Elitist Sequential Monte Carlo Inference for SchedulExpert: A Neural Network Policy for JSSP

Henrik Abgaryan[1], Tristan Cazenave[1], and Ararat Harutyunyan[1]

LAMSADE, Université Paris Dauphine–PSL, CNRS, Paris, France
henrik.abgaryan@dauphine.eu
tristan.cazenave@lamsade.dauphine.fr
ararat.harutyunyan@lamsade.dauphine.fr

Abstract. The Job Shop Scheduling Problem (JSSP) is a classical NP-hard combinatorial optimisation problem with broad industrial relevance. Neural approaches have shown promise, but many rely on unstable reinforcement-learning pipelines or expensive supervised labels, while inference-time sampling has received comparatively little attention. We address both modelling and inference. First, we introduce SchedulExpert, a self-supervised JSSP policy that combines a graph-attention encoder with a Mixture-of-Experts (MoE) architecture and attention-based routing. The model requires no expert labels: it samples schedules, evaluates their makespans, and uses the best sample as a pseudo-label. On Taillard's benchmarks, SchedulExpert is competitive with a strong self-supervised baseline while using 5.22% fewer encoder parameters and 17.66% fewer decoder parameters. Second, we compare ordinary independent sampling with two structured population-based decoders: an elitist Sequential Monte Carlo (SMC) sampler with proxy rollouts and trajectory harvesting, and a Cross–Entropy Method (CEM) decoder based on iterative logit shaping. Across all 80 Taillard instances, paired tests show that SMC significantly improves on ordinary sampling from budget 64 for SchedulExpert. For a second frozen policy, L2D, the improvement is significant at budgets 256–1024, demonstrating portability when the sampling budget is sufficiently large. The frozen CEM configuration improves neither policy. These results show that MoE-based self-supervised policies are competitive for JSSP and that inference-time decoders should be evaluated under a fixed, multiplicity-corrected protocol rather than assumed to improve every policy or budget.

Keywords: Job Shop Scheduling · Neural combinatorial optimisation · Mixture of Experts · Self-supervised learning · Sequential Monte Carlo · Cross–Entropy Method

1 Introduction

The Job Shop Scheduling Problem (JSSP) is a central problem in production optimisation and scheduling. It requires scheduling N jobs, each composed of operations with specified processing times, on a finite set of M machines. The usual objective, and the one considered in this paper, is to minimise the makespan ($C_{\max}$), i.e., the completion time of the last operation. JSSP has widespread relevance in both manufacturing and

service industries, where it influences key outcomes such as operational efficiency, effective resource allocation, and the quality of customer service. JSSP is NP-hard when the number of machines is greater than two [15]. Consequently, computing optimal solutions becomes impractical for many medium and large instances [15, 37]. Historically, methods for addressing JSSP have relied on mathematical programming, dispatching rules and metaheuristic search [18, 37]. Despite their utility, these approaches often struggle to scale effectively and face challenges in handling intricate job–machine precedence constraints, especially in large-scale instances.

Metaheuristic approaches have been widely investigated as an alternative to exact methods for addressing the JSSP [37]. Among these, Priority Dispatching Rules (PDRs) are commonly used heuristic components in practical scheduling systems [39]. Designing an effective rule is nevertheless challenging and traditionally requires substantial domain expertise, especially for complex JSSP instances. Advanced metaheuristics, such as those presented by Nowicki and Smutnicki [24], can produce high-quality solutions within short computational times; however, these methods can be complex to implement and reproduce [4].

Rapid advances in artificial intelligence (AI) have sparked interest in alternative approaches. Neural networks have shown promise for finding near-optimal JSSP solutions [5, 41], with prominent learning paradigms including supervised learning and reinforcement learning (RL). Deep RL, in particular, has motivated several JSSP methods [21, 41], although unstable training dynamics can make these pipelines complex and time-consuming [10, 22].

Although supervised learning avoids many of the instability issues inherent in reinforcement learning, it depends heavily on labelled data. This dependence is challenging for combinatorial optimisation problems such as JSSP: because JSSP is NP-hard [15], generating optimal or near-optimal labels using exact solvers is computationally expensive and often impractical [27]. To address these limitations, semi-supervised learning [25] has gained attention for its ability to use unlabelled data effectively. Moreover, self-supervised learning [9, 28] is emerging as a promising approach for neural combinatorial optimisation problems, providing a practical direction for JSSP and related domains.

Another promising technique to enhance model performance is the Mixture of Experts (MoE) approach. This method combines multiple expert models, each specialising in different regions or aspects of a problem, to improve overall performance [19, 7, 35]. In particular, Cai et al. [7] highlight the effectiveness of MoE in scaling model capacity while incurring minimal computational overhead, making it valuable for large-scale machine learning tasks.

Despite these advantages, the application of the Mixture of Experts framework to combinatorial optimisation problems, including JSSP, remains limited [16]. Expanding its use in this domain could provide useful modelling capacity for complex scheduling and optimisation tasks.

Building on these observations, we propose SchedulExpert, a neural Mixture-of-Experts architecture tailored to JSSP. The model operates on a disjunctive-graph representation and combines a Graph Attention Network (GAT) encoder with multiple expert modules and an attention-based router, followed by a multi-head attention de-

coder. The experts provide distinct transformations of the job–machine representation, and the router learns their relative contributions for each input. To avoid expensive labels, we train SchedulExpert in a self-supervised manner: for each instance, the policy generates multiple candidate schedules and uses the one with the smallest makespan as a pseudo-label, following recent self-improvement schemes for neural combinatorial optimisation [9, 28].

A second focus of this work is the *inference-time* use of a trained policy. Most existing neural JSSP approaches rely on ordinary Monte Carlo sampling, i.e., drawing multiple independent trajectories from the policy and keeping the best schedule. In contrast, we ask how far one can push performance by changing only the sampling strategy, without modifying or retraining the underlying network. To this end, we use SchedulExpert as the policy inside three decoders: (i) ordinary independent sampling, (ii) a Sequential Monte Carlo (SMC) sampler with elitist resampling and trajectory harvesting, and (iii) a Cross–Entropy Method (CEM) decoder that iteratively reshapes the logits based on elites. All three methods share the same policy network and differ only in how they allocate a fixed computational budget across trajectories.

We evaluate SchedulExpert on Taillard's JSSP benchmarks [33] against classical heuristics, metaheuristics and recent neural baselines [41, 14, 26, 40, 17]. For the sampling study, we introduce a budget-fair protocol that measures the number of effective forward calls to the policy network and compare ordinary sampling, SMC and CEM under the same budgets. Our main contributions are:

- a graph-based Mixture-of-Experts architecture with attention routing, trained end-to-end by self-supervised pseudo-labelling;
- an empirical study of inference-time sampling strategies for a fixed neural policy, comparing ordinary sampling, elitist SMC and CEM under a common budget definition;
- experimental evidence on Taillard's benchmarks that SchedulExpert is competitive with strong heuristic and neural baselines, and that SMC can reduce optimality gaps over ordinary sampling at sufficiently large inference budgets without retraining the policy.

The rest of the paper is organised as follows. Section 2 reviews related neural, heuristic and sampling approaches. Section 3 defines the JSSP and the graph representation used by our model. Section 4 details the architecture and training procedure. Section 5 reports the comparison with classical and neural baselines, and Section 6 analyses the inference-time sampling schemes. Section 7 concludes the paper.

2 Related Work

Deep learning (DL) has become an important tool across many areas of artificial intelligence. A notable example is the work of Lin et al. [20], where the authors employed a deep Q-network (DQN) to solve scheduling problems in semiconductor manufacturing, illustrating the potential of Deep Reinforcement Learning (DRL) for industrial applications. Building on this line of research, Zhang et al. [41] proposed a DRL-based

framework that uses Priority Dispatching Rules (PDRs) for the JSSP. Their method recasts the scheduling task as a Markov Decision Process (MDP) and leverages a disjunctive graph representation to capture both machine states and operation dependencies. To process this representation, they introduced a Graph Isomorphism Network (GIN), whose resulting embeddings feed into a policy network trained with Proximal Policy Optimisation (PPO) [32].

Subsequent works have focused on improving the state representation of JSSP to enhance the learning process. For instance, Yuan et al. [38] expanded the feature space and adopted a bidirectional scheduling strategy within an MDP framework, reducing the risk of multiple equivalent optimal actions. They also introduced Invalid Action Masking (IAM) to remove infeasible choices, effectively narrowing the search space and guiding the policy towards better solutions. Meanwhile, Drakulic et al. [12] tackled combinatorial optimisation problems (COPs) more generally by modelling them as MDPs and applying Bisimulation Quotienting (BQ) to exploit problem symmetries. This technique not only reduces computational overhead but also improves generalisation to larger problem instances. Recent work has also explored whether large language models can produce schedules directly [1].

Various strategies for sequentially constructing JSSP solutions have been explored. Single-shot (greedy) policies, guided by neural networks, are used in prior work [41, 38], while neural construction methods sample solutions from network probability distributions [3]. Monte Carlo Tree Search (MCTS) [6] and its variants [8] leverage rollouts but require significant resources. Beam search [13] greedily expands partial solutions, also demanding high computational effort. Randomised Greedy Sampling [2] explores deviations from heuristics to identify better solutions.

Recent JSSP-specific neural baselines used in our experiments include ScheduleNet, NeuroLS and L2S. ScheduleNet [26] uses a decentralised graph-attention policy for multi-agent scheduling. NeuroLS [14] learns to control local-search components, including neighbourhood and acceptance choices. L2S [40] combines deep reinforcement learning with an improvement heuristic, while curriculum-learning approaches study how dispatching policies generalise across instance sizes [17]. These methods motivate our comparison in Section 5: they cover constructive dispatching policies, learned local search, and improvement-based neural solvers.

Recent work continues to broaden the learning paradigms and JSSP settings considered. Van Remmerden et al. [29] study offline reinforcement learning for dispatching, while Zhang et al. [42] consider structural uncertainty with an asymmetric actor–critic architecture. Their training information, uncertainty assumptions, evaluation protocols, and sampling budgets differ from ours, so we discuss them as related work but do not combine their reported results with our numerical comparison. The comparison instead comprises methods with directly comparable Taillard results for the standard static JSSP and is not an exhaustive ranking of recent JSSP solvers.

Dynamic JSSP and flow-shop scheduling are related settings that share NP-hard structure with the standard JSSP [34, 30]. Flow-shop scheduling is a restricted shop-scheduling variant, whereas dynamic JSSP introduces time-dependent information or events.

The mathematical-programming and constraint-programming values in our comparison are the Taillard shape-level results reported by Corsini et al. [9]. Their MIP baseline uses Gurobi 9.5 with Manne's disjunctive formulation, as described by Ku and Beck [18], while their constraint-programming baseline uses Google OR-Tools 9.8 CP-SAT; both have a 3,600-second time limit. These published baselines provide reproducible reference points but are not new runs under our hardware environment. We therefore restrict cross-method claims to the reported methods and do not claim superiority over untested solvers such as IBM CP Optimizer or Hexaly.

A growing branch of literature addresses the cost of expert labels by adopting self-supervised training. For example, Corsini et al. [9] introduced a Pointer Network-based generative model that iteratively refines multiple solutions and selects the best one as a pseudo-label. This method outperforms many heuristics and DRL approaches on standard JSSP benchmarks, all without relying on expensive ground-truth solutions. An important factor in these self-improvement schemes is the method used to generate pseudo-labels. Specifically, Pirnay and Grimm [28] presented a problem-independent sequence decoding approach aimed at boosting solution diversity. By sampling without replacement and excluding previously generated sequences, their method broadens the search space and improves solution quality. Experimental results on the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP) show the robustness of this idea.

Our work builds on these self-supervised approaches [9, 28]. We retain the sampling-based pseudo-labelling framework of Corsini et al. [9], introduce a Mixture-of-Experts policy architecture, and study how structured inference-time decoders affect solution quality without retraining the policy.

3 Preliminaries

Definition 1 (Job-Shop Scheduling Problem). *A JSSP instance consists of a set of jobs $J = \{1, \dots, N_J\}$ and machines $\mathcal{M} = \{1, \dots, N_M\}$. Each job i is an ordered sequence of operations $(O_{i1}, \dots, O_{in_i})$. Operation O_{ij} must be processed without preemption for $p_{ij} \in \mathbb{N}$ time units on a prescribed machine $m_{ij} \in \mathcal{M}$, and it cannot start before $O_{i,j-1}$ has completed. Each machine processes at most one operation at a time. A feasible schedule assigns a start time $S_{ij} \geq 0$ to every operation such that all precedence and machine-capacity constraints are satisfied. The objective considered in this paper is to minimise the makespan*

$$C_{\max} = \max_{i,j} C_{ij}, \qquad C_{ij} = S_{ij} + p_{ij}.$$

For the standard Taillard benchmarks used in our experiments, all jobs have $n_i = N_M$ operations, so an instance shape is denoted by $N_J \times N_M$.

A JSSP instance can also be represented by a disjunctive graph [41]. Let $O = \{O_{ij} : i \in J,\, j = 1, \dots, n_i\} \cup \{S, T\}$ contain all operations and two zero-duration dummy operations denoting the start and end of the schedule. The disjunctive graph $G = (O, C, D)$ is a mixed graph in which the directed arcs C encode precedence constraints within jobs and the undirected edges D connect operations that require the

same machine. A feasible schedule corresponds to an acyclic orientation of the disjunctive edges [41]. The Markov Decision Process (MDP) formulation, state representation, action space, and state transition follow Zhang et al. [41]. An action $a_t \in A_t$ is an eligible operation at decision step t.

4 Mixture of Experts with Attention-Based Routing

SchedulExpert augments its graph encoder with a Mixture-of-Experts (MoE) component. Multiple *experts* transform the graph representation, and an attention-based router assigns input-dependent weights to their outputs.

The MoE component consists of:

- **Expert modules**: two-layer multilayer perceptrons (MLPs) that learn distinct transformations of their inputs;
- **Attention router**: a learnable mechanism that computes a per-input weight vector over a second set of routing experts and aggregates their outputs.

4.1 Expert Modules

Let $\{\text{Expert}_i\}_{i=1}^{M}$ be the set of M expert modules, each parameterised by a two-layer MLP. For an input feature vector $\mathbf{x} \in \mathbb{R}^{D_{\text{in}}}$, the i-th expert outputs:

$$\mathbf{z}_i^{(1)} = \sigma\big(\mathbf{W}_i^{(1)}\,\mathbf{x} + \mathbf{b}_i^{(1)}\big), \tag{1}$$

$$\mathbf{z}_i^{(2)} = \mathbf{W}_i^{(2)}\,\mathbf{z}_i^{(1)} + \mathbf{b}_i^{(2)}, \tag{2}$$

where $\mathbf{W}_i^{(1)} \in \mathbb{R}^{H \times D_{\text{in}}}$ and $\mathbf{W}_i^{(2)} \in \mathbb{R}^{D_{\text{out}} \times H}$ are weight matrices, $\mathbf{b}_i^{(1)} \in \mathbb{R}^{H}$ and $\mathbf{b}_i^{(2)} \in \mathbb{R}^{D_{\text{out}}}$ are bias terms, and $\sigma(\cdot)$ is a non-linear activation function (e.g., ReLU). We denote the final output of expert i by:

$$\mathbf{z}_i = \text{Expert}_i(\mathbf{x}) \;=\; \mathbf{z}_i^{(2)} \;\in\; \mathbb{R}^{D_{\text{out}}}. \tag{3}$$

Thus, each expert produces a D_{out}-dimensional representation of the input $\mathbf{x}$.

4.2 Attention-Based Router

The router computes a distribution over M routing experts for each input. Let $\mathbf{e} \in \mathbb{R}^{D_{\text{embed}}}$ be the specialised embedding produced from the GAT representation. Routing proceeds in three stages: multi-head self-attention, expert-score calculation, and weighted aggregation.

Multi-Head Self-Attention The router first produces query, key, and value vectors from $\mathbf{e}$:

$$\mathbf{Q} = \mathbf{W}_Q\,\mathbf{e}, \quad \mathbf{K} = \mathbf{W}_K\,\mathbf{e}, \quad \mathbf{V} = \mathbf{W}_V\,\mathbf{e}, \tag{4}$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{H_{\text{attn}} \times D_{\text{embed}}}$ are learnable matrices. The implementation treats each node embedding as a sequence of length one.

The multi-head attention output $\mathbf{a}$ is computed by:

$$\mathbf{a} = \text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}), \tag{5}$$

where $\text{MHA}(\cdot)$ denotes multi-head attention:

$$\mathbf{a} = \left[\text{head}_1 \,||\, \text{head}_2 \,||\, \ldots \,||\, \text{head}_h\right] \mathbf{W}_O, \tag{6}$$

$$\text{head}_j = \text{softmax}\Big(\frac{\mathbf{Q}_j \mathbf{K}_j^\top}{\sqrt{d_j}}\Big) \mathbf{V}_j, \tag{7}$$

with h denoting the number of heads, $d_j = H_{\text{attn}}/h$ the dimension of each head, and $\mathbf{W}_O \in \mathbb{R}^{H_{\text{attn}} \times H_{\text{attn}}}$ an output projection matrix. Layer normalisation is then applied to stabilise training:

$$\mathbf{a}_{\text{norm}} = \text{LayerNorm}\big(\mathbf{a}\big). \tag{8}$$

Expert Score Calculation The router generates expert-specific scores from the normalised attention output:

$$\mathbf{s} = \mathbf{W}_S \, \mathbf{a}_{\text{norm}} + \mathbf{b}_S \quad \in \quad \mathbb{R}^M, \tag{9}$$

where $\mathbf{W}_S \in \mathbb{R}^{M \times H_{\text{attn}}}$ and $\mathbf{b}_S \in \mathbb{R}^M$ are learnable parameters. We then apply a softmax function to obtain a probability distribution over the M experts:

$$\mathbf{w} = \text{softmax}(\mathbf{s}) \quad \in \quad \mathbb{R}^M, \tag{10}$$

where w_i is the weight assigned to expert i.

Weighted Aggregation Each routing expert receives the same attention-processed input $\mathbf{a}_{\text{norm}} \in \mathbb{R}^{H_{\text{attn}}}$ and produces

$$\mathbf{z}_i = \text{Expert}_i\big(\mathbf{a}_{\text{norm}}\big), \tag{11}$$

where $\mathbf{z}_i \in \mathbb{R}^{H_{\text{attn}}}$.

The router output is the weighted sum

$$\mathbf{z}_{\text{routed}} \;=\; \sum_{i=1}^{M} w_i \, \mathbf{z}_i \quad \in \quad \mathbb{R}^{H_{\text{attn}}}. \tag{12}$$

The vector $\mathbf{z}_{\text{routed}}$ is the routed embedding used by the decoder.

4.3 Incorporation into the GAT Pipeline

The MoE component follows the final GAT layer. The encoder represents each JSSP instance as a disjunctive graph and uses the same 11 hand-crafted state features as the

self-labelling baseline of Corsini et al. [9]. Fixing the input representation controls the model comparison: performance differences can be attributed more directly to the architecture and routing mechanism rather than to a richer feature set. Features 1–6 describe the readiness and relative completion of a job's next operation, whereas features 7–11 describe the relative availability and load of its required machine. Ratios and differences relative to the current makespan, means, and quartiles reduce sensitivity to the absolute processing-time scale. Because the feature representation is fixed in this study, we do not assess the importance of individual features or claim that the set is optimal; comparing it with learned state representations is an avenue for future work. The 11 features are:

1. $C_{o(t,j)-1}(\pi_{<t})$ minus the completion time of machine $\mu_{o(t,j)}$, representing job j's idle time if scheduled at step t;
2. $C_{o(t,j)-1}(\pi_{<t})$ divided by the makespan of $\pi_{<t}$, measuring how close job j's completion time is to the current makespan;
3. $C_{o(t,j)-1}(\pi_{<t})$ minus the mean completion time of all jobs;

4–6. the differences between $C_{o(t,j)-1}(\pi_{<t})$ and the first, second, and third quartiles of job completion times;

7. the completion time of $\mu_{o(t,j)}$ divided by the makespan of $\pi_{<t}$;
8. the completion time of $\mu_{o(t,j)}$ minus the mean completion time of all machines;

9–11. the differences between the completion time of $\mu_{o(t,j)}$ and the first, second, and third quartiles of machine completion times.

Specifically, we take the output $\mathbf{H}^{(2)} \in \mathbb{R}^{N \times E_{\text{embed}}}$ of the second GAT layer, partition each node embedding across M preliminary expert modules, and feed each partition to its corresponding module. Let

$$\mathbf{H}_n^{(2)} \;=\; [\,\mathbf{p}_1 \,||\, \mathbf{p}_2 \,||\, \ldots \,||\, \mathbf{p}_M] \;\in\; \mathbb{R}^{E_{\text{embed}}},$$

where $\mathbf{p}_i \in \mathbb{R}^{E_{\text{embed}}/M}$ is the portion of node n's embedding assigned to preliminary expert i. We concatenate the resulting specialised representations and pass them to the attention router defined above. The router applies its own set of weighted experts to produce the final representation $\mathbf{z}_{\text{routed}}$:

$$\mathbf{z}_i = \text{Expert}_i(\mathbf{p}_i), \quad i = 1, 2, \ldots, M, \tag{13}$$

$$\mathbf{z}_{\text{routed}} = \text{Router}\Big([\,\mathbf{z}_1 \,||\, \ldots \,||\, \mathbf{z}_M]\Big). \tag{14}$$

Each preliminary expert processes its partition independently. The routed output is concatenated with the original node features $\mathbf{X}_n$ to produce the final node embedding used by the decoder:

$$\mathbf{h}_n = \big[\mathbf{X}_n \,||\, \mathbf{z}_{\text{routed}}\big]. \tag{15}$$

4.4 Decoder Architecture

At each time step t, the decoder generates a probability distribution over the eligible jobs by leveraging both job embeddings $\mathbf{e}_i$ and solution-related features. The decoder is composed of two main components:

Memory Network. The memory network generates a state $\mathbf{s}_j \in \mathbb{R}^d$ for each job $j \in J$ from the partial solution $\pi_{<t}$.

First, we extract a context vector $\mathbf{c}_j$ from $\pi_{<t}$ (as in [9]), which encodes handcrafted features indicating the status of job j. These vectors are fed into a Multi-Head Attention layer (MHA) followed by a non-linear projection:

$$\mathbf{s}_j \;=\; \mathrm{ReLU}\Big(\big[\mathbf{c}_j\mathbf{W}_1 \;+\; \mathrm{MHA}_{b\in J}\big(\mathbf{c}_b\mathbf{W}_1\big)\big]\mathbf{W}_2\Big), \tag{16}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are projection matrices. The MHA module accounts for the entire set of jobs J when generating $\mathbf{s}_j$, following the idea in [9].

Classifier Network. The classifier outputs the probability p_j of selecting job j by combining the embedding $\mathbf{e}_{o(t,j)}$ of its *ready* operation with the state $\mathbf{s}_j$ from the memory network. We concatenate $\mathbf{e}_{o(t,j)}$ and $\mathbf{s}_j$, then apply a feed-forward network (FNN):

$$z_j \;=\; \mathrm{FNN}\Big(\big[\mathbf{e}_{o(t,j)} \,\|\, \mathbf{s}_j\big]\Big). \tag{17}$$

Let $J_t \subseteq J$ be the set of jobs with an eligible next operation at step t. For $j \in J_t$, the scalar $z_j \in \mathbb{R}$ is transformed by a masked softmax:

$$p_j \;=\; \frac{e^{z_j}}{\sum_{b\in J_t} e^{z_b}}, \qquad p_j = 0 \text{ for } j \notin J_t.$$

The job selected at step t is sampled from this distribution.

4.5 Benefits and Intuition

Parameter Efficiency. SchedulExpert uses fewer parameters than the self-labelling model of Corsini et al. [9] while achieving comparable performance (Table 2). With matched input, hidden, and embedding sizes, its encoder has 158,084 parameters instead of 166,784, a reduction of 8,700 parameters (5.22%). Its decoder has 173,057 parameters instead of 210,177, a reduction of 37,120 parameters (17.66%).

Specialisation. The distinct expert transformations allow the model to represent different job–machine interaction patterns.

Dynamic Routing. The attention router assigns input-dependent weights to the routing experts, providing more flexible aggregation than static parameter sharing.

4.6 Integrated Architecture

The complete architecture combines the GAT encoder, the preliminary expert transformations, the attention router, and the decoder. For each node n:

1. **GAT Layers:**

$$\mathbf{H}^{(1)} = \mathrm{GAT}_1(\mathbf{X}, \mathbf{E}), \quad \mathbf{H}^{(2)} = \mathrm{GAT}_2\big([\mathbf{X} \,||\, \mathbf{H}^{(1)}], \mathbf{E}\big).$$

2. **MoE Module:**

$$\mathbf{z}_{\text{routed}}, \mathbf{w} = \text{Router}\Big(\mathbf{H}_n^{(2)}\Big), \quad \mathbf{h}_n = [\,\mathbf{X}_n \,||\, \mathbf{z}_{\text{routed}}\,].$$

3. **MHA decoder:**

$$\text{logits}_n = \text{MHAdecoder}\big(\mathbf{h}_n, \mathbf{s}_n\big).$$

Here, $\mathbf{s}_n$ denotes the partial-schedule context used by the decoder. The resulting logits define the masked distribution over eligible jobs.

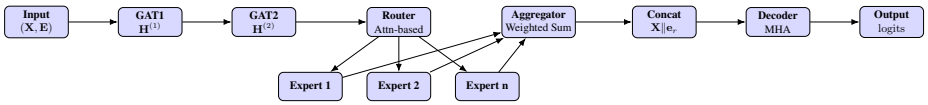


Fig. 1: SchedulExpert architecture. The diagram focuses on the attention router and its weighted routing experts; the partition-specific transformations applied after GAT2 are described in Section 4.3.

4.7 Training Details

The complete model is trained end-to-end by backpropagation, following the self-labelling procedure of Corsini et al. [9]. For each training instance, the current policy generates S schedules; the schedule with the smallest makespan becomes the pseudo-label for a cross-entropy update. Repeating this process allows the pseudo-labels to improve with the policy. Training used one NVIDIA RTX A6000 GPU with 48 GB of memory and



Fig. 2: Average percentage gap during training and validation.

consumed approximately 35 GB of VRAM. Following Corsini et al. [9], the training set contains 30,000 randomly generated instances: 5,000 instances for each of the shapes 10×10, 15×10, 15×15, 20×10, 20×15, and 20×20. We trained the GAT encoder, MoE component, and MHA decoder jointly with the Adam optimiser. The encoder hidden and output dimensions were 64. The memory-network hidden and output dimensions, classifier hidden size, and latent dimension were also 64. The MoE used four experts. Training used a learning rate of 2×10^{-4}, a batch size of 128, gradient clipping, and at most 20 epochs. A plateau-based scheduler reduced the learning rate to a minimum of 10^{-6} according to validation performance. The self-labelling sample count was $S = 512$. We selected the final checkpoint using the validation metric.

5 Experimental Results

We evaluate SchedulExpert on all 80 Taillard (TA) benchmark instances [33]. Unless stated otherwise, SchedulExpert uses ordinary sampling with 512 trajectories and returns the schedule with the smallest makespan. For an algorithmic makespan C and the optimal or best-known reference makespan $C^\star$, the percentage gap is

$$\text{gap} = 100 \times \left(\frac{C}{C^\star} - 1 \right). \tag{18}$$

Lower values are better.

Unless otherwise stated, tables report averages over the Taillard instances of each shape. For sampling-based methods we repeat decoding with the stated sample budget and report the best schedule obtained under that budget; Section 6 additionally reports variability across instances and repetitions. The values copied from historical baselines are published shape-level aggregates, and per-instance repeated-run observations are not available consistently across methods. A valid paired significance test against those baselines therefore cannot be reconstructed from the reported aggregates. We accordingly treat Tables 1 and 2 as descriptive comparisons and avoid claims of statistical significance against historical baselines. In contrast, for our three-decoder sampling study we retain one row for every instance, method, budget, and seed; this permits the paired tests and multiplicity correction reported in Section 6.4.

We first describe the baseline methods and then compare their published Taillard results with SchedulExpert.

5.1 Priority Dispatching Rules

We begin by introducing the notation used in these rules:

$$\begin{aligned} Z_{ij} &: \text{ the priority index of operation } O_{ij}, \\ n_i &: \text{ the number of operations for job } J_i, \\ p_{ij} &: \text{ the processing time of operation } O_{ij}. \end{aligned}$$

Using this notation, the decision rule for each baseline is:

– **Shortest Processing Time (SPT):**

$$\min Z_{ij} \;=\; p_{ij}.$$

– **Most Work Remaining (MWKR):**

$$\max Z_{ij} \;=\; \sum_{k=j}^{n_i} p_{ik}.$$

– **Most Operations Remaining (MOPNR):**

$$\max Z_{ij} \;=\; n_i - j + 1.$$

– **Tabu search** [23] uses a specific neighbourhood definition based on critical paths and blocks of operations.

Table 1 separates greedy decoding from sampling-based evaluation. Most sampling baselines use $s = 128$ trajectories; for SchedulExpert, we report both $s = 128$ and $s = 512$. In the greedy setting, SchedulExpert obtains the smallest average gap and is strongest on most instance shapes, with particularly good performance on 30×15 and 30×20. Validation over the SchedulExpert sampling parameters selected $\texttt{top_k} = 0.5$ and $\texttt{temperature} = 0.5$. SchedulExpert with $s = 512$ obtains the smallest average gap among the sampling-based neural and heuristic methods in the table and improves on its $s = 128$ result. Row-wise minima are highlighted in green and marked with an asterisk.

In Table 2, we compare SchedulExpert with neural methods and the MIP and CP-SAT baselines reported by Corsini et al. [9]. SchedulExpert$_{512}$ is the best neural method on every shape. CP-SAT achieves the smallest gap on seven of the eight shapes and the smallest average gap overall, whereas SchedulExpert$_{512}$ has the smallest reported gap on 100×20. The result therefore establishes competitiveness among the reported neural methods rather than superiority over exact or constraint-programming solvers. On 100×20, SchedulExpert$_{512}$ achieves a gap of 2.5, compared with 7.9 for L2S, 11.0 for MIP and 3.9 for CP-SAT. Figure 3 compares the average gap with aggregate decoding time over all 80 Taillard instances. Small sample counts ($s \in \{1, 4, 8\}$) are fast but produce average gaps of approximately 12–15%. Increasing s lowers the average gap to about 8% at $s = 1024$, for an aggregate decoding time of more than seven minutes, or approximately five seconds per instance. Figure 4 compares model size and average gap for SchedulExpert and SLJ at $s = 512$. SchedulExpert achieves a comparable average gap while using 17.66% fewer decoder parameters than SLJ [9].

5.2 Neural and Solver Baselines

– **L2D** [41] models JSSP as an MDP where each step selects an operation, updating a disjunctive graph $G(t)$ and state s_t. The reward $R(a_t, s_t) = H(s_t) - H(s_{t+1})$ guides makespan minimisation via policy $\pi(a_t \mid s_t)$. L2D employs a GIN [36] to encode graph-structured data. Node embeddings update iteratively:

$$h_v^{(k)} = \mathrm{MLP}_{\theta_k}\Big((1 + \epsilon^{(k)})h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)}\Big). \tag{19}$$

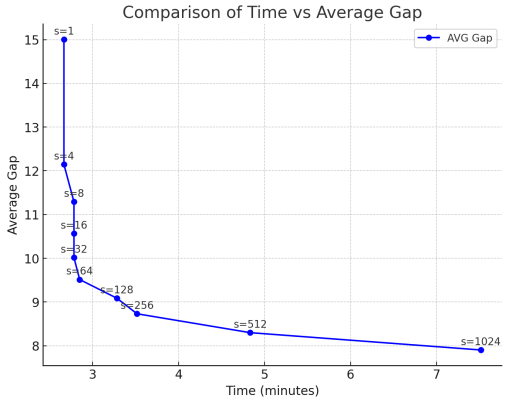


Fig. 3: Runtime for different sample sizes and their effect on the average gap (lower is better). Each point reports average time versus average percentage gap on 80 instances from the Taillard [33] dataset.

Global embeddings guide action selection via MLP and softmax. Training uses PPO-based [32] actor-critic with a shared GIN backbone.

- **RASCL** [17] uses Reinforced Adaptive Staircase Curriculum Learning to adjust task difficulty dynamically and revisit challenging instances during policy training.
- **L2S** [40] constructs an initial JSSP solution with a GNN and refines it iteratively through a learned improvement heuristic.
- **NeuroLS** [14] presents a GNN-based controller that dynamically selects acceptance criteria, neighbourhood operators, and perturbation strategies within a local search framework to effectively solve combinatorial optimisation problems.
- **ScheduleNet** [26] uses an RL-based decentralised scheduler and type-aware graph attention to coordinate multiple agents.
- The **MIP** and **CP-SAT** comparison values are reproduced from Corsini et al. [9]. Their MIP implementation uses Gurobi 9.5 with Manne's disjunctive formulation described by Ku and Beck [18]; their constraint-programming implementation uses Google OR-Tools 9.8 CP-SAT. Both solvers use a 3,600-second time limit.

6 Extended Analysis of Sampling Strategies

The previous sections evaluated SchedulExpert under ordinary sampling, where the trained policy network is queried multiple times and the schedule with the smallest makespan is retained.[1] In this section, we extend the study by introducing two structured sampling schemes: (i) a Sequential Monte Carlo (SMC) [11] sampler with elitist resampling and trajectory harvesting, and (ii) a Cross–Entropy Method (CEM) [31] sampler based on iterative logit shaping. We use the same trained SchedulExpert model

[1] We use the same trained SchedulExpert model and training protocol as in the rest of the paper.

	Greedy							With sampling ($s = 128$)						
Shape	**SPT**	**MWKR**	**MOPNR**	**Tabu**	**L2D**	**RASCL**	**SE**	**SPT**	**MWKR**	**MOPNR**	**L2D**	**RASCL**	**SE$_{128}$**	**SE$_{512}$**
15×15	26.1	19.1	20.4	14.4	26.0	**14.3***	14.7	13.5	13.5	12.5	17.1	19.1	7.5	**6.5***
20×15	32.3	23.3	24.9	18.9	30.0	16.5	**14.8***	18.4	17.2	16.4	23.7	16.0	10.0	**9.1***
20×20	28.3	21.8	22.9	17.3	31.6	**17.3***	18.3	16.7	15.6	14.7	22.6	19.0	10.1	**9.3***
30×15	35.0	24.1	22.9	21.1	33.6	18.5	**18.5***	23.2	19.0	17.3	24.4	14.0	11.8	**11.1***
30×20	33.4	24.8	26.2	20.7	36.3	21.5	**20.5***	23.6	19.9	20.4	28.4	16.1	13.9	**12.8***
50×15	24.0	16.4	17.6	15.9	22.4	12.2	**12.2***	14.1	13.5	13.5	17.1	19.3	7.8	**7.0***
50×20	25.6	17.8	16.8	20.3	26.5	**13.2***	14.0	17.6	14.6	14.0	20.4	9.9	8.4	**7.7***
100×20	14.0	8.3	8.7	13.5	13.6	5.9	**5.8***	10.4	7.0	7.1	13.3	4.0	2.6	**2.5***
Avg	**27.4**	**19.5**	**20.1**	**18.0**	**27.1**	**14.9**	**14.8***	**17.2**	**15.0**	**14.5**	**14.8**	**10.5**	**9.0**	**8.2***

Table 1: Average percentage gap comparison between dispatching rules, tabu search, L2D, RASCL, and SchedulExpert (SE). In each row, the best value is highlighted in green and marked with an asterisk. Sampling generates multiple solutions with a fixed method and selects the one with the smallest makespan.

Shape	**L2S**	**NeuroLS**	**MIP**	**CP-SAT**	**SN**	**SE$_{512}$**
15×15	9.3	7.7	0.1	**0.1***	15.3	6.5
20×15	11.6	12.2	3.2	**0.2***	19.4	9.1
20×20	12.4	11.5	2.9	**0.7***	17.2	9.3
30×15	14.7	14.1	10.7	**2.1***	19.1	11.1
30×20	17.5	16.4	13.2	**2.8***	23.7	12.8
50×15	11.0	11.0	12.2	**3.0***	13.9	7.0
50×20	11.8	11.2	13.6	**2.8***	13.5	7.7
100×20	7.9	5.9	11.0	3.9	6.7	**2.5***
Avg	**12.2**	**11.3**	**8.4**	**2.0***	**16.1**	**8.2**

Table 2: Average percentage gap comparison between neural methods, MIP, CP-SAT, and SchedulExpert (SE). The best value in each row is highlighted in green and marked with an asterisk; the tied MIP value for 15×15 is not highlighted separately. The MIP and CP-SAT values are reproduced from Corsini et al. [9]: Gurobi 9.5 with Manne's disjunctive MIP formulation [18] and Google OR-Tools 9.8 CP-SAT, respectively, each with a 3,600-second limit.

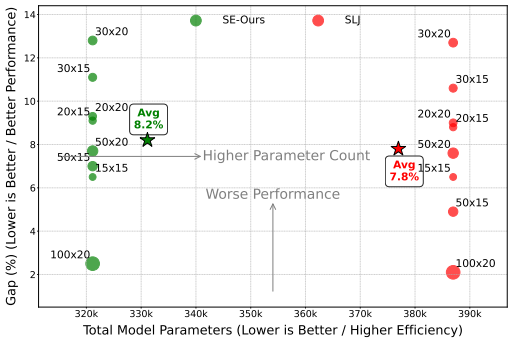


Fig. 4: Average gap versus total parameter count for SchedulExpert with $s = 512$ and SLJ [9] on all 80 Taillard instances [33]. The average across instance shapes is marked with a star.

for all experiments, changing only the sampling strategy at inference time. The sampling study uses the same Taillard benchmark set as Section 5: the 80 Taillard instances grouped by the same eight instance shapes. The reported plots aggregate the retained summaries over these same shapes for each method and budget. This design isolates the effect of the inference procedure from changes in training data, architecture or benchmark distribution. This section addresses two questions: (i) how much additional performance can be extracted from the same policy network by using more sophisticated sampling schemes, and (ii) how the different strategies trade off solution quality against wall-clock time under a common notion of computational budget.

6.1 Sampling algorithms

Given a JSSP instance, the encoder produces node embeddings on the disjunctive graph, and the decoder assigns a probability distribution over feasible jobs at each decision step, conditioned on the current partially built schedule and on invalid action masking. The algorithms below describe how these probabilities are used to generate multiple trajectories under a fixed budget.

Ordinary sampling Ordinary sampling performs N_{eff} independent rollouts of the policy and returns the best schedule among them. It corresponds to the sampling procedure already used in the self–labelling stage, but here the budget is explicitly controlled through β.

Algorithm 1 Ordinary sampling

Require: Instance I, encoder f_θ, decoder g_ϕ, batch size B.
1: Encode the instance once: $E \leftarrow f_\theta(I)$.
2: Initialise B parallel job–shop states and feasibility masks.
3: **for** $t = 1, \ldots, T$ **do**
4: For each particle, identify the ready operation and its features from E.
5: Use the decoder to obtain masked logits over feasible jobs.
6: Convert logits to probabilities with a softmax.
7: Sample one job per particle and update all states and masks.
8: **end for**
9: For each particle, complete the schedule and compute its makespan.
10: Return the trajectory with the smallest makespan and its value.

Sequential Monte Carlo with elitism and trajectory harvesting The Sequential Monte Carlo sampler [11] maintains a population of particles that jointly explore the space of partial schedules. Sampling is performed with a temperature controlled softmax; at a small number of decision steps ("checkpoints") the algorithm computes a fast low-temperature completion of all particles to obtain proxy makespans. These proxy costs are turned into importance weights, which are used to trigger resampling when the effective sample size falls below a threshold. Resampling is *elitist*: a fixed fraction of the best particles is always preserved, while the remaining particles are resampled from the weighted population. In addition, the best complete trajectory seen so far is cached and may be injected back into the final population (trajectory harvesting).

Cross–Entropy Method with soft logit shaping The Cross–Entropy Method [31] variant used here can be interpreted as an iterative logit–shaping procedure. At each round, it decodes a batch of trajectories using the original policy logits, plus an additive guidance term that depends on the empirical frequencies of elite trajectories from past rounds. Guidance is represented as a matrix $G \in \mathbb{R}^{T \times N_J}$, where $G_{t,j}$ stores the log–probability bias of choosing job j at decision step t. The matrix is initialised to zero and updated as a moving average of elite log–frequencies. Decoding uses either Gumbel–Max sampling (for approximately greedy selection) or standard multinomial sampling.

6.2 Budget definition and evaluation protocol

We consider the decoding of a JSSP instance as a generative process that constructs a sequence of job decisions until a complete schedule is obtained. At each decision step, the decoder performs a *forward call* of the policy network (encoder + decoder) for a batch of partial schedules. We define the basic unit of computational budget as:

Definition 2 (Sampling budget). *A single* unit of sampling budget *corresponds to one forward call of the policy network for a batch of particles. All reported budgets β in this section are measured in this unit.*

Algorithm 2 Sequential Monte Carlo (SMC) sampling with elitism and trajectory harvesting

Require: Instance I, encoder f_θ, decoder g_ϕ, batch size B, number of checkpoints n_{cp}, temperature schedule $(\tau_{\text{start}}, \tau_{\text{end}}, \tau_{\text{finish}})$, ESS threshold ρ_{ESS}, elite fraction ρ_{elite}.

Encode the instance once: $E \leftarrow f_\theta(I)$.
Initialise B particles with empty trajectories and feasibility masks.
Choose a set of checkpoint steps $\mathcal{C} \subset \{1, \dots, T\}$ from n_{cp} (e.g., uniformly spaced).
Set global best makespan $C_{\text{best}} \leftarrow +\infty$ and no best trajectory.
for $t = 1, \dots, T$ **do**
    Compute the current temperature τ_t by interpolating between τ_{start} and τ_{end}.
    For all particles in parallel, use the decoder to obtain masked logits over feasible jobs.
    Convert logits to probabilities with a softmax at temperature τ_t.
    Sample one feasible job per particle and update all states, masks and trajectories.
    if $t \in \mathcal{C}$ **then**
        For each particle, sample a completion using the low temperature τ_{finish} and record a proxy makespan.
        If any proxy makespan improves on C_{best}, update the global best trajectory and C_{best}.
        Convert the proxy makespans into normalised importance weights (better particles receive larger weights).
        Compute the effective sample size (ESS) from these weights.
        if $\text{ESS} < \rho_{\text{ESS}} \cdot B$ **then**
            Keep the top $\lfloor \rho_{\text{elite}} B \rfloor$ particles (elites) with the smallest proxy makespans.
            Resample the remaining particles from the full weighted population (with replacement).
            Rebuild particle states, masks and trajectories using the resulting index set.
        end if
    end if
end for
Greedily schedule any remaining operations to obtain final makespans for all particles.
Let $C_{\min}$ and its trajectory be the best final particle.
if $C_{\text{best}} < C_{\min}$ and a harvested trajectory is available **then**
    Replace the best final particle with the harvested trajectory and set $C_{\min} \leftarrow C_{\text{best}}$.
end if
return the best trajectory and its makespan $C_{\min}$.

Different sampling schemes reuse the policy network in various ways. To ensure a fair comparison, we enforce that for a given budget β, the *effective number of policy evaluations* is approximately the same for all methods. Let $\text{bs}_{\text{internal}}$ denote the internal batch size within a method and m the method-specific multiplier that counts how many times each partial trajectory is revisited by the policy:

- **Ordinary sampling:** each trajectory is decoded once. Hence $m = 1$ and the effective number of samples is $N_{\text{eff}} = \text{bs}_{\text{internal}}$.
- **SMC:** the population of particles is revisited at each of n_{cp} resampling checkpoints. Between checkpoints, the decoding proceeds as in ordinary sampling. The effective number of samples is approximated by $N_{\text{eff}} = \text{bs}_{\text{internal}} \cdot n_{\text{cp}}$.

Algorithm 3 Cross–Entropy (CEM) sampling with iterative logit shaping

Require: Instance I, encoder f_θ, decoder g_ϕ, total batch size B, number of rounds R, elite fraction ρ_{elite}, momentum μ, temperature schedule (τ_r), guidance strengths (λ_r).

Encode the instance once: $E \leftarrow f_\theta(I)$.
Initialise a guidance matrix $G \in \mathbb{R}^{T \times N_J}$ with zeros (one row per decoding step, one column per job).
Set global best makespan $C_{\text{best}} \leftarrow +\infty$ and no best trajectory.
for $r = 1, \ldots, R$ **do**
  Choose the batch size for this round, denoted B_r (e.g., a fixed split of B).
  Initialise B_r particles with empty trajectories and feasibility masks.
  for $t = 1, \ldots, T$ **do**
    For all particles in parallel, use the decoder to compute base logits over feasible jobs at step t.
    Take the guidance row G_t and set entries of infeasible jobs to a very negative value.
    Add guidance to the logits: shaped logits = base logits $+\lambda_r G_t$.
    Convert shaped logits to probabilities with a softmax at temperature τ_r.
    Sample one feasible job per particle and update all states, masks and trajectories.
  end for
  Greedily complete any remaining operations and compute final makespans for the B_r trajectories.
  If any trajectory improves on C_{best}, update the stored global best trajectory and C_{best}.
  Select the $\lfloor \rho_{\text{elite}} B_r \rfloor$ elite trajectories with the smallest makespans.
  From these elites, estimate at each step t how often each job j is chosen (empirical frequencies).
  Convert these frequencies into smoothed log-probabilities and store them in a matrix $\hat{G}$.
  Update the guidance matrix by an exponential moving average: $G \leftarrow \mu G + (1 - \mu)\, \hat{G}$.
end for
return the globally best trajectory and its makespan C_{best}.

– **CEM:** decoding is repeated for R rounds, where each round uses updated guidance derived from the elites of the previous rounds. The effective number of samples is $N_{\text{eff}} = \text{bs}_{\text{internal}} \cdot R$.

Given a budget β, we compute

$$\text{bs}_{\text{internal}} = \max\Big(1,\ \big\lfloor \tfrac{\beta}{m} \big\rfloor\Big), \qquad N_{\text{eff}} = \text{bs}_{\text{internal}} \cdot m \le \beta, \tag{20}$$

so that each method uses at most β effective forward calls. The exact values of N_{eff} and $\text{bs}_{\text{internal}}$ are recorded in the metadata, together with the method configuration.

Budgets and hyperparameters. We evaluate all sampling schemes under the budgets $\beta \in \{16, 32, 64, 128, 256, 512, 1024\}$. For SMC and CEM we perform small hyperparameter grids over a few intuitive parameters; the ranges are summarised in Table 3.

CEM is explored over the number of rounds R, the elite fraction, and the momentum used to smooth the guidance updates. The earlier exploratory grid yielded `smc_cfg19` and `cem_cfg05`; these two configurations were frozen before the confirmatory seed-level experiments and were reused unchanged for both neural policies

Table 3: Discrete hyperparameter configurations used in the grid search for SMC and CEM.

Method	Hyperparameter	Values used in grid
SMC	Number of checkpoints n_{cp}	$\{4, 8, 12\}$
	$(\tau_{\text{start}}, \tau_{\text{end}})$	$\{1.5, 2.0\} \times \{0.1, 0.2\}$
	ESS threshold	$\{0.4, 0.6\}$
	Elite fraction	fixed to 0.10
	τ_{finish}	fixed to 0.02 (proxy rollout)
	Mixed precision	`use_amp = True`
CEM	Number of rounds R	$\{2, 3\}$
	Elite fraction	$\{0.10, 0.15\}$
	Momentum μ	$\{0.7, 0.85\}$
	Gumbel–Max	`gumbel_map = True`
	Legacy top-p arguments	recorded but inactive in the frozen Gumbel branch
	Mixed precision	`use_amp = True`
	Initial temperature τ_0	1.1
	Temperature decay	$\tau_{\text{decay}} = 0.99$, $\tau_{\min} = 0.45$
	Legacy prior arguments	$(p_0, p_{\text{decay}}, p_{\min}) = (0.92, 0.995, 0.75)$; inactive
	Smoothing ε	`smooth_eps = 0.001`

and every budget. No configuration was selected or reselected using the confirmatory Taillard observations. For SMC, `smc_cfg19` corresponds to $n_{\text{cp}} = 12$, $\tau_{\text{start}} = 1.5$, $\tau_{\text{end}} = 0.2$, ESS threshold 0.6, elite fraction 0.1, and proxy completion temperature $\tau_{\text{finish}} = 0.02$. For CEM, `cem_cfg05` uses $R = 3$ rounds, elite fraction 0.1, momentum $\mu = 0.85$, Gumbel–Max decoding with initial temperature $\tau_0 = 1.1$ (decayed with factor 0.99 down to $\tau_{\min} = 0.45$) and smoothing constant $\varepsilon = 10^{-3}$(`smooth_eps` $= 0.001$). The configuration record also contains legacy top-p fields $(p_0, p_{\text{decay}}, p_{\min}) = (0.92, 0.995, 0.75)$; these fields are inactive in the frozen Gumbel–Max branch and therefore do not affect either policy's confirmatory results. Figure 8 reports the preceding exploratory sensitivity analysis; it is not used to choose configurations again in the confirmatory experiment.

Instances and repetitions. We conduct experiments on Taillard's benchmark instances (denoted `TA`) for the Job Shop Scheduling Problem. For each instance and each combination of method, budget and hyperparameters, we perform n_{reps} independent repetitions with different random seeds. In all results reported below, we set $n_{\text{reps}} = 10$, but the pipeline is general.

For every repetition, we decode $\text{bs}_{\text{internal}}$ trajectories in parallel and retain the one with the smallest makespan $C_{\min}$. We compute its percentage gap from Eq. (18), using the optimal or best-known makespan $C^\star$ as the reference.

Aggregation across repetitions and instances. Let $g_{i,r}$ denote the best gap (as in (18)) for instance i in repetition r. For each instance, we compute:

$$\text{inst-min-gap}_i = \min_r g_{i,r}, \qquad \text{inst-max-gap}_i = \max_r g_{i,r},$$
$$\text{inst-avg-gap}_i = \tfrac{1}{n_{\text{reps}}} \sum_r g_{i,r}, \qquad \text{inst-std-gap}_i = \mathrm{std}_r(g_{i,r}).$$

Similarly, we record the mean and standard deviation of the wall-clock time per repetition. Aggregated statistics across instances are then obtained by averaging the per-instance quantities. In the figures, solid lines report mean values across instances, while the shaded bands correspond to one standard deviation (for min/max gaps, this is the standard deviation across instances; for average gaps and wall times, this is the root mean square of the per-instance standard deviations). The inference jobs are parallelised over available CUDA devices: three NVIDIA RTX 2080 Ti GPUs and one RTX A6000. The SchedulExpert encoder–decoder is evaluated in *batched mode*: at each decoding step, the logit computation for all particles in the current batch is performed in a single forward pass on the GPU, using mixed precision (`torch.cuda.amp`) when available. The L2D adapter likewise batches policy evaluation, while its published scheduling environment updates are performed on the CPU. We retain the wall-clock time of every seed-level run, but because the completion run uses heterogeneous devices, pooled timing values are descriptive and are not used to support the solution-quality comparisons.

6.3 Experimental results

We now discuss the empirical comparison between ordinary sampling, SMC and CEM using the protocol described in Section 6.2. The confirmatory results use only the frozen configurations `smc_cfg19` and `cem_cfg05`.

Scaling with budget. Figure 5 shows the best minimum gap across repetitions as a function of the budget β for the three schemes. All methods benefit from an increased budget. Under the frozen configurations, ordinary sampling has the lowest mean gap at budgets 16 and 32, whereas SMC becomes best from budget 64 onward. At larger budgets, the resampling and harvesting mechanisms of SMC exploit additional computation more effectively, leading to an increasing paired advantage over ordinary sampling. CEM is consistently worse than SMC and does not show a Holm-significant improvement over ordinary sampling at any evaluated budget.

Figure 6 depicts the corresponding average wall-clock time per repetition for the SchedulExpert runs. SMC and CEM incur overhead from resampling, proxy rollouts and guidance updates. This overhead is most visible at small budgets and is increasingly amortised as ordinary sampling becomes more expensive at larger budgets. These timings describe this implementation and hardware pool; they are not treated as a hardware-normalised comparison. At budgets of at least 64, SMC achieves lower SchedulExpert gaps than ordinary sampling with moderate additional runtime in this implementation.

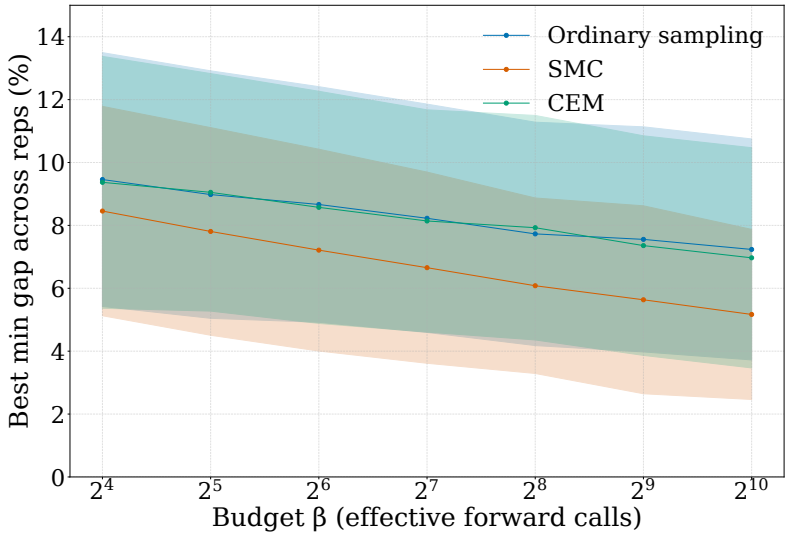
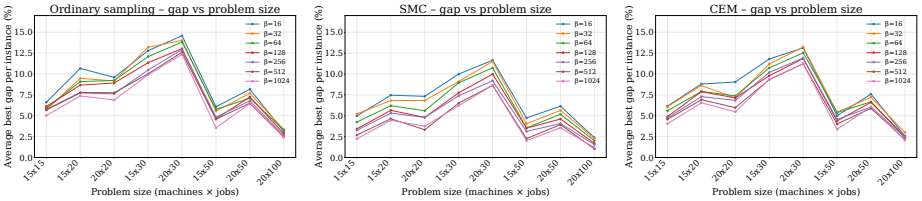


Fig. 5: Best minimum optimality gap across repetitions as a function of the budget β (measured in effective forward calls) for ordinary sampling, SMC and CEM. Solid lines show the mean across instances; shaded regions show one standard deviation.



(a) Ordinary sampling (b) SMC (c) CEM

Fig. 7: Average best optimality gap per instance as a function of problem size (number of jobs $\times$ number of machines) for ordinary sampling, SMC and CEM. Different curves correspond to different sampling budgets β. The confirmatory curves use the frozen configurations `smc_cfg19` and `cem_cfg05`; no configuration is selected again on the Taillard test instances.

Dependence on problem size. To better understand how the sampling schemes behave on instances of different sizes, we group the Taillard instances by their number of machines M and jobs N_J and, for each group, compute the average best gap per instance. Figure 7 reports this statistic for the three methods at the representative budgets used in our study. Each curve corresponds to a fixed budget and the x-axis enumerates the problem sizes (e.g., 15×15, 20×20, 30×20, ...).

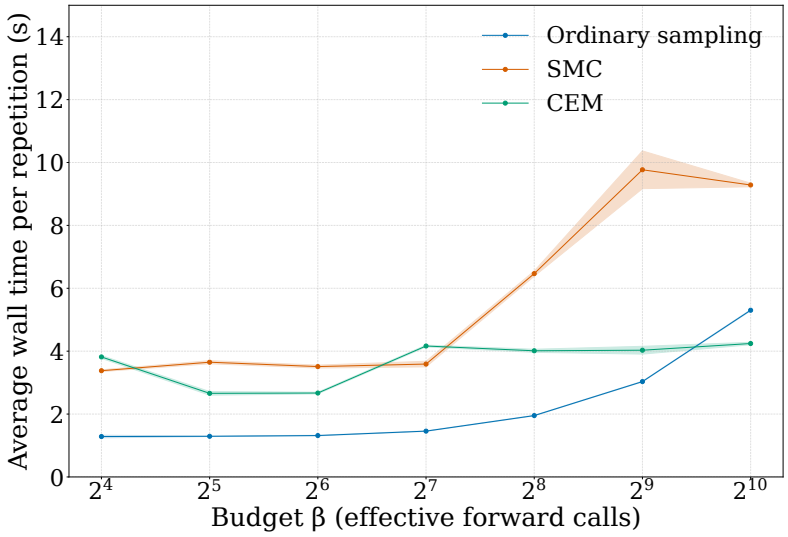


Fig. 6: Average wall-clock time per repetition as a function of the budget β for the three sampling schemes. Times are averaged over instances and repetitions; shaded regions correspond to the variability across instances.

For ordinary sampling, the gap tends to increase with the size of the instance and only decreases again for the largest instances, which are reported to be comparatively easier for these neural baselines, as noted in the literature [41, 40, 9]. The methods display the same qualitative dependence on shape, but the decoder advantage is not uniform over budgets. SMC offers its clearest improvements at budgets of at least 64, including on mid-sized and large instances, indicating that population-based exploration and proxy rollouts become useful when enough particles are available. CEM does not reliably improve on independent sampling under the frozen configuration. These results show that a competitive policy can benefit from a carefully designed inference procedure without retraining, while also demonstrating that portability and improvement must be established empirically for each decoder. At the interface level, the algorithms require only masked logits over feasible operations and do not otherwise depend on the internal MoE architecture.

6.4 Statistical Significance and Sampler Portability

We next evaluate the statistical robustness of the decoder comparisons and whether their behaviour generalises beyond SchedulExpert. The analysis combines paired tests across all 80 Taillard instances with a portability study using the Learning-to-Dispatch (L2D) neural policy [41].

Statistical significance on SchedulExpert. For each budget, we average the ten independent repetitions per instance (yielding 80 paired observations per comparison) and perform a two-sided Wilcoxon signed-rank test. We apply the Holm correction across all 21 paired comparisons (7 budgets $\times$ 3 comparisons). To report the size of the improvement, we compute the median paired percentage-gap improvement and its 95% confidence interval via a paired percentile bootstrap with 10,000 instance-level resamples.

Table 4 reports the results; positive median improvement favours the first method named. SMC is worse than ordinary sampling at budget 16, is not detectably different after correction at budget 32, and improves significantly from budget 64 onward. Its median paired improvement increases from 0.583 percentage points at budget 64 to 1.831 points at budget 1024. CEM is significantly worse than ordinary sampling at budgets 16, 32, 64, and 128, and is not significantly different after Holm correction at budgets 256, 512, or 1024. SMC outperforms CEM at all seven budgets (Holm-adjusted $p \leq 0.0013$).

Table 4: SchedulExpert paired comparisons after averaging the ten seeds within each instance. Tests are two-sided Wilcoxon signed-rank tests over 80 pairs with Holm correction across the 21 tests. Median improvement and its 95% paired-bootstrap confidence interval are in percentage-gap points; positive values favour the first method named.

Budget	Comparison	p	Holm p	Median improvement	95% paired-bootstrap CI	
16	SMC vs ordinary	1.82×10^{-6}	1.64×10^{-5}	-0.567	-0.841	-0.304
16	CEM vs ordinary	1.23×10^{-11}	1.47×10^{-10}	-0.923	-1.144	-0.542
16	SMC vs CEM	0.0002	0.0013	0.363	0.150	0.695
32	SMC vs ordinary	0.0669	0.1719	-0.087	-0.276	0.100
32	CEM vs ordinary	1.19×10^{-9}	1.31×10^{-8}	-0.533	-0.757	-0.430
32	SMC vs CEM	2.60×10^{-6}	2.08×10^{-5}	0.422	0.335	0.526
64	SMC vs ordinary	6.47×10^{-8}	6.47×10^{-7}	0.583	0.419	0.779
64	CEM vs ordinary	0.0004	0.0026	-0.224	-0.472	-0.029
64	SMC vs CEM	1.31×10^{-12}	1.83×10^{-11}	0.796	0.666	0.987
128	SMC vs ordinary	2.43×10^{-12}	3.16×10^{-11}	0.990	0.802	1.126
128	CEM vs ordinary	0.0015	0.0074	-0.210	-0.339	-0.042
128	SMC vs CEM	4.22×10^{-14}	6.32×10^{-13}	1.097	0.916	1.297
256	SMC vs ordinary	1.38×10^{-14}	2.21×10^{-13}	1.304	1.140	1.549
256	CEM vs ordinary	0.0430	0.1719	-0.124	-0.240	0.008
256	SMC vs CEM	1.10×10^{-14}	1.87×10^{-13}	1.481	1.204	1.680
512	SMC vs ordinary	7.85×10^{-15}	1.65×10^{-13}	1.579	1.326	1.724
512	CEM vs ordinary	0.7048	0.7048	0.009	-0.048	0.128
512	SMC vs CEM	8.47×10^{-15}	1.65×10^{-13}	1.509	1.291	1.774
1024	SMC vs ordinary	7.85×10^{-15}	1.65×10^{-13}	1.831	1.440	2.140
1024	CEM vs ordinary	0.0471	0.1719	0.124	-0.006	0.220
1024	SMC vs CEM	8.15×10^{-15}	1.65×10^{-13}	1.749	1.393	2.115

Portability to L2D. We use the frozen 30×20 GraphCNN checkpoint distributed with the authors' official L2D implementation [41]; its SHA-256 digest is recorded in the

accompanying archive. The checkpoint is loaded once in evaluation mode, and its log-probabilities over eligible operations are exposed unchanged to all three decoders, with ineligible operations assigned an exact negative-infinity logit. It is not retrained or fine-tuned separately for SMC or CEM. We reuse `smc_cfg19` and `cem_cfg05` without L2D-specific tuning and run the same 80 Taillard instances, ten seeds, and seven budgets.

Table 5 presents the Wilcoxon test results on L2D, and Table 6 summarises the mean makespan gap and standard deviation for each policy–decoder combination. SMC is significantly worse than ordinary sampling at budgets 16, 32, and 64, is not detectably different after Holm correction at budget 128, and is significantly better at budgets 256, 512, and 1024. Its median paired improvement at those three larger budgets is respectively 1.328, 2.343, and 2.948 percentage-gap points; at budget 1024 its mean gap is 29.906%, compared with 33.096% for ordinary sampling. CEM is significantly worse than ordinary sampling at every budget, while SMC outperforms CEM at every budget. Combining the two policy studies, SMC therefore improves both SchedulExpert and L2D at the common budgets 256, 512, and 1024, although the useful budget threshold is policy-dependent. The frozen CEM configuration improves neither policy.

Table 5: L2D paired comparisons after averaging the ten seeds within each instance. Tests are two-sided Wilcoxon signed-rank tests over 80 pairs with Holm correction across the 21 tests. Median improvement and its 95% paired-bootstrap confidence interval are in percentage-gap points; positive values favour the first method named.

Budget	Comparison	p	Holm p	Median improvement	95% paired-bootstrap CI	
16	SMC vs ordinary	7.85×10^{-15}	1.65×10^{-13}	-3.355	-3.726	-2.953
16	CEM vs ordinary	7.85×10^{-15}	1.65×10^{-13}	-3.863	-4.268	-3.414
16	SMC vs CEM	0.0028	0.0055	0.852	0.111	1.402
32	SMC vs ordinary	5.26×10^{-14}	3.68×10^{-13}	-2.649	-3.360	-2.118
32	CEM vs ordinary	7.85×10^{-15}	1.65×10^{-13}	-3.466	-3.850	-3.242
32	SMC vs CEM	5.57×10^{-6}	2.23×10^{-5}	1.158	0.562	1.606
64	SMC vs ordinary	2.65×10^{-5}	7.95×10^{-5}	-0.690	-0.998	-0.274
64	CEM vs ordinary	9.48×10^{-15}	1.65×10^{-13}	-2.713	-3.124	-2.428
64	SMC vs CEM	6.31×10^{-14}	3.79×10^{-13}	2.148	1.505	2.647
128	SMC vs ordinary	0.1409	0.1409	0.255	-0.154	0.565
128	CEM vs ordinary	7.85×10^{-15}	1.65×10^{-13}	-2.071	-2.247	-1.677
128	SMC vs CEM	1.10×10^{-14}	1.65×10^{-13}	2.167	1.761	2.537
256	SMC vs ordinary	1.72×10^{-12}	8.61×10^{-12}	1.328	1.048	1.677
256	CEM vs ordinary	1.28×10^{-14}	1.65×10^{-13}	-1.691	-1.925	-1.552
256	SMC vs CEM	8.47×10^{-15}	1.65×10^{-13}	2.998	2.783	3.492
512	SMC vs ordinary	2.33×10^{-14}	1.87×10^{-13}	2.343	1.999	2.589
512	CEM vs ordinary	1.94×10^{-14}	1.74×10^{-13}	-1.650	-1.904	-1.480
512	SMC vs CEM	7.85×10^{-15}	1.65×10^{-13}	3.971	3.572	4.190
1024	SMC vs ordinary	7.85×10^{-15}	1.65×10^{-13}	2.948	2.757	3.528
1024	CEM vs ordinary	8.47×10^{-15}	1.65×10^{-13}	-1.602	-1.977	-1.427
1024	SMC vs CEM	7.85×10^{-15}	1.65×10^{-13}	5.085	4.224	5.373

Table 6: Mean gap and standard deviation (%) across all 80 Taillard instances for different policy–decoder combinations.

Budget	Decoder	SchedulExpert	L2D
16	ordinary	11.423 ± 4.102	40.411 ± 10.387
16	SMC	12.011 ± 4.259	43.946 ± 10.354
16	CEM	12.379 ± 4.321	44.652 ± 10.905
32	ordinary	10.783 ± 3.977	38.753 ± 9.957
32	SMC	10.964 ± 3.959	41.414 ± 9.784
32	CEM	11.402 ± 4.187	42.376 ± 10.334
64	ordinary	10.241 ± 3.892	37.309 ± 9.811
64	SMC	9.704 ± 3.767	37.994 ± 9.405
64	CEM	10.509 ± 3.961	40.092 ± 9.978
128	ordinary	9.730 ± 3.786	35.985 ± 9.563
128	SMC	8.814 ± 3.486	35.806 ± 9.153
128	CEM	9.938 ± 3.884	38.149 ± 9.625
256	ordinary	9.281 ± 3.699	34.933 ± 9.331
256	SMC	7.964 ± 3.281	33.586 ± 8.863
256	CEM	9.396 ± 3.760	36.825 ± 9.600
512	ordinary	8.894 ± 3.613	33.944 ± 9.214
512	SMC	7.276 ± 3.120	31.753 ± 8.761
512	CEM	8.887 ± 3.698	35.703 ± 9.232
1024	ordinary	8.500 ± 3.531	33.096 ± 9.032
1024	SMC	6.711 ± 2.964	29.906 ± 8.463
1024	CEM	8.422 ± 3.617	34.794 ± 9.197

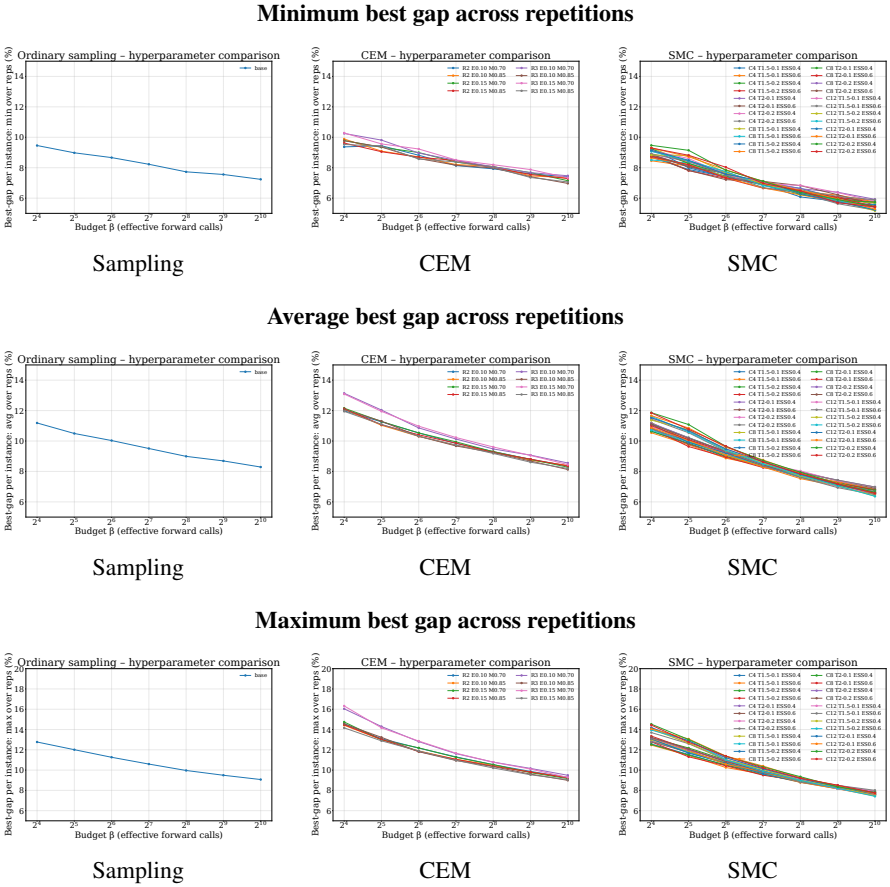


Fig. 8: Hyperparameter sensitivity of the three inference methods across different budgets β. Columns correspond to methods (from left to right): **Sampling**, **CEM**, and **SMC**. Rows correspond to different aggregate statistics of the best gap across repetitions (minimum, average, and maximum). Sampling serves as the *baseline* method (no additional hyperparameters beyond the model). For **SMC**, curves correspond to different settings of the number of checkpoints C, temperature schedule $T_{\text{start}} \rightarrow T_{\text{end}}$, and effective sample size threshold ESS. For **CEM**, curves correspond to different numbers of refinement rounds R, elite fraction E (top proportion of samples used to update the proposal), and momentum parameter M controlling how aggressively the proposal distribution is updated.

7 Conclusion

In this work, we presented SchedulExpert, a neural Mixture-of-Experts model for efficient JSSP solving. By combining graph-based representations, attention-driven expert routing, and self-supervised learning, SchedulExpert achieves strong performance with fewer parameters and reduced reliance on labelled data and is competitive with both classical heuristics and recent deep learning baselines on standard benchmarks.

We further studied how inference-time sampling affects performance by introducing a budget-fair comparison between ordinary Monte Carlo sampling, an SMC sampler with elitist resampling and trajectory harvesting, and a Cross–Entropy Method (CEM) decoder based on iterative logit shaping. On SchedulExpert, the all-instance paired analysis shows that SMC improves significantly over ordinary sampling from budget 64 onward, while it is worse at budget 16 and indistinguishable after Holm correction at budget 32. On the frozen L2D policy, SMC is significantly better at budgets 256–1024, worse at budgets 16–64, and indistinguishable at budget 128. Hence SMC improves both policies at the three largest common budgets without policy retraining, but its useful budget threshold is policy-dependent. The frozen CEM configuration improves neither policy and is significantly worse than ordinary sampling at every L2D budget. Thus, structured decoding can improve solution quality without retraining, but the benefit depends on the decoder, policy, and available budget.

Future work will extend SchedulExpert and these sampling schemes to dynamic JSSP settings and other combinatorial optimisation problems.

A Shape-Level Distribution Summaries

Figures 9–12 show the shape-level distributions for every decoder and budget. Each point summarises 100 seed-level observations: ten instances of the given shape evaluated with ten seeds. Mean plots use shaded bands to show one sample standard deviation, while median plots show the interquartile range. The exact numerical summaries remain available in the accompanying CSV archive. For the paired inferential tests in Section 6.4, the ten repetitions are first averaged within each instance, yielding 80 paired observations per comparison.

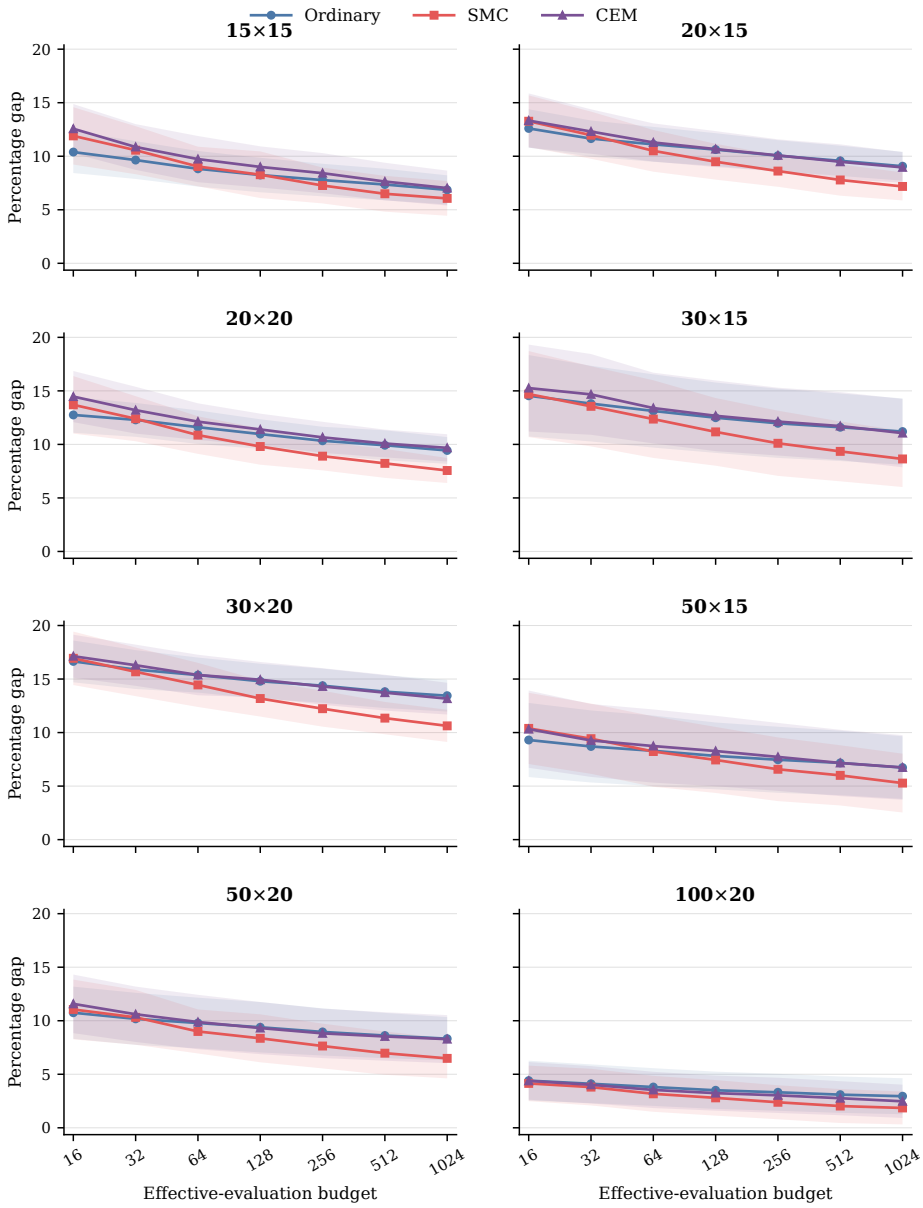


Fig. 9: SchedulExpert mean percentage gap by Taillard shape and effective-evaluation budget. Lines show means across the 100 seed-level observations for each shape, decoder, and budget; shaded bands show one sample standard deviation. Lower values are better.

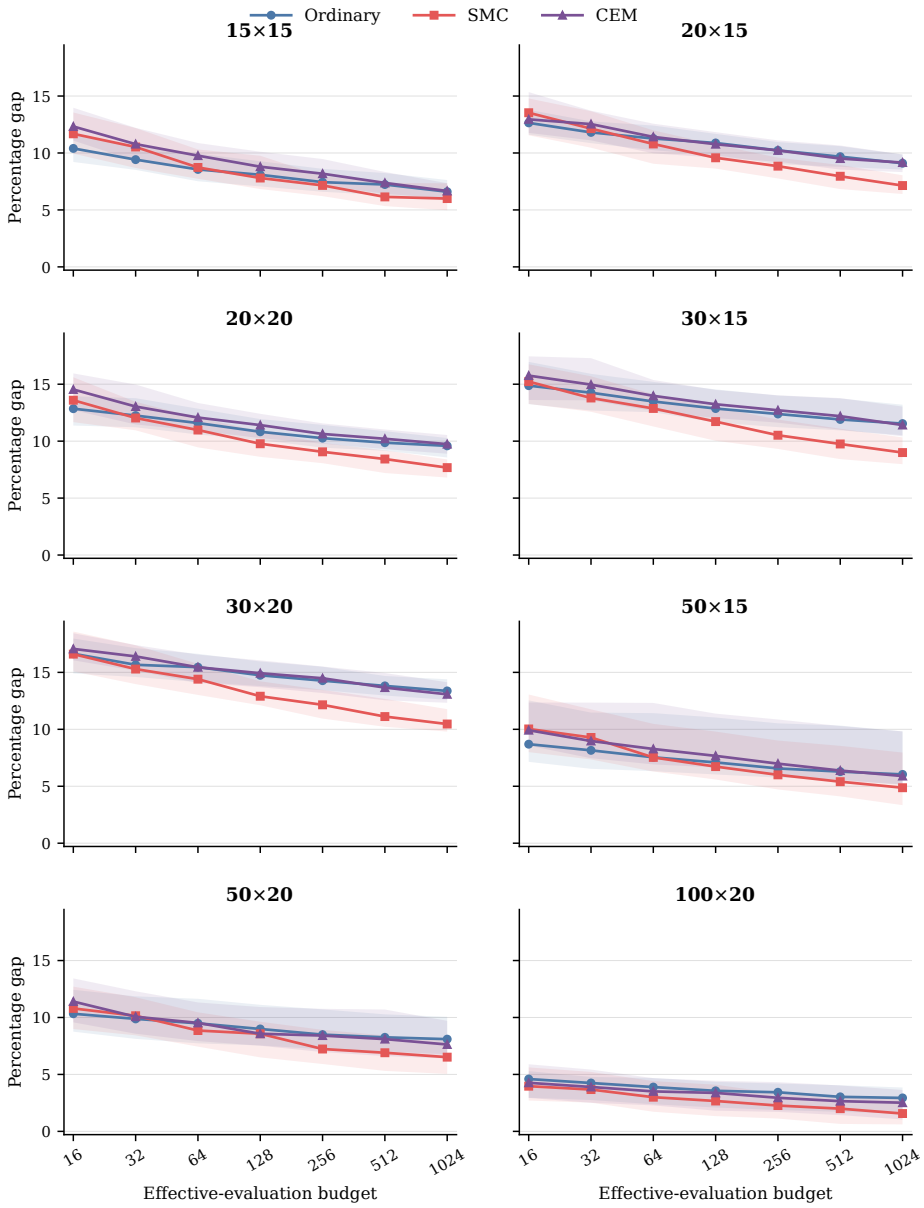


Fig. 10: SchedulExpert median percentage gap by Taillard shape and effective-evaluation budget. Lines show medians across the 100 seed-level observations for each shape, decoder, and budget; shaded bands span the first to third quartiles. Lower values are better.

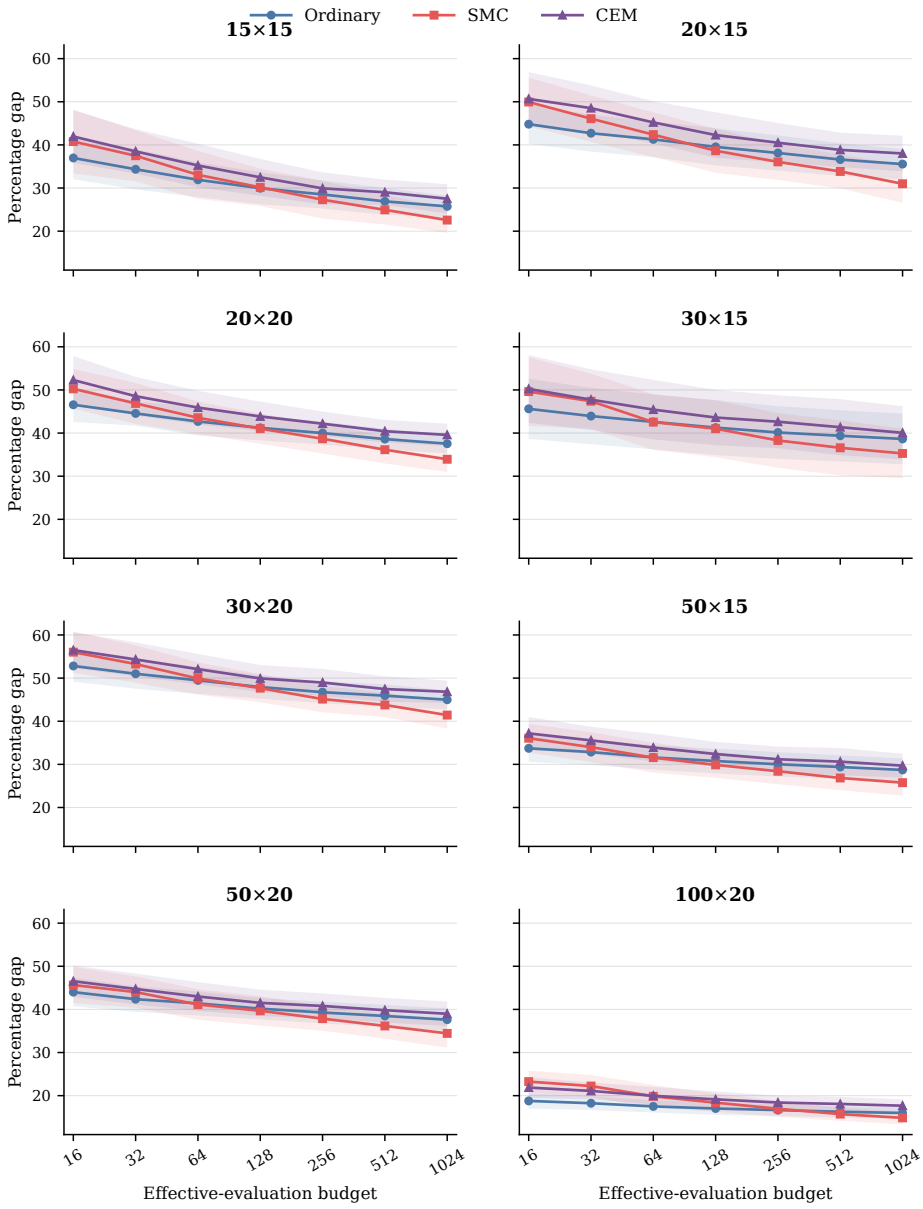


Fig. 11: L2D mean percentage gap by Taillard shape and effective-evaluation budget. Lines show means across the 100 seed-level observations for each shape, decoder, and budget; shaded bands show one sample standard deviation. Lower values are better.

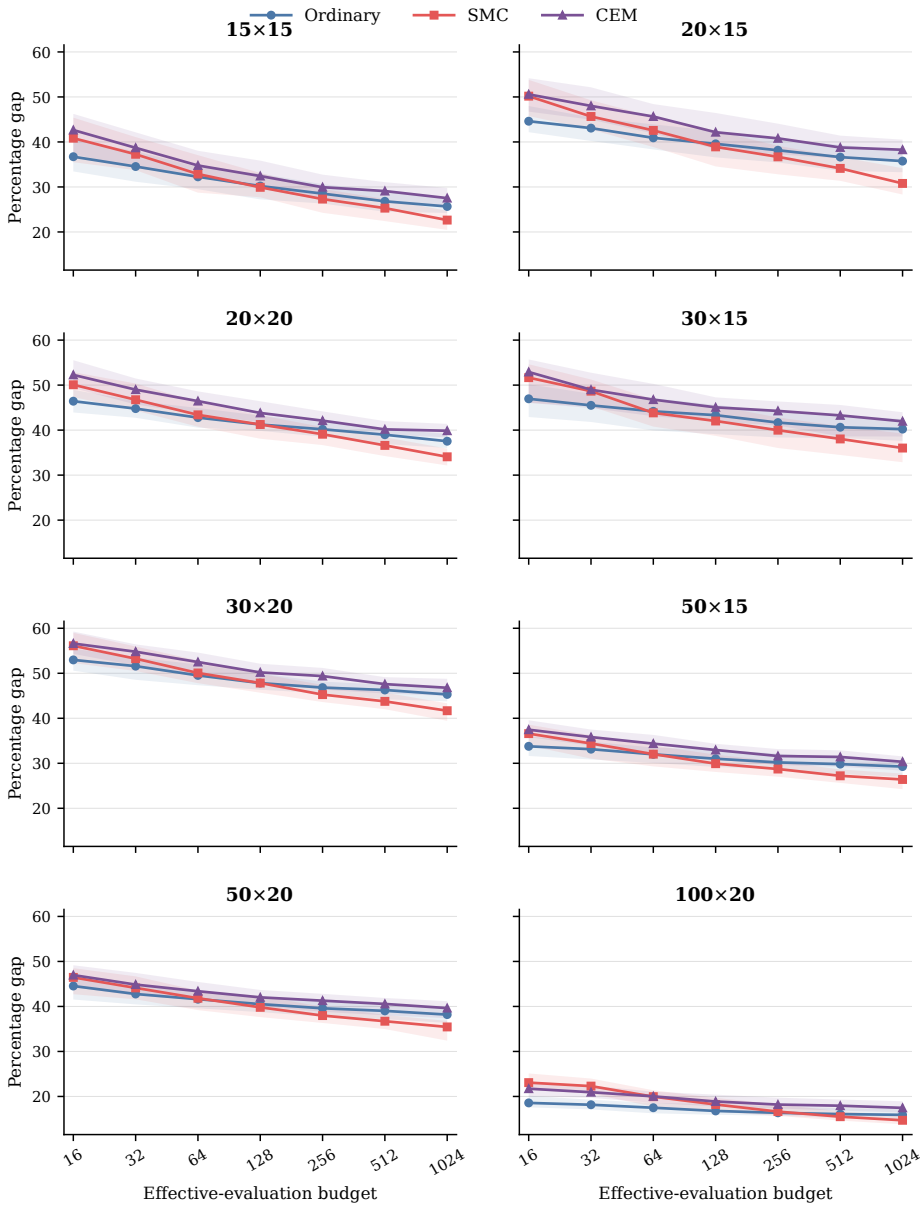


Fig. 12: L2D median percentage gap by Taillard shape and effective-evaluation budget. Lines show medians across the 100 seed-level observations for each shape, decoder, and budget; shaded bands span the first to third quartiles. Lower values are better.

References

1. Abgaryan, H., Harutyunyan, A., Cazenave, T.: Llms can schedule (2024), https://arxiv.org/abs/2408.06993
2. Abgaryan, H., Harutyunyan, A., Cazenave, T.: Randomized greedy sampling for jssp. In: Proceedings of the 18th Learning and Intelligent Optimization Conference (LION 18). Lecture Notes in Computer Science, vol. 14990, pp. 1–12. Springer (2025)
3. Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning (2017)
4. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys **35**(3), 268–308 (2003). https://doi.org/10.1145/937503.937505, https://doi.org/10.1145/937503.937505
5. Bonetta, G., Zago, D., Cancelliere, R., Grosso, A.: Job shop scheduling via deep reinforcement learning: a sequence to sequence approach. In: Learning and Intelligent Optimization. pp. 475–490. Springer International Publishing (2023)
6. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (2012). https://doi.org/10.1109/TCIAIG.2012.2186810
7. Cai, W., Jiang, J., Wang, F., Tang, J., Kim, S., Huang, J.: A survey on mixture of experts. arXiv preprint arXiv:2407.06204 (2024), https://arxiv.org/abs/2407.06204
8. Cazenave, T.: Nested Monte-Carlo search. In: Proceedings of the IJCAI International Joint Conference on Artificial Intelligence. pp. 456–461 (2009)
9. Corsini, A., Porrello, A., Calderara, S., Dell'Amico, M.: Self-labeling the job shop scheduling problem. In: Advances in Neural Information Processing Systems 37 (NeurIPS 2024) (2024), https://arxiv.org/abs/2401.11849
10. Dasagi, V., Bruce, J., Peynot, T., Leitner, J.: Ctrl-z: Recovering from instability in reinforcement learning. CoRR **abs/1910.03732** (2019), http://arxiv.org/abs/1910.03732
11. Doucet, A., de Freitas, N., Gordon, N. (eds.): Sequential Monte Carlo Methods in Practice. Springer (2001)
12. Drakulic, D., Michel, S., Mai, F., Sors, A., Andreoli, J.M.: BQ-NCO: Bisimulation Quotienting for Efficient Neural Combinatorial Optimization. In: Advances in Neural Information Processing Systems 37 (NeurIPS 2023) (2023), https://arxiv.org/abs/2301.03313
13. Ettrich, R., Huber, M., Raidl, G.: A Policy-Based Learning Beam Search for Combinatorial Optimization, pp. 130–145. Springer (03 2023). https://doi.org/10.1007/978-3-031-30035-6_9
14. Falkner, J.K., Thyssens, D., Bdeir, A., Schmidt-Thieme, L.: Learning to control local search for combinatorial optimization. In: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD) (2022), accepted for publication
15. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**(2), 117–129 (1976)
16. Ibrahim, S., Chen, W., Hazimeh, H., Ponomareva, N., Zhao, Z., Mazumder, R.: Comet: Learning cardinality constrained mixture of experts with trees and local search. arXiv preprint arXiv:2306.02824 (2023), https://arxiv.org/abs/2306.02824
17. Iklassov, Z., Medvedev, D., Solozabal Ochoa de Retana, R., Takac, M.: On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) (2023)
18. Ku, W.Y., Beck, J.C.: Mixed integer programming models for job shop scheduling: A computational analysis. Computers & Operations Research **73**, 42–53 (2016)

19. Li, Y., Gu, Q.: Towards understanding the mixture-of-experts layer in deep learning. In: Advances in Neural Information Processing Systems (NeurIPS) (2022), https://arxiv.org/abs/2208.02813
20. Lin, C.C., Deng, D.J., Chih, Y.L., Chiu, H.T.: Smart manufacturing scheduling with edge computing using multiclass deep q network. IEEE Transactions on Industrial Informatics **15**(7), 4276–4284 (2019). https://doi.org/10.1109/TII.2019.2908210
21. Mazyavkina, N., Sviridov, S., Ivanov, S., Burnaev, E.: Reinforcement learning for combinatorial optimization: A survey. Computers & Operations Research **134**, 105400 (2021). https://doi.org/10.1016/j.cor.2021.105400, https://doi.org/10.1016/j.cor.2021.105400
22. Nikishin, E., Izmailov, P., Athiwaratkun, B., Podoprikhin, D., Garipov, T., Shvechikov, P., Vetrov, D., Wilson, A.G.: Improving stability in deep reinforcement learning with weight averaging. In: Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI) Workshop on Uncertainty in Deep Learning (2018), https://www.gatsby.ucl.ac.uk/ balaji/udl-camera-ready/UDL-24.pdf
23. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job shop problem. Management Science **42**(6), 797–813 (1996)
24. Nowicki, E., Smutnicki, C.: An advanced tabu search algorithm for the job shop problem. Journal of Scheduling **8**, 145–159 (2005). https://doi.org/10.1007/s10951-005-6364-5, https://link.springer.com/article/10.1007/s10951-005-6364-5
25. Ouali, Y., Hudelot, C., Tami, M.: An overview of deep semi-supervised learning (2020), https://arxiv.org/abs/2006.05278
26. Park, J., Bakhtiyarov, S., Park, J.: Schedulenet: Learn to solve multiagent scheduling problems with reinforcement learning (2022), preprint
27. Perron, L., Furnon, V.: Or-tools, https://developers.google.com/optimization/
28. Pirnay, J., Grimm, D.G.: Take a step and reconsider: Sequence decoding for self-improved neural combinatorial optimization. In: Proceedings of the 27th European Conference on Artificial Intelligence (ECAI). Frontiers in Artificial Intelligence and Applications, vol. 392, pp. 1927–1934. IOS Press (2024). https://doi.org/10.3233/FAIA240707, https://arxiv.org/abs/2407.17206
29. van Remmerden, J., Bukhsh, Z., Zhang, Y.: Offline reinforcement learning for learning to dispatch for job shop scheduling. Machine Learning **114**(8), 191 (2025). https://doi.org/10.1007/s10994-025-06826-w
30. Rossit, D.A., Tohmé, F., Frutos, M.: The non-permutation flow-shop scheduling problem: A literature review. Omega **77**, 143–153 (2018). https://doi.org/10.1016/j.omega.2017.05.010, https://doi.org/10.1016/j.omega.2017.05.010
31. Rubinstein, R.Y.: The cross-entropy method for combinatorial and continuous optimization. Methodology and Computing in Applied Probability **1**(2), 127–190 (1999)
32. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
33. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993)
34. Wang, L., Hu, X., Wang, Y., Xu, S., Ma, S., Yang, K., Liu, Z., Wang, W.: Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. Computer Networks **190**, 107969 (2021). https://doi.org/10.1016/j.comnet.2021.107969, https://doi.org/10.1016/j.comnet.2021.107969
35. Willi, T., Obando-Ceron, J., Foerster, J., Dziugaite, K., Castro, P.S.: Mixture of experts in a mixture of rl settings. arXiv preprint arXiv:2406.18420 (2024), https://arxiv.org/abs/2406.18420
36. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (ICLR) (2019)

37. Xu, M., Mei, Y., Zhang, F., Zhang, M.: Learn to optimise for job shop scheduling: A survey with comparison between genetic programming and reinforcement learning. Artificial Intelligence Review **58**(6), 160 (2025). https://doi.org/10.1007/s10462-024-11059-9
38. Yuan, E., Cheng, S., Wang, L., Song, S., Wu, F.: Solving job shop scheduling problems via deep reinforcement learning. Applied Soft Computing **143**, 110436 (2023). https://doi.org/https://doi.org/10.1016/j.asoc.2023.110436, https://www.sciencedirect.com/science/article/pii/S1568494623004544
39. Zeiträg, Y., Figueira, J.R.: Automatically evolving preference-based dispatching rules for multi-objective job shop scheduling. Journal of Scheduling **26**, 289–314 (2023). https://doi.org/10.1007/s10951-023-00783-9
40. Zhang, C., Cao, Z., Song, W., Wu, Y., Zhang, J.: Deep reinforcement learning guided improvement heuristic for job shop scheduling. In: International Conference on Learning Representations (ICLR) (2024)
41. Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Xu, C.: Learning to dispatch for job shop scheduling via deep reinforcement learning. In: 34th Conference on Neural Information Processing Systems (NeurIPS) (2020)
42. Zhang, R., Niu, J., Liu, X., Tang, S., Yuan, J.: Learning to optimize job shop scheduling under structural uncertainty. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 40, pp. 36509–36517 (2026). https://doi.org/10.1609/aaai.v40i43.40973