DSGNN: A Dual-Backbone Graph Surrogate for Makespan Prediction in Scheduling Problem

Anonymous author
Anonymous affiliation

Abstract

The classical Job Shop Scheduling Problem (JSP) is a canonical benchmark in constraint programming, yet obtaining fast instance-level signals that can support solver selection, bound initialization, or early search guidance remains difficult. We study supervised makespan prediction directly from structured JSP encodings. Each instance is represented as an edge-typed scheduling graph whose nodes are operations and whose edges encode conjunctive precedence and disjunctive machine-conflict relations. On this representation, we propose DSGNN, a dual-backbone graph surrogate that combines two complementary message-passing mechanisms: GraphSAGE for stable neighbourhood aggregation and graph attention for adaptive relation weighting. Each backbone produces an independent graph-level estimate, and the final prediction is their convex combination. We train on approximately 65,000 solver outcomes from IGJSP and JSPLIB, using high-quality labels obtained from exact and CP/MIP solvers under size-dependent time limits. Evaluation on synthetic and benchmark instances shows strong tolerance-based accuracy across mixed-source regimes, while the dual-backbone model consistently improves robustness over its single-backbone components. These results suggest that lightweight graph surrogates can provide useful instance-level signals for hybrid CP workflows in job shop scheduling.

2012 ACM Subject Classification Computing methodologies → Machine learning approaches; Theory of computation → Scheduling algorithms; Mathematics of computing → Discrete optimization

Keywords and phrases job shop scheduling, graph neural networks, makespan prediction, surrogate models, constraint programming, machine learning

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The Job shop scheduling problem (JSP) is a classical NP-hard combinatorial optimisation problem in which each job is an ordered sequence of operations that must be processed on specific machines, subject to precedence and capacity constraints, typically with the objective of minimizing the makespan. Exact approaches based on constraint programming (CP), mixed-integer programming (MIP), and branch-and-bound provide strong guarantees; however, their computational cost grows rapidly with instance size, limiting their use for online decision-making in large or highly dynamic environments. Heuristics and metaheuristics scale better, but they often require expert tuning and can vary substantially across instance distributions.

Artificial intelligence methods have recently become an effective complement to these classical techniques. Deep reinforcement learning (DRL) has been used to learn dispatching policies from graph-structured shop states, including dynamic settings with disruptions and uncertainty [2]. Beyond sequential decision-making, learning has also been applied to discover or select dispatching rules [8] and to configure metaheuristics for job-shop variants. In parallel, there is growing interest in integrating learning into exact optimization, for example by using learned models to steer search, provide surrogate guidance, or improve bounds within CP/MIP pipelines [1, 7]. These hybrid perspectives motivate supervised predictors that can provide fast, instance-level signals without requiring rollout-based reward optimization.


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Learning-based methods depend critically on diverse and well-characterised benchmarks. In this paper, we adopt a supervised surrogate perspective: we encode JSP instances as disjunctive/conjunctive scheduling graphs and train graph neural networks (GNNs) to directly predict makespan from high-quality labels (solver-certified optimal when available, otherwise best-found feasible makespans under time limits). Specifically, labels are obtained by running several CP/MIP-based solvers under size-dependent time limits and retaining the best feasible makespan (optimal when certified).

While learning-based JSP research largely focuses on DRL dispatching, supervised instance-level makespan regression remains less studied at scale, under protocols that separate optimal from feasible-only supervision, and for test robustness across synthetic generators and heterogeneous public benchmarks such as JSPLIB.

This paper makes four concrete contributions:

- **DSGNN surrogate model:** we introduce DSGNN, a dual-backbone graph neural surrogate for instance-level makespan prediction in JSP that combines SAGE and GAT predictors via a fixed uniform convex combination, designed to improve robustness over single-backbone variants.
- **Problem framing:** we cast instance-level makespan estimation for JSP as supervised regression on disjunctive/conjunctive scheduling graphs, targeting fast signals for downstream tasks such as difficulty assessment, solver selection, and pruning/bounding inside CP/MIP pipelines.
- **Reproducible data and labeling protocol:** we generate a large synthetic corpus with IGJSP and complement it with JSPLIB; makespan labels are obtained by running multiple solvers under size-dependent time limits and retaining the best feasible solution (optimal when certified).
- **Controlled evaluation and analysis:** we evaluate DSGNN against solver-derived makespans, reporting tolerance-based accuracy separately on solver-certified optimal instances under time limits; we additionally analyze its single-backbone components under identical training conditions.

2 Problem setting

We consider the classical Job Shop Scheduling Problem (JSP) with a single time-related objective: makespan minimization. The mathematical formulation follows [4]; here, we summarize the main elements, specialized to the single-objective setting without speed scaling.

2.1 Sets and parameters.

We are given a set of jobs $J = \{1, \dots, n\}$ and a set of machines $M = \{1, \dots, m\}$. Each job $j \in J$ consists of an ordered sequence of operations $T_j = \{t_1^j, \dots, t_m^j\}, |T_j| = |M|$; each operation $t \in T_j$ must be processed on a predetermined machine $m(t) \in M$. For each operation (j, t) we are given:

- P_t^j: processing time of operation t of job j.

In the MILP used to generate makespan labels, we minimize the maximum completion time/makespan. This MILP is used solely as a labeling oracle to obtain reference makespans at scale; we do not claim modeling or solution improvements over standard JSP formulations.

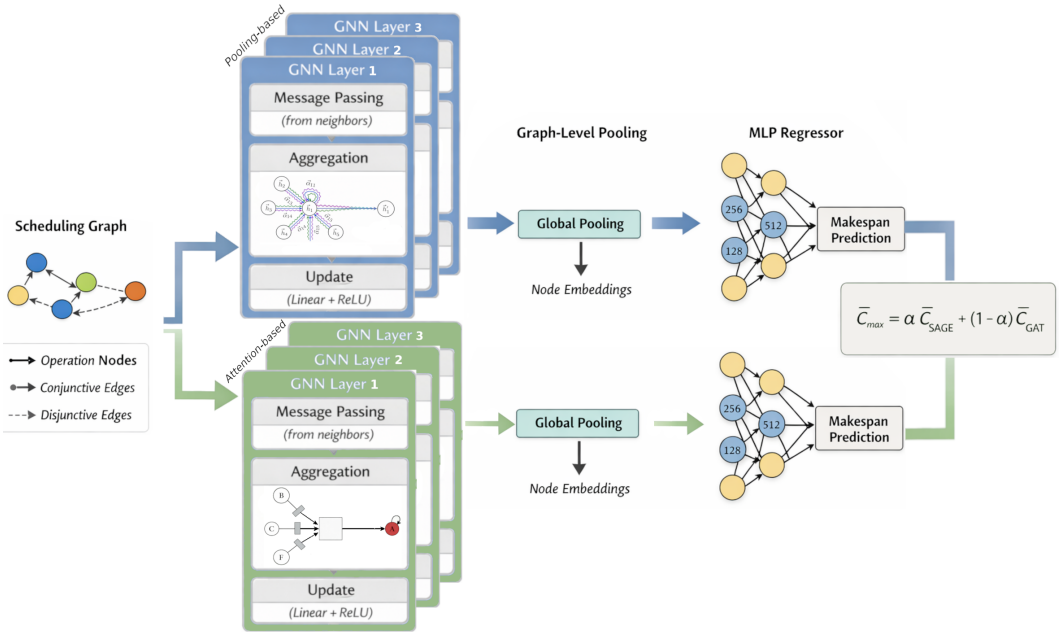

Figure 1 DSGNN: A Dual-backbone architecture for makespan prediction. GNNs with pooling-based / attention-based layers are combined via a convex combination controlled by $\alpha \in [0, 1]$ to predict makespan.

3 A Dual-Surrogate GNNs Approach (DSGNN)

GNNs learn node embeddings through an iterative process of neighborhood aggregation. We follow the standard message-passing formulation and notation in [5, 6]: at each layer, each node aggregates information from its neighbors through a learnable mechanism and combines it with its current representation via trainable transformations and nonlinearities. In scheduling graphs, this enables information flow across both precedence and machine-conflict relations, producing graph representations suitable for instance-level prediction, here focused on makespan estimation.

Figure 1 summarizes the common pipeline adopted in this work. A JSP instance is encoded as a multi-relational (edge-typed) graph with a single node type (operations) and two edge types: conjunctive precedence constraints and disjunctive machine-conflict relations. We apply three stacked message-passing layers, each consisting of neighborhood aggregation followed by batch normalization, ReLU activation, and dropout. Node embeddings are then pooled into a graph-level representation (global mean pooling in our implementation) and passed to a regression MLP to predict the makespan. All evaluated backbones share this overall structure and differ only in the aggregation operator used within the message-passing layers (sample and aggregate or graph attention network).

3.1 Sample and Aggregate (SAGE)

SAGE is an inductive GNN that aggregates neighbor information using a chosen pooling operator and then combines it with the node's current embedding. Since we operate on edge-typed graphs, we write the update per relation $r \in \{c, d\}$, using the relation-specific neighborhood $\mathcal{N}_r(v) = \{u : (u, v) \in E_r\}$

$$\tilde{\mathbf{h}}_v^{(k,r)} = \mathrm{AGG}^{(k,r)}\left(\left\{\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}_r(v)\right\}\right) \tag{1}$$

$$\hat{\mathbf{h}}_v^{(k,r)} = \sigma\left(W_r^{(k)} \cdot \left[\,\mathbf{h}_v^{(k-1)} \,\|\, \tilde{\mathbf{h}}_v^{(k,r)}\,\right]\right) \tag{2}$$

Equation 1 defines the relation-wise neighborhood aggregation, producing $\tilde{\mathbf{h}}_v^{(k,r)}$, while Equation 2 combines the current node embedding with the aggregated neighborhood via concatenation ($\|$), a trainable matrix $W_r^{(k)}$ (relation- and layer-specific), and a nonlinearity $\sigma(\cdot)$. In our implementation, SAGE convolutions are applied independently to conjunctive and disjunctive edges at each layer, and their relation-specific outputs are fused as described in Section 3.3.

3.2 Graph Attention Networks (GAT)

Graph Attention Networks replace uniform neighborhood aggregation with an attention mechanism that learns the relative importance of neighboring nodes. In the edge-typed setting, attention is computed independently per relation $r \in \{c, d\}$.

$$\mathbf{z}_v^{(k,r)} = W_r^{(k)}\,\mathbf{h}_v^{(k-1)} \tag{3}$$

$$e_{uv}^{(k,r)} = \mathrm{LeakyReLU}\left(\mathbf{a}_r^{(k)\top}\left[\,\mathbf{z}_u^{(k,r)} \,\|\, \mathbf{z}_v^{(k,r)}\,\right]\right) \tag{4}$$

$$\alpha_{uv}^{(k,r)} = \frac{\exp\left(e_{uv}^{(k,r)}\right)}{\sum_{w \in \mathcal{N}_r(v)} \exp\left(e_{wv}^{(k,r)}\right)} \tag{5}$$

$$\hat{\mathbf{h}}_v^{(k,r)} = \sigma\left(\sum_{u \in \mathcal{N}_r(v)} \alpha_{uv}^{(k,r)}\,\mathbf{z}_u^{(k,r)}\right) \tag{6}$$

Equation 3 applies a relation-specific linear projection, Equation 4 computes unnormalized attention scores, and Equation 5 normalizes them over the incoming neighborhood $\mathcal{N}_r(v)$. The final attention-weighted aggregation is given in Equation 6. In our architecture, multi-head attention is used independently for each relation type; head outputs are concatenated to preserve hidden dimensionality and capture heterogeneous interaction patterns across precedence and machine-conflict relations.

3.3 Dual-surrogate with convex combination

We represent each JSP instance as an edge-typed directed graph $G = (V, E_c, E_d)$. Each operation is a node $v \in V_{\mathrm{op}}$; we additionally include two dummy nodes $s = \mathit{init}$ and $t = \mathit{end}$, so $V = V_{\mathrm{op}} \cup \{s, t\}$.

$$\begin{aligned} E_c \;=\; & E_{\mathrm{prec}} \;\cup\; E_{\mathrm{prec}}^{-1} \\ & \cup \bigcup_{j \in J} \{(s, v_{j,1}), (v_{j,1}, s)\} \\ & \cup \bigcup_{j \in J} \{(v_{j,|M|}, t), (t, v_{j,|M|})\} \end{aligned} \tag{7}$$

Equation 7 defines the conjunctive edge set E_c by augmenting within-job precedence edges E_{prec} with reverse copies (to allow bidirectional information flow along routes) and with source/sink links through the dummy nodes s and t, without introducing additional relation types. Disjunctive edges E_d encode machine conflicts: for each machine, we connect every pair of operations processed on that machine with bidirectional edges (a directed clique per machine), enabling information exchange among competing operations.

$$\mathbf{h}_v^{(0)} = \psi(x_v) \tag{8}$$

Equation 8 initializes node embeddings via a trainable projection ψ. Each operation node has an input feature vector $x_v \in \mathbb{R}^2$ containing its processing time (min–max normalized using training-set statistics) and a machine identifier normalized to $[0, 1]$ by dividing the machine index by $(|M| - 1)$. Dummy nodes use the zero vector.

Instead of committing to a single aggregation operator, we employ a dual-backbone architecture that processes the same multi-relational graph G with two complementary message-passing operators: SAGE (pooling-based) and GAT (attention-based). The two branches share graph construction, input features, depth, post-processing blocks, pooling, and regression head, differing only in the definition of MP_r.

$$\hat{\mathbf{h}}_{v,S}^{(k,r)} = \text{SAGEConv}_r^{(k)}\left(\mathbf{h}_S^{(k-1)}, E_r\right) \tag{9}$$

$$\mathbf{h}_{v,S}^{(k)} = \phi\left(\frac{\hat{\mathbf{h}}_{v,S}^{(k,c)} + \hat{\mathbf{h}}_{v,S}^{(k,d)}}{2}\right) \tag{10}$$

$$\hat{\mathbf{h}}_{v,A}^{(k,r)} = \text{GATConv}_r^{(k)}\left(\mathbf{h}_A^{(k-1)}, E_r\right) \tag{11}$$

$$\mathbf{h}_{v,A}^{(k)} = \phi\left(\frac{\hat{\mathbf{h}}_{v,A}^{(k,c)} + \hat{\mathbf{h}}_{v,A}^{(k,d)}}{2}\right) \tag{12}$$

Equations 10–12 specify branch-wise message passing: the SAGE branch applies relation-specific SAGEConv layers, while the GAT branch applies relation-specific GATConv layers (with multi-head attention per relation; head outputs are concatenated as in standard GAT).

$$\mathbf{h}_G^S = \frac{\sum_{v \in V_{\text{op}}} \mathbf{h}_{v,S}^{(K)}}{|V_{\text{op}}|} \qquad \mathbf{h}_G^A = \frac{\sum_{v \in V_{\text{op}}} \mathbf{h}_{v,A}^{(K)}}{|V_{\text{op}}|} \tag{13}$$

Equation 13 defines branch-specific graph embeddings using the same global mean pooling over operation nodes.

$$\hat{C}_{\max}^S = f_{\theta_S}\left(\mathbf{h}_G^S\right) \qquad \hat{C}_{\max}^A = f_{\theta_A}\left(\mathbf{h}_G^A\right) \tag{14}$$

$$\hat{C}_{\max} = \alpha \hat{C}_{\max}^S + (1 - \alpha)\hat{C}_{\max}^A \qquad \alpha \in [0, 1] \tag{15}$$

Equation 14 yields one scalar prediction per branch via lightweight regressors. Equation 15 combines both predictions where $\alpha \in [0, 1]$ controls the convex combination of the SAGE and GAT predictions. In principle, α can be treated as a hyperparameter (or learned), yielding a family of models parameterized by the mixing weight. In our experiments, we fix $\alpha = 0.5$ to obtain a simple, parameter-free mixture and to avoid introducing additional tuning degrees of freedom when comparing backbones under identical training conditions.

4 Evaluation

This section describes the evaluation protocol for GNN-based makespan predictors on JSP instances. We report (i) in-distribution accuracy on the held-out test split and (ii) mixed-source generalization on JSPLIB to validate that parsing, normalization, and graph construction remain reliable across heterogeneous standard benchmarks under multi-source training, without instance-specific tuning. We construct both solver-certified *Optimal* and best-found *Feasible* subsets; unless stated otherwise, we report results on *Optimal* instances to isolate performance under high-quality supervision.

4.1 Data setup

We built a unified corpus from two complementary instance sources: a large synthetic set generated with the IGJSP generator [3] and the JSPLIB benchmark suite. We included both sources in all splits, contributing a proportional amount from each dataset. Together, these sources yielded approximately $\approx 65\,000$ solver runs over $\approx 23\,000$ instances spanning a broad range of job–machine configurations and solvers.

For IGJSP, we generated instances by sweeping over a wide range of job sizes (2-50) and machine counts (2-50), and considering multiple processing-time distributions (uniform, normal, and exponential) and three different random seeds. Each job consisted of $|M|$ operations, visited each machine exactly once, and processing times were sampled from the selected distribution. To obtain makespan labels at scale, each instance was solved under size-dependent time limits using three solvers (OR-Tools, Gurobi, and CPLEX), and the best outcome was retained: an instance was *optimal* if optimality was certified by any solver within the time limit, *feasible* if only a best-found solution was obtained, and *null* if no feasible schedule was found.

For JSPLIB, we converted all benchmark instances to the same internal graph representation and labeled them using consolidated makespans from a curated repository. The repository annotates whether the reported makespan is *optimal* (proven/known optimum) or *best-known feasible*; we used this annotation to construct the Optimal and Feasible subsets and treated missing/invalid entries as *null*. Overall, outcomes were approximately $\approx 25\%$ *null*, $\approx 47\%$ *feasible*, and $\approx 28\%$ *optimal*. Null instances were removed, and the remaining instances (All) were split into 80% for training, 5% for validation, and 15% for testing, using stratification to preserve optimal/feasible proportions.

4.2 Model training configuration and environment

All models were implemented in PyTorch Geometric and trained for graph-level makespan prediction with two node features ($d_{\text{in}} = 2$): the processing time (min–max normalized using training-set statistics) and a machine index normalized by $(|M| - 1)$. Training-set normalization statistics were reused unchanged for validation and test. Backbone-specific baselines were obtained by fixing the ensemble weight to $\alpha = 1$ (SAGE) or $\alpha = 0$ (GAT), while DSGNN used the uniform mixture $\alpha = 0.5$ in Equation 15; all settings shared the same hidden size (256), ReLU and dropout (0.2), and train/validation/test splits.

Models were trained using Adam with early stopping (patience 25) and a maximum of 200 epochs (lr $= 10^{-4}$, weight decay 10^{-4}, batch size 64).

Table 1 reports regression metrics on the test split for the two GNN backbones. We report *Loss* as the training objective (mean squared error on the normalized target), alongside MAE/RMSE/MSE/R^2 computed on the original makespan scale. Both models achieved

GNN	Loss	MAE	RMSE	MSE	R^2
GAT	0.5427	263.60	**520.09**	**270 498**	**0.9277**
SAGE	**0.3800**	**248.98**	579.02	335 269	0.9104

Table 1 Test performance of GNN backbones for makespan regression on the testing split.

a strong fit, with R^2 values above 0.91, indicating that a large fraction of the variance in makespan values was captured by the learned representations. Regarding squared-error metrics, GAT achieved a lower RMSE (and MSE), suggesting fewer large deviations on high-makespan instances; in contrast, SAGE yielded smaller typical errors but incurred larger squared penalties, consistent with occasional larger deviations on the largest instances. Since makespan scale grows rapidly with instance size, we report RMSE/MSE for completeness, but rely primarily on the relative-tolerance evaluation below.

4.3 Scheduling graphs accuracy

We assess scheduling-graph accuracy by measuring the percentage of instances whose predicted makespan falls within a relative tolerance of 5%, 10%, and 15% with respect to the target makespan label. Formally, for a tolerance $\tau \in \{0.05, 0.10, 0.15\}$,

$$\mathbb{A}_\tau(D) = \frac{100}{|D|} \sum_{i \in D} \mathbb{I}\left(\frac{\left| \hat{C}_{\max}^{(i)} - C_{\max}^{(i)} \right|}{C_{\max}^{(i)}} \leq \tau \right) \tag{16}$$

where D is the evaluated subset, $C_{\max}$ is the label makespan and $\hat{C}_{\max}$ is the predicted makespan. We report results for solver-certified *Optimal* instances.

α	$\mathbb{A}_{0.05}$	$\mathbb{A}_{0.10}$	$\mathbb{A}_{0.15}$
0	46.68%	78.67%	90.14%
0.25	59.15%	82.70%	90.74%
0.5	65.39%	83.30%	**91.35**%
0.75	**68.81**%	**84.31**%	90.34%
1	64.99%	83.90%	90.95%

Table 2 Tolerance-based accuracy (%) for different weights α in optimal instances.

Table 2 reports tolerance-based accuracy on the merged set of solver-certified optimal instances, as a function of α in the convex combination of SAGE and GAT predictions. The best performance is obtained at $\alpha = 0.5$ for $\tau = 0.15$, with an accuracy of 91.35%. This indicates that combining the two backbones yields a robust predictor whose performance is stable with respect to the mixing weight while allowing moderate gains in specific regimes when α is tuned. Overall, these results support using $\alpha = 0.5$ as a robust default. Consequently, these observations motivate future work on instance-adaptive mixtures that select α based on graph structure.

Table 3 provides a fine-grained view of the *JSPLIB* subset categorized by benchmark family and size $(|J|, |M|)$. For each group, the number of instances N is reported, along with the mean and standard deviation of both predicted and labeled makespans. Additionally, a mean percentage accuracy score is included, which quantifies the average proximity to the reference makespans, as shown in Equation 17.

Instances	N	Jobs	Machines	DSGNN ($\mu \pm \sigma$)	Optimal ($\mu \pm \sigma$)	μAcc.
abz	2	10	10	1031.2 ± 127.6	1088.5 ± 205.8	95.34%
	3	20	15	724.5 ± 24.5	667.0 ± 11.0	91.39%
ft	1	6	6	70.0 ± 0.0	55.0 ± 0.0	72.73%
	1	10	10	853.5 ± 0.0	930.0 ± 0.0	91.77%
	1	20	5	1177.0 ± 0.0	1165.0 ± 0.0	98.97%
la	5	10	5	655.0 ± 51.1	620.2 ± 37.1	94.39%
	5	10	10	853.3 ± 28.5	864.2 ± 61.5	96.18%
	5	15	5	928.3 ± 55.1	917.6 ± 40.5	97.86%
	5	15	10	1031.9 ± 54.1	983.4 ± 54.4	95.01%
	5	15	15	1267.0 ± 56.9	1263.2 ± 79.1	98.53%
	5	20	5	1242.5 ± 102.7	1182.0 ± 94.6	93.48%
	5	20	10	1372.7 ± 54.3	1235.2 ± 74.1	88.74%
	5	30	10	2020.8 ± 115.4	1792.4 ± 75.9	87.27%
orb	10	10	10	827.6 ± 143.9	902.8 ± 187.3	90.74%
swv	5	20	10	1262.9 ± 30.1	1433.6 ± 34.3	88.16%
	5	20	15	1428.3 ± 45.0	1649.8 ± 63.0	86.62%
	10	50	10	2831.9 ± 85.2	2914.8 ± 94.1	95.59%
ta	10	15	15	1213.0 ± 34.4	1228.9 ± 25.1	97.88%
	10	20	15	1423.6 ± 47.1	1364.8 ± 38.7	95.45%
	10	20	20	1644.6 ± 43.2	1615.1 ± 38.9	97.95%
	10	30	15	2075.2 ± 133.9	1788.0 ± 95.3	83.76%
	10	30	20	2311.1 ± 84.4	1910.3 ± 63.9	78.95%
	10	50	15	3143.5 ± 136.7	2773.8 ± 91.1	86.64%
	10	50	20	3848.0 ± 202.7	2843.9 ± 116.3	64.64%
	10	100	20	6111.8 ± 181.8	5365.7 ± 118.3	86.10%
yn	4	20	20	1149.1 ± 10.4	911.8 ± 37.7	73.82%

Table 3 Evaluation of DSGNN in JSPLIB instances grouped by author and size

$$\mu Acc(D) = 100 \cdot \mathrm{Avg}_{i \in D} \left(1 - \frac{\left| \hat{C}_{\max}^{(i)} - C_{\max}^{(i)} \right|}{C_{\max}^{(i)}} \right) \tag{17}$$

which corresponds to $100 \cdot (1 - \text{relative absolute error})$ averaged over D.

The breakdown highlights two sources of variability. First, there is a clear dependency on instance family: families differ in routing patterns and processing-time structure, which translate into distinct graph topologies and label distributions; as a consequence, the regression error is not uniform across authors. Second, the largest configurations exhibit the strongest degradation. In particular, large `ta` groups show both higher dispersion and lower mean accuracy (e.g., `ta` with $(50, 20)$), indicating that prediction becomes more challenging as the scale increases and the disjunctive structure densifies. This is consistent with two effects: large instances are under-represented among solver-certified *optimal* labels under time limits, and the magnitude of the makespan increases rapidly with size, so moderate relative deviations correspond to large absolute gaps, increasing the frequency of tolerance violations.

5 Conclusions

This paper presented DSGNN, a dual-backbone surrogate GNN for instance-level makespan prediction in the classical Job Shop Scheduling Problem. Each JSP instance was encoded as an edge-typed disjunctive/conjunctive scheduling graph, and two complementary message-passing operators (SAGE and GAT) were used to produce independent makespan estimates that were combined through a convex mixture. Our findings suggest that combining complementary aggregation mechanisms yields more robust surrogates than relying on a single backbone, particularly when supervision quality varies across regimes.

In future works, we will work on a natural extension to learn an instance-adaptive mixing coefficient $\alpha(G)$ from graph structure, for instance via a lightweight gating network that takes pooled graph-level embeddings (or simple topology statistics) as input and outputs $\alpha(G) \in (0, 1)$. Concretely, we will condition α on topology-aware summaries (such as degree statistics, machine-clique density, or pooled backbone embeddings) so that the model can emphasize SAGE or attention-based propagation depending on the instance regime. In addition, the disjunctive machine cliques increase edge density for large instances, motivating sparse conflict encodings and scaling studies. Finally, we will investigate integrating DSGNN into heuristic pipelines, to select solvers and time limits per instance, warm-start bounds, or prioritize search regions based on predicted difficulty.

References

1 Anbang Liu, Peter B. Luh, Kailai Sun, Mikhail A. Bragin, and Bing Yan. Integrating machine learning and mathematical optimization for job shop scheduling. *IEEE Transactions on Automation Science and Engineering*, 21(3):4829–4850, 2024. doi:10.1109/TASE.2023.3303175.

2 Chien-Liang Liu and Tzu-Hsuan Huang. Dynamic job-shop scheduling problems using graph neural network and deep reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(11):6836–6848, 2023. doi:10.1109/TSMC.2023.3287655.

3 Christian Pérez, Carlos March, and Miguel A. Salido. Igjsp: A reproducible instance generator for job shop scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '25 Companion, page 159–162, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3712255.3726612.

4 Christian Pérez, Miguel A. Salido, and Carlos March Moya. Optimizing energy efficiency in unrelated parallel machine scheduling problem through reinforcement learning. *Information Sciences*, 693:121674, 2025. doi:10.1016/j.ins.2024.121674.

5 Igor G. Smit, Jianan Zhou, Robbert Reijnen, Yaoxin Wu, Jian Chen, Cong Zhang, Zaharah Bukhsh, Yingqian Zhang, and Wim Nuijten. Graph neural networks for job shop scheduling problems: A survey. *Computers & Operations Research*, 176:106914, 2025. doi:10.1016/j.cor.2024.106914.

6 Hengliang Tang and Jinda Dong. Solving flexible job-shop scheduling problem with heterogeneous graph neural network based on relation and deep reinforcement learning. *Machines*, 12(8):584–584, 2024. doi:10.3390/machines12080584.

7 Katie H. Williams. Graph neural networks-based scheduler for production planning problems using reinforcement learning. *Journal of Manufacturing Systems*, 69:91–102, 2023. doi:10.1016/j.jmsy.2023.06.005.

8 Fangfang Zhang, Yi Mei, Su Duy Nguyen, and Mengjie Zhang. Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling. *IEEE Transactions on Evolutionary Computation*, pages 1–1, 2023. doi:10.1109/tevc.2023.3255246.