We greatly appreciate the valuable comments from all reviewers.

General response: comparison with sophisticated heuristics. Both reviewer #1 and #4 point out the necessity of comparing with more sophisticated non-PDR heuristics (often complicated metaheuristics). On one hand, it is not quite fair to compare with them since they often consume much longer time to iteratively explore the solution space, while PDR heuristic is a one-time quick construction process. On the other hand, though not explicitly mentioned, over 90% of the best results for the Taillard's and DMU benchmarks (generated using the same procedure we used to sample training instances) are obtained by well-tuned metaheuristics as listed in the links we provide in the paper. Hence in Table 3 and 4, we are actually comparing with the best results from a collection of well-tuned, non-PDR heuristics.

Reviewer#1

Regarding strong heuristic. Please kindly refer to the general response above.

Regarding feasibility constraints. In job shop scheduling, due dates are often considered as soft constraints, i.e. violations incur tardiness with costs. The optimization objective in such case is often in the form of minimizing weighted tardiness. Nevertheless, for the cases where due dates are hard constraints, our method can also be applied by introducing extra (large) penalty into the reward such that the agent could learn to avoid violating the hard constraints.

Reviewer#2

Regarding better quality but longer time. Our baselines are traditional hand-crafted PDRs, which are derived from human experience and often give solutions relatively far from optimality. The better solution quality of our method comes from the fact that our PDRs are automatically learned and optimized by the reinforcement learning system, which exceeds the limitation of human knowledge. On the other hand, traditional PDRs are computationally cheap while our PDRs need additional overhead in the inference of GNN, hence have longer (but acceptable) computational time.

Regarding reproducibility. As we mentioned in the paper, all source code and trained models will be released.

Reviewer#3

Regarding generalization. The main reason for not testing on larger instances is lacking benchmarking solutions since OR-Tools already times out on 98% of 100x20 ones. Here we quickly run our 30x20 policy and baselines on 10 200x50 instances generated using Taillard's method. The average makespans are: SPT (15879.8), MWKR (15391.7), FDD/MWKR (14095.0), MOPNR (13387.3), and Ours (13070.8). This shows that our policy still performs better than traditional PDRs on instances an order of magnitude larger than those used in training.

Regarding training cost. The training time of all sizes is: 0.95h (6x6), 2.6h (10x10), 6.2h (15x15), 8.7h (20x15), 11.6h (20x20), 13.5h (30x15), 20.3h (30x20). Since training is offline, such cost is reasonable considering the good online quality and speed. Training always converges, and we will present training costs and curves in the final version.

Regarding OR-Tools time. The average time of OR-Tools is 0.012s, 0.027s, and 211.814s for size 6x6, 10x10, and 15x15, respectively. For sizes larger than 20x20, OR-Tools are timed out on most instances, i.e. exceeds 3600s.

Regarding the example in Figure 2. We thank the reviewer for pointing out this error. In the case of Figure 2, O_{22} will always be a successor of O_{11}. We will correct this mistake in the final version.

For other detailed comments, we greatly appreciate the careful review and will incorporate them in the final version.

Reviewer#4

Regarding GIN. Our main focus is to design a size-agnostic end-to-end model via GNN that can automatically learn strong PDRs. However, we are not restricted to particular GNN models. We choose GIN mainly because it is one of the strongest GNN architectures with proved discriminative power. We will try other GNNs in the future.

Regarding well-tuned heuristic. Please kindly refer to the general response above.

Regarding uncertainty and other shop scheduling. While we focus on deterministic setting here, our method could potentially handle uncertainty since it is naturally captured by MDP. The feature and reward used here are not necessarily deterministic, and could be replaced by expectations/observations of random variables. For other shop scheduling problems, as long as they share the disjunctive graph representation, they can be handled by our method which operates directly on the disjunctive graphs. As mentioned in the paper, both directions will be investigated in the future.

Regarding contemporary publications. We thank the reviewer for pointing out these papers. They are indeed relevant, though they are not solving JSSP and do not exploit the disjunctive graph representation scheme. We will cite them properly in the final version.

Regarding hyper-parameters and GNN design. The architecture of our model (e.g. K, and hidden dimension) and all the hyper-parameters are empirically tuned on small instances and fixed for both training and evaluation.

Regarding other objectives. Currently we focus on makespan which is the most common objective for JSSP. Nevertheless, our method can be applied to other objectives such as tardiness by modifying the reward function. We will investigate this in the future.

Regarding the example in Figure 2. We thank the reviewer for pointing out this error. In the case of Figure 2, O_{22} will always be a successor of O_{11}. We will correct this mistake in the final version.

Regarding greedy policy during testing. For testing, greedy policy is much faster for inference and still gives good results (see [15]). Sampling multiple solutions will certainly yield better results, and we will try it in the future.