	3249	3339	3359	**3168**	3207	2725
Ta66	50×20	3677	3610	3335	3340	3555	3199	**3109**	2845
Ta67	50×20	3487	3612	3392	3371	3567	3236	**3180**	2825
Ta68	50×20	3336	3471	3251	3265	3680	3072	**2962**	2784
Ta69	50×20	3862	3607	3526	3798	3592	3535	**3344**	3071
Ta70	50×20	3801	3784	3590	3919	3643	**3436**	3479	2995
Ta71	100×20	6232	6270	5938	5962	6452	5879	**5642**	5464
Ta72	100×20	5973	5671	5639	5522	5695	5456	**5371**	5181
Ta73	100×20	6482	6357	6128	6335	6462	6052	**5851**	5568
Ta74	100×20	6062	6003	5642	5827	5885	**5513**	5604	5339
Ta75	100×20	6217	6420	6212	6042	6355	5992	**5837**	5392
Ta76	100×20	6370	6183	5936	5707	6135	5773	**5636**	5342
Ta77	100×20	6045	5952	5829	5737	6056	5637	**5501**	5436
Ta78	100×20	6143	6328	5886	5979	6101	5833	**5558**	5394
Ta79	100×20	6018	6003	5652	5799	5943	5556	**5385**	5358
Ta80	100×20	5848	5763	5707	5718	5892	5545	**5460**	5183