DIFFTP: GRAPH DIFFUSION FOR JOB SHOP SCHEDULING WITH TRANSITIVE PRECEDENCE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph diffusion models have shown strong potential for routing problems, where routes are converted into heatmaps of consecutive node connections and solutions are generated through iterative denoising. However, directly applying graph diffusion to the Job Shop Scheduling Problem (JSSP), by encoding each machine sequence as a heatmap of consecutive operation pairs, does not yield satisfactory results. Our analysis reveals that the key lies in a fundamental difference between routes and schedules: while a route is defined by connections between consecutive nodes, a schedule is characterized by precedence relations that must satisfy **transitivity**. This leads to our key insight: diffusion learning for JSSP should explicitly capture the transitive precedence relations among operations, rather than only their consecutive relations. Based on this insight, we propose DiffTP, a graph Diffusion framework for JSSP with Transitive Precedence modeling. DiffTP consists of two components: (1) a Graph Transformer denoiser that alternates message passing over job and machine relations to model the scheduling constraints in JSSP; and (2) a diffusion learning scheme that uses precedence heatmaps derived from the transitive closure of machine sequences as denoising targets. During inference, DiffTP iteratively denoises the precedence heatmap to predict machine precedence relations, which are then used to guide feasible schedule construction. Experiments on standard JSSP benchmarks demonstrate that DiffTP achieves competitive performance against state-of-the-art constructive baselines.

1 INTRODUCTION

Graph diffusion models have been widely studied for routing problems, particularly the Traveling Salesman Problem (TSP), where they achieve strong solution quality (Sun & Yang, 2023; Li et al., 2023; 2024; Wang et al., 2025). They encode a route as an edge heatmap marking connections between consecutive nodes. During training, the model learns to recover clean edge variables from corrupted inputs; at inference, it starts from random noise and iteratively predicts edge probabilities that guide route construction (Sun & Yang, 2023). This non-autoregressive formulation predicts the edge variables jointly. Despite these advances in routing, adapting graph diffusion to the Job Shop Scheduling Problem (JSSP) remains underexplored, particularly in choosing a denoising target that captures the precedence structure of schedules.

JSSP seeks a minimum-makespan schedule while respecting each job's operation sequence and preventing overlapping operations on the same machine. Common solution approaches include exact optimization, dispatching rules, search heuristics, and learning-based methods (Li et al., 2025; Rihane et al., 2025). Learning-based solvers typically follow either a construction or an improvement paradigm: construction methods extend a partial schedule by selecting operations sequentially (Zhang et al., 2020; Park et al., 2021; Iklassov et al., 2023; van Remmerden et al., 2024), while improvement methods start from a complete schedule and repeatedly revise its operation orders (Zhang et al., 2022). The joint prediction mechanism of graph diffusion suggests a different way to generate these ordering decisions. This motivates our question: *Can we transfer the non-autoregressive graph diffusion paradigm from routing to JSSP, and what should its denoising target represent?*

Our investigation reveals a distinction between route adjacency and scheduling precedence. In symmetric TSP, a tour such as (a, b, c, d, a) marks consecutive unordered pairs as selected edges (Sun &

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Yang, 2023). Adjacency is not transitive: selecting $\{a, b\}$ and $\{b, c\}$ does not imply selecting $\{a, c\}$. In contrast, a machine sequence $o_1 \rightarrow o_2 \rightarrow o_3$ requires o_1 to precede o_3. Consecutive arcs suffice to reconstruct this sequence, but only a transitive target explicitly supervises the nonconsecutive pair (o_1, o_3) (Figure 1). We compare both representations using Transformer models trained for ten epochs on the same 200,000 10×5 instances. Transitive precedence achieves lower average gaps on both independent same-size tests and the LA benchmark (Adams et al., 1988) (Table 1), supporting better solution quality and generalization under the evaluated setup. Appendix A provides the detailed analysis. These results motivate explicit transitive precedence modeling without implying that consecutive targets cannot be learned.

Our key idea is to align the diffusion target with the transitive precedence relation of a schedule. We propose DiffTP, a graph diffusion framework that learns precedence heatmaps derived from the transitive closure of each machine sequence. Its Graph Transformer denoiser alternates message passing over fixed job relations and noisy machine relations, allowing machine-order predictions to incorporate job constraints. Training recovers these clean precedence targets from corrupted heatmaps. During inference, DiffTP starts from random machine-edge states and iteratively refines the precedence probabilities, conditioned on the instance features and fixed job order. A heatmap-guided Giffler–Thompson (GT) decoder (Giffler & Thompson, 1960) then constructs a feasible schedule; among K generated candidates, we select the one with the smallest makespan. The heatmap supplies ordering preferences, while the decoder enforces job precedence and machine non-overlap.

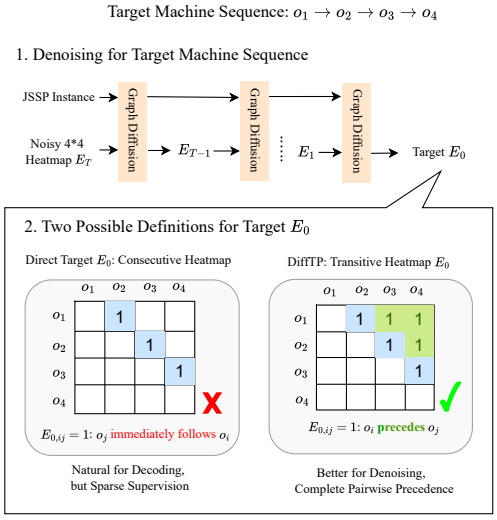

Figure 1: Toy example illustrating transitive precedence.

Table 1: Mean gaps (%) on 50 same-size test instances and LA40 after 10×5 training ($K = 32$, $S = 30$). Protocols appear in Appendix A.

Representation	10×5	LA40
Consecutive relations	2.69	145
Transitive precedence	**0.15**	**19.6**

Our contributions are threefold. **(1)** We identify the distinction between consecutive connections and transitive precedence as a key design consideration for diffusion-based JSSP, and construct machine precedence heatmaps that directly supervise nonconsecutive ordering relations. **(2)** We develop DiffTP, combining job–machine alternating graph denoising with conditional heatmap generation and feasibility-preserving schedule decoding. **(3)** We evaluate DiffTP on standard JSSP benchmarks and analyze target representations, candidate sampling, denoising steps, and model components. DiffTP achieves the lowest average gaps among the evaluated constructive methods on TA and DMU at the reported sampling budgets.

2 Related Work

Constructive solvers build schedules through sequential decisions. L2D (Zhang et al., 2020) combines disjunctive-graph representations with reinforcement learning (RL); ScheduleNet (Park et al., 2021) models job–machine interactions; RASCL (Iklassov et al., 2023) uses curriculum learning; and Offline-LD (van Remmerden et al., 2024) learns from offline scheduling data. SLIM (Corsini et al., 2024) trains on the best sampled schedules, while SIGD (Pirnay & Grimm, 2024) and BOPO (Liao et al., 2025) improve sampling and preference learning, respectively. Self-Evaluation (Echeverria et al., 2025) guides construction by evaluating action subsequences.

Improvement methods revise complete solutions through local search. Tabu search (Zhang et al., 2007) uses memory to guide neighborhood exploration, while L2S (Zhang et al., 2022) learns move

selection through RL. TBGAT (Zhang et al., 2024) uses topology-aware bidirectional graph attention to represent operation dependencies. More broadly, NeuRewriter (Chen & Tian, 2019) learns local rewrites for combinatorial optimization, including online job scheduling.

DIFUSCO (Sun & Yang, 2023) learns edge heatmaps through discrete diffusion (Austin et al., 2021) and decodes them into routes. T2T (Li et al., 2023) adds objective-guided inference, FastT2T (Li et al., 2024) introduces optimization-consistency training, and DEITSP (Wang et al., 2025) improves inference efficiency and generalization. These results motivate diffusion-based scheduling, where the denoising target must capture operation precedence.

A similar work, GOAL (Li, 2026), applies graph diffusion to scheduling by selectively propagating constraint information through heterogeneous edge types, supporting the generation of feasible schedules conditioned on specified objective values. The key distinction lies in the aspect of diffusion learning under study. GOAL emphasizes how scheduling constraints are incorporated into the denoising network, whereas we focus on the denoising target itself as a key determinant of the supervision available to the model. Specifically, DiffTP explicitly supervises transitive precedence relations and examines their effect relative to consecutive-arc targets, showing that the choice of target representation can substantially affect the quality of generated schedules in our experiments.

3 PRELIMINARIES

Job-shop Scheduling. The Job Shop Scheduling Problem (JSSP) is defined by a set of n jobs $\mathcal{J}$ and m machines $\mathcal{M}$. Each job J_i visits every machine once in a prescribed order; operation O_{ij} requires uninterrupted processing on its assigned machine for a duration d_{ij}. A feasible schedule assigns nonnegative start times s_{ij} such that successive operations satisfy $s_{i,j+1} \geq s_{ij} + d_{ij}$ and processing intervals on the same machine do not overlap. The objective is to minimize the makespan $C_{\max} = \max_{i,j}(s_{ij} + d_{ij})$, which is the completion time of the last operation. The instance can be represented as a disjunctive graph $G = (\mathcal{O}, \mathcal{C}, \mathcal{D})$, where nodes $\mathcal{O}$ denote operations, directed conjunctive edges $\mathcal{C}$ encode fixed job precedences, and undirected disjunctive edges $\mathcal{D}$ connect operations competing for the same machine (Błażewicz et al., 2000). Orienting all edges in $\mathcal{D}$ to form a directed acyclic graph (DAG) determines a feasible machine ordering, represented as $G_\pi = (\mathcal{O}, \mathcal{C} \cup \mathcal{A})$, where $\mathcal{A}$ contains the directed machine edges.

Discrete Graph Diffusion. Discrete graph diffusion generates structured solutions by reversing a stochastic corruption process. Following D3PM (Austin et al., 2021) and DIFUSCO (Sun & Yang, 2023), let $\mathbf{E}_0 \in \{0,1\}^N$ represent a clean solution using N binary graph variables. At timestep t, their corrupted states form $\mathbf{E}_t$, whose one-hot encoding is $\mathbf{X}_t = \mathrm{onehot}(\mathbf{E}_t) \in \{0,1\}^{N\times 2}$. The forward process independently corrupts these variables through categorical transitions (Austin et al., 2021; Sun & Yang, 2023):

$$\begin{aligned} q(\mathbf{X}_t \mid \mathbf{X}_{t-1}) &= \mathrm{Cat}(\mathbf{X}_t; \mathbf{X}_{t-1}\mathbf{Q}_t), \\ q(\mathbf{X}_t \mid \mathbf{X}_0) &= \mathrm{Cat}(\mathbf{X}_t; \mathbf{X}_0\overline{\mathbf{Q}}_t), \qquad \overline{\mathbf{Q}}_t = \prod_{r=1}^{t} \mathbf{Q}_r. \end{aligned} \tag{1}$$

Here Cat acts independently on each row, and $\mathbf{Q}_t \in [0,1]^{2\times 2}$ is the transition matrix. For symmetric binary corruption, $\mathbf{Q}_t = (1-2\beta_t)\mathbf{I} + \beta_t \mathbf{1}\mathbf{1}^\top$, where $\beta_t \in [0, \frac{1}{2}]$ is the flip probability and $\mathbf{1} \in \mathbb{R}^2$ is the all-ones vector. The cumulative transition $\overline{\mathbf{Q}}_t$ permits direct sampling from the clean target; a schedule with $\prod_{t=1}^{T}(1-2\beta_t) \approx 0$ yields an approximately uniform terminal distribution. The clean-conditioned reverse posterior follows from Bayes' rule:

$$q(\mathbf{E}_{t-1} \mid \mathbf{E}_t, \mathbf{E}_0) = \frac{q(\mathbf{E}_t \mid \mathbf{E}_{t-1})\, q(\mathbf{E}_{t-1} \mid \mathbf{E}_0)}{q(\mathbf{E}_t \mid \mathbf{E}_0)}. \tag{2}$$

The factors are products of binary transitions determined by $\mathbf{Q}_t$, $\overline{\mathbf{Q}}_{t-1}$, and $\overline{\mathbf{Q}}_t$, with $\overline{\mathbf{Q}}_0 = \mathbf{I}$. Since $\mathbf{E}_0$ is unknown during generation, we use x_0-prediction (Austin et al., 2021): starting from uniform noise $\mathbf{E}_T$, the denoiser predicts a factorized distribution $p_\theta(\mathbf{E}_0 \mid \mathbf{E}_t, t, g)$ conditioned on instance g. Its state-1 probabilities form the estimated clean heatmap $\hat{\mathbf{E}}_0$. To account for uncertainty in this estimate, we average the posterior in Equation 2 over the predicted clean states:

$$p_\theta(\mathbf{E}_{t-1} \mid \mathbf{E}_t, t, g) = \sum_{\mathbf{E}_0 \in \{0,1\}^N} q(\mathbf{E}_{t-1} \mid \mathbf{E}_t, \mathbf{E}_0)\, p_\theta(\mathbf{E}_0 \mid \mathbf{E}_t, t, g). \tag{3}$$

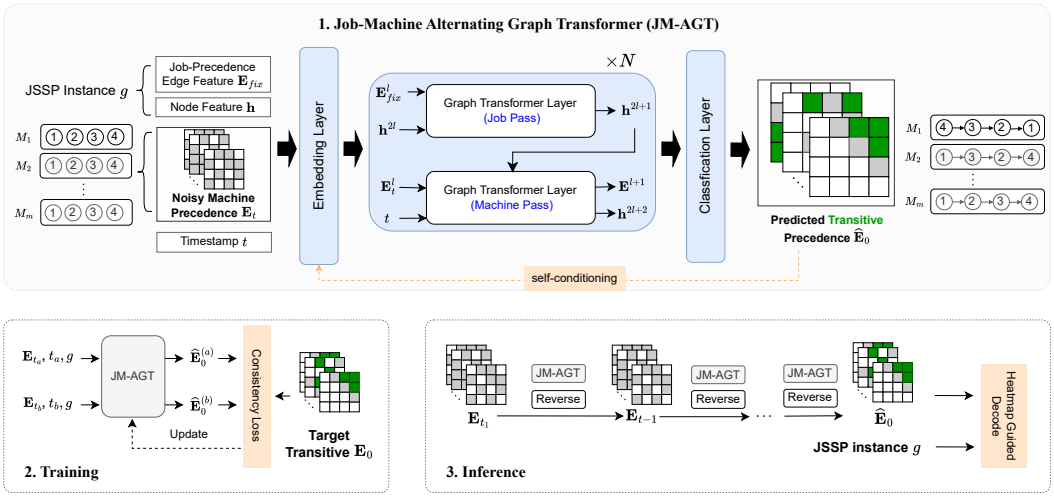

Figure 2: Overview of DiffTP. (1) JM-AGT alternates job and machine updates to predict $\hat{\mathbf{E}}_0$ from fixed features $(\mathbf{h}, \mathbf{E}_{\text{fix}})$, noisy machine precedence $\mathbf{E}_t$, and timestep t. (2) Training supervises two independently corrupted inputs with the same clean target. (3) Inference iteratively denoises the heatmap and decodes a feasible schedule.

Factorization reduces this sum to two states per binary variable and ties the learned transition to the corruption process (Austin et al., 2021). Sampling from Equation 3 gives $\mathbf{E}_{t-1}$ for the next prediction. Iterating to $t = 0$ yields a final heatmap $\hat{\mathbf{E}}_0$ for solution decoding. DDIM-inspired timestep subsampling (Song et al., 2020) accelerates inference using $S < T$ timesteps and the corresponding multi-step categorical posteriors.

4 Proposed Method

4.1 Task Formulation

Given $G = (\mathcal{O}, \mathcal{C}, \mathcal{D})$, DiffTP infers machine precedence $\mathcal{A}$ to construct an acyclic schedule graph $G_\pi = (\mathcal{O}, \mathcal{C} \cup \mathcal{A})$ with minimum makespan $C_{\max}(\pi)$ (Section 3). The operations $\mathcal{O}$ and job constraints $\mathcal{C}$ remain fixed. Figure 2 represents the instance as $\mathcal{H}_t = (\mathbf{h}, \mathbf{E}_{\text{fix}}, \mathbf{E}_t)$: $\mathbf{h}$ contains static operation features from processing times and job/machine statistics; $\mathbf{E}_{\text{fix}}$ equals one on fixed earlier-to-later job pairs; and $\mathbf{E}_t$ contains noisy binary machine precedence variables on the same-machine pairs specified by $\mathcal{D}$. The clean target $\mathbf{E}_0$ records their relative order, including nonconsecutive precedence (Section 4.2).

Conditioned on $g = (\mathbf{h}, \mathbf{E}_{\text{fix}})$ and the connectivity in G, DiffTP learns $p_\theta(\mathbf{E}_0 \mid g)$ from reference schedules by diffusing only machine variables:

$$\mathbf{E}_T \sim \text{Bernoulli}(\tfrac{1}{2})^{|\mathcal{D}|} \xrightarrow[\text{conditioned on } g]{\text{iterative denoising}} \hat{\mathbf{E}}_0 \xrightarrow[\text{subject to } g]{\text{decoding}} \pi. \tag{4}$$

One binary variable is sampled per unordered same-machine pair. The final heatmap $\hat{\mathbf{E}}_0$ supplies precedence probabilities for decoding a feasible schedule under g.

4.2 Transitive Precedence Representation

We now define the fixed job features $\mathbf{E}_{\text{fix}}$ and the clean machine target $\mathbf{E}_0$ introduced in Section 4.1. Both encode transitive precedence relations, so that an edge denotes relative execution order even when the operations are not consecutive.

Transitive closure. For a directed acyclic graph with edge set $\mathcal{E}$, $\text{TC}(\mathcal{E})$ denotes its transitive closure: the set of ordered pairs (i, j) for which a directed path of positive length connects operation

i to operation j using edges in $\mathcal{E}$. Thus, for a feasible schedule graph $G_\pi = (\mathcal{O}, \mathcal{C} \cup \mathcal{A})$, the induced strict partial order is

$$\mathcal{R}_\pi = \mathrm{TC}(\mathcal{C} \cup \mathcal{A}), \qquad i \prec_\pi j \iff (i,j) \in \mathcal{R}_\pi. \tag{5}$$

In particular, $i \prec_\pi j$ and $j \prec_\pi \ell$ imply $i \prec_\pi \ell$. Let $\mathcal{C}_\ell$ contain the consecutive-operation arcs of job J_ℓ, and let $\mathcal{A}_k$ contain those of machine sequence $\pi_k = (\pi_{k,1}, \ldots, \pi_{k,n})$. We separately encode their closures:

$$\begin{aligned} \mathcal{R}_\ell^{\mathrm{job}} &= \mathrm{TC}(\mathcal{C}_\ell) = \{(O_{\ell,r}, O_{\ell,s}) : 1 \le r < s \le m\}, \\ \mathcal{R}_k^{\mathrm{mach}} &= \mathrm{TC}(\mathcal{A}_k) = \{(\pi_{k,r}, \pi_{k,s}) : 1 \le r < s \le n\}. \end{aligned} \tag{6}$$

Job closures define the fixed edge support with $\mathbf{E}_{\mathrm{fix}} = 1$, while machine closures provide the target $\mathcal{A} = \bigcup_k \mathcal{R}_k^{\mathrm{mach}}$ (Figure 2). Relations along paths alternating between jobs and machines belong to $\mathcal{R}_\pi$ but are not stored as additional heatmap entries.

Machine precedence heatmap. For distinct operations i, j assigned to machine k, define

$$E_{0,ij} = \mathbb{I}[(i,j) \in \mathcal{R}_k^{\mathrm{mach}}] = \mathbb{I}[\mathrm{pos}_{\pi_k}(i) < \mathrm{pos}_{\pi_k}(j)], \tag{7}$$

where $\mathrm{pos}_{\pi_k}(i)$ is the position of i in the machine sequence. Each machine has an $n \times n$ heatmap with zero diagonal, complementary entries $E_{0,ji} = 1 - E_{0,ij}$, and transitive orders: $E_{0,ij} = E_{0,j\ell} = 1$ implies $E_{0,i\ell} = 1$. Unlike an edge heatmap encoding consecutive connections in a route (Sun & Yang, 2023), this representation records every earlier-to-later pair in the machine sequence, as illustrated by the heatmaps and output sequences in Figure 2.

The machine heatmaps form $\mathbf{E}_0$. Conditioned on g, the denoiser estimates $\hat{\mathbf{E}}_0$ from $\mathbf{E}_t$. The classification layer enforces complementary directions, transitivity is learned from clean targets, and decoding enforces scheduling feasibility.

4.3 Graph Transformer Denoising Network

The Job-Machine Alternating Graph Transformer (JM-AGT) in Figure 2 adapts DiGress (Vignac et al., 2023) to jointly model operation states and job/machine precedence.

Node, edge, and time embeddings. Each operation has 15 static features adapted from Corsini et al. (2024); each edge has a scalar precedence attribute, $E_{\mathrm{fix},ij} = 1$ or $E_{t,ij}$. We then use sinusoidal encoding (Vaswani et al., 2017) followed by separate linear projections to map both into hidden dimension d_h: the node path is $15 \to 240 \to d_h$ (16 encoded dimensions per feature), and the edge path is $1 \to d_h \to d_h$. The initial states are $\mathbf{h}^0$, $\mathbf{e}_{\mathrm{J}}^0$, and $\mathbf{e}_{\mathrm{M}}^0$; lowercase $\mathbf{e}$ denotes hidden edge features embedded from heatmap values $\mathbf{E}$. The timestep is sinusoidally encoded and projected into an additive signal for each machine layer. Appendix Sections B.1 and B.2 provide the operation feature definitions and encoding details.

Job–machine alternating updates. Let $\mathcal{E}_{\mathrm{J}}$ contain fixed earlier-to-later job pairs and $\mathcal{E}_{\mathrm{M}}$ both directions of same-machine pairs. Each node state $\mathbf{h}_i^l$ and edge state $\mathbf{e}_{\mathrm{J},ij}^l$ or $\mathbf{e}_{\mathrm{M},ij}^l$ is d_h-dimensional. We use self-conditioning (Chen et al., 2023) by adding a projected clean estimate $\mathbf{c}_t$ to the initial machine-edge states:

$$\begin{aligned} \mathbf{e}_{\mathrm{J},ij}^0 &= W_e\,\mathrm{PE}(E_{\mathrm{fix},ij}) + \mathbf{b}_e, && (i,j) \in \mathcal{E}_{\mathrm{J}}, \\ \mathbf{e}_{\mathrm{M},ij}^0 &= W_e\,\mathrm{PE}(E_{t,ij}) + \mathbf{b}_e + W_c\,\mathrm{PE}(c_{t,ij}), && (i,j) \in \mathcal{E}_{\mathrm{M}}. \end{aligned} \tag{8}$$

Here $\mathrm{PE} : \mathbb{R} \to \mathbb{R}^{d_h}$ is the sinusoidal encoding; $W_e, W_c \in \mathbb{R}^{d_h \times d_h}$ and $\mathbf{b}_e \in \mathbb{R}^{d_h}$ are learned parameters. The edge projection is shared across relations, while W_c is separately initialized to zero. Each block updates job relations first, then machine relations:

$$(\mathbf{h}^{2l+1}, \mathbf{e}_{\mathrm{J}}^{l+1}) = \mathcal{T}_{\mathrm{J}}^l(\mathbf{h}^{2l}, \mathbf{e}_{\mathrm{J}}^l; \mathcal{E}_{\mathrm{J}}), \tag{9}$$

$$(\mathbf{h}^{2l+2}, \mathbf{e}_{\mathrm{M}}^{l+1}) = \mathcal{T}_{\mathrm{M}}^l(\mathbf{h}^{2l+1}, \mathbf{e}_{\mathrm{M}}^l; \mathcal{E}_{\mathrm{M}}, t). \tag{10}$$

Here $l = 0, \ldots, N-1$ indexes alternating blocks, each containing one job layer and one timestep-conditioned machine layer. Node states advance twice per block, while each edge stream advances

once. Shared node states exchange job and machine information; Appendix B.3 gives the attention equations.

We write the self-conditioned prediction as $p_\theta(\mathbf{E}_t, t, g; \mathbf{c}_t)$, where $\mathbf{c}_t$ supplies a prior clean estimate through Equation 8.

During training, an auxiliary prediction on the same noisy input supplies the prior estimate:

$$\mathbf{c}_t = \begin{cases} \mathrm{stopgrad}[p_\theta(\mathbf{E}_t, t, g; \mathbf{c}_{\mathrm{neutral}})], & \text{with probability } 0.5, \\ \mathbf{c}_{\mathrm{neutral}}, & \text{otherwise.} \end{cases} \tag{11}$$

Here $\mathbf{c}_{\mathrm{neutral}} = \frac{1}{2}\mathbf{1}$ is an uninformative heatmap, and $\mathrm{stopgrad}$ blocks gradients.

During inference, we initialize $\mathbf{c}_T = \mathbf{c}_{\mathrm{neutral}}$. Each subsequent step directly reuses the clean prediction from the preceding reverse timestep $t_{\mathrm{prev}} > t$:

$$\mathbf{c}_t = \mathrm{stopgrad}\left[\hat{\mathbf{E}}_0^{(t_{\mathrm{prev}})}\right], \qquad \hat{\mathbf{E}}_0^{(t)} = p_\theta(\mathbf{E}_t, t, g; \mathbf{c}_t). \tag{12}$$

The superscript (t) identifies the prediction timestep; inference requires no auxiliary forward pass.

Classification layer. After N blocks, a shared head maps $\mathbf{e}_{\mathrm{M},ij}^N$ to two logits $\mathbf{z}_{ij} = (z_{ij,0}, z_{ij,1})^\top$ for clean states 0 and 1:

$$\begin{aligned} \mathbf{z}_{ij} &= W_{\mathrm{cls}}\,\mathrm{ReLU}\big(\mathrm{LN}(\mathbf{e}_{\mathrm{M},ij}^N)\big) + \mathbf{b}_{\mathrm{cls}}, \\ \ell_{ij} &= z_{ij,1} - z_{ij,0}. \end{aligned} \tag{13}$$

Here LN denotes layer normalization, $W_{\mathrm{cls}} \in \mathbb{R}^{2\times d_h}$ and $\mathbf{b}_{\mathrm{cls}} \in \mathbb{R}^2$ are learned parameters, and ℓ_{ij} scores i before j. Combining opposite directions gives

$$\hat{E}_{0,ij} = \sigma\left(\frac{\ell_{ij} - \ell_{ji}}{2}\right), \qquad \hat{E}_{0,ji} = 1 - \hat{E}_{0,ij}, \tag{14}$$

where σ is the sigmoid function. The probabilities $\hat{E}_{0,ij}$ form the clean precedence heatmap $\hat{\mathbf{E}}_0$ used for reverse updates and final decoding.

4.4 Training and Inference

Training. We adapt the optimization-consistency training of FastT2T (Li et al., 2024): predictions at two noise levels are supervised by the same clean precedence heatmap. Using the directed machine-pair set $\mathcal{E}_{\mathrm{M}}$ defined above, the uniformly averaged cross-entropy is

$$\mathcal{L}_1(\mathbf{E}_0, \mathbf{P}) = -\frac{1}{|\mathcal{E}_{\mathrm{M}}|} \sum_{(i,j)\in\mathcal{E}_{\mathrm{M}}} \big[E_{0,ij}\log P_{ij} + (1 - E_{0,ij})\log(1 - P_{ij})\big]. \tag{15}$$

For t_1 sampled uniformly from $\{1, \ldots, T\}$, set $t_2 = \lfloor 0.5 t_1 \rfloor$ and independently corrupt the same target at these two times. With self-conditioned predictions $\hat{\mathbf{E}}_0^{(t_1)}$ and $\hat{\mathbf{E}}_0^{(t_2)}$, the training loss is

$$\mathcal{L}_2 = \mathcal{L}_1(\mathbf{E}_0, \hat{\mathbf{E}}_0^{(t_1)}) + \mathcal{L}_1(\mathbf{E}_0, \hat{\mathbf{E}}_0^{(t_2)}). \tag{16}$$

Appendix C.1 and Algorithm 1 detail the training procedure.

Inference. We generate K trajectories from independent uniform binary noise and refine each for S reverse steps, using the previous clean prediction as self-conditioning. The final probability heatmap of each trajectory is decoded into a feasible schedule $\pi^{(k)}$, and we return

$$k^* = \arg\min_{1\le k\le K} C_{\max}(\pi^{(k)}), \qquad \pi^* = \pi^{(k^*)}. \tag{17}$$

We accelerate inference using DDIM-inspired timestep subsampling (Song et al., 2020), with the corresponding categorical posterior transitions described in Appendix C.2 and Algorithm 2. The final predicted precedence heatmap $\hat{\mathbf{E}}_0$ guides feasible schedule construction using the Giffler–Thompson (GT) procedure (Giffler & Thompson, 1960). At each step, GT forms a conflict set of eligible operations on a machine, and the heatmap selects the operation with the largest aggregate probability of preceding the others. The selected operation is then scheduled at its earliest feasible start time, thereby preserving job precedence and preventing machine overlap. Appendix D.2 provides the details of the GT decoding procedure.

Table 2: Average gaps (%) of evaluated methods on JSSP benchmarks. "-" indicates unavailable results from the corresponding paper. An asterisk marks problem sizes absent from DiffTP training. The best constructive results are highlighted in bold.

		Non-Constructive				Greedy Constructive								Sampling Constructive							
		Exact Solver	RL-based Improvement			Traditional PDR			RL		SLL		Our	$RL_{K=128}$		$SLL_{K=128}$			$SLL_{K=512}$		$Our_{K=32}$
	Shape	OR-Tools	$L2S_{500}$	TGA_{500}	$L2S_{5k}$	SPT	MOR	MWR	L2D	CL	SLIM	BOPO	DiffTP	L2D	CL	SLIM	BOPO	SLIM	BOPO	DiffTP
TA	15×15	0.1	9.3	8.0	6.2	25.8	20.5	19.2	26.0	14.3	13.8	13.6	4.6	17.1	9.0	7.2	7.1	6.5	6.3	**1.9**
	20×15*	0.2	11.6	9.9	8.3	32.9	23.6	23.4	30.0	16.5	15	14.3	8.3	23.7	10.6	9.3	9.0	8.8	8.3	**5.5**
	20×20*	0.7	12.4	10.0	9.0	27.8	21.7	21.8	31.6	17.3	15.2	15.1	7.9	22.6	10.9	10.0	9.8	9.0	9.1	**5.1**
	30×15*	2.1	14.7	13.3	9.0	35.1	22.7	23.7	33.0	18.5	17.1	16.6	8.7	24.4	14.0	11.0	11.0	10.6	10.3	**6.5**
	30×20	2.8	17.5	16.4	12.6	34.4	24.9	25.2	33.6	21.5	18.5	17.1	11.3	28.4	16.1	13.4	13.3	12.7	12.2	**9.0**
	50×15*	3.0	11.0	9.6	4.6	24.1	17.3	16.8	22.4	12.2	10.1	9.8	6.1	17.1	9.3	5.5	5.8	4.9	4.9	**4.4**
	50×20*	2.8	13.0	11.9	6.5	25.6	17.7	17.9	26.5	13.2	11.6	11.8	8.0	20.4	9.9	8.4	8.0	7.6	7.4	**5.3**
	100×20*	3.9	7.9	6.4	3.0	14.4	9.2	8.3	13.6	5.9	5.8	4.9	5.6	13.3	4.0	2.3	1.8	2.1	**1.4**	3.8
	Avg	2.0	12.2	10.7	7.4	27.5	19.7	19.5	27.1	14.9	13.4	12.9	7.6	20.8	10.4	8.4	8.2	7.8	7.5	**5.2**
DMU	20×15*	1.8	-	-	-	28.0	30.9	28.8	39.0	-	18.0	17.5	13.6	29.3	19.4	12.0	11.2	11.3	10.4	**9.6**
	20×20*	1.9	-	-	-	31.3	27.4	27.3	37.7	-	19.4	20.3	14.5	27.1	16.0	13.5	12.7	12.3	11.8	**8.8**
	30×15*	2.5	-	-	-	31.5	37.4	32.3	42.0	-	21.8	19.1	15.2	34.0	16.5	14.4	13.9	14.0	12.9	**11.3**
	30×20	4.4	-	-	-	34.4	34.7	31.4	39.7	-	25.7	25.6	18.0	33.6	20.2	17.1	16.5	15.8	15.5	**14.0**
	40×15*	4.1	-	-	-	24.0	36.7	27.5	35.6	-	17.5	15.9	13.9	31.5	17.6	11.7	11.4	10.9	10.9	**10.7**
	40×20*	4.6	-	-	-	37.2	37.1	32.9	39.6	-	22.2	22.3	16.7	35.8	25.6	16.0	16.7	14.8	15.9	**14.2**
	50×15*	3.8	-	-	-	24.8	35.5	28.0	36.5	-	15.7	14.5	13.2	32.7	21.7	11.2	11.2	10.6	**10.4**	11.5
	50×20*	4.8	-	-	-	30.1	37.0	30.8	39.5	-	22.4	25.2	17.8	36.1	15.2	15.8	16.5	**15.0**	15.5	15.7
	Avg	3.5	-	-	-	30.2	34.6	29.9	38.7	-	20.3	20.0	15.4	32.5	19.0	14.0	13.8	13.1	12.9	**12.0**

5 EXPERIMENTS

5.1 EXPERIMENTAL SETTINGS

Datasets. We use synthetic datasets containing 1,000,000 instances of size 10×10, 150,000 distinct instances of size 15×15, and 50,000 instances of size 30×20 for model training. Each instance follows the Taillard generation standard (Taillard, 1993), with a random machine permutation for each job and integer processing times sampled uniformly from 1 to 99. Reference schedules are obtained using full-instance constraint programming (CP) or CP-based large neighborhood search (CP-LNS) and converted into precedence heatmaps for supervision. Appendix E describes the CP labeling procedure, dataset construction, and generation budgets. We use 50 held-out instances of each problem size for validation. Evaluation is conducted on the Taillard (TA) (Taillard, 1993) and Demirkol (DMU) (Demirkol et al., 1998) benchmarks at the problem sizes listed in Table 2.

Model Settings. DiffTP uses a 12-layer Graph Transformer adapted from DiGress (Vignac et al., 2023), with hidden dimension 128 and eight attention heads. We train with $T = 1000$ categorical diffusion steps and a cosine noise schedule, using AdamW with learning rate 5×10^{-5} and weight decay 10^{-4}. Checkpoints are selected by validation gap. For TA and DMU, the 15×15 model is used for 15×15, 20×15, and 20×20 instances, and the 30×20 model is used for the remaining sizes. Unless otherwise stated, inference uses $S = 50$ reverse steps and heatmap-guided GT decoding. Table 2 reports both a single-candidate variant ($K = 1$) and a sampling variant ($K = 32$). Training and inference use a single NVIDIA RTX 4090 GPU and seed 12345. Appendix G reports training cost and inference-time scaling with instance size, candidate count, and denoising steps.

Baselines. Table 2 groups the methods into non-constructive, greedy constructive, and sampling-based constructive approaches. The non-constructive group contains OR-Tools and the reinforcement learning (RL) improvement methods L2S (Zhang et al., 2022) and TGA (Zhang et al., 2024), with the search budgets shown in the table. Greedy constructive baselines include the dispatching rules Shortest Processing Time (SPT), Most Operations Remaining (MOR), and Most Work Remaining (MWR); the RL methods L2D (Zhang et al., 2020) and curriculum learning (CL) (Iklassov et al., 2023); and SLIM (Corsini et al., 2024) and BOPO (Liao et al., 2025). For sampling-based construction, L2D and CL use 128 candidates, while SLIM and BOPO use 128 or 512 candidates. DiffTP uses $K = 32$, balancing solution quality and inference cost. Figure 3 shows diminishing gains with larger K; Appendix G reports runtime comparisons and scaling (Tables 13 and 14). These budgets do not imply equal runtime. Dashes indicate unavailable results.

Metric. We evaluate solution quality using the percentage gap relative to the optimal or best-known makespan:

$$\text{Gap} = 100\% \cdot \frac{C_{alg} - C_{ub}}{C_{ub}}, \tag{18}$$

where C_{alg} is the makespan returned by the evaluated method and C_{ub} is the reference makespan for the same instance. For sampling methods, C_{alg} is the smallest makespan among the generated candidates. Table 2 reports the mean of the per-instance gaps for each benchmark and problem size, with "Avg" summarizing each benchmark suite. Lower values indicate better solution quality.

5.2 Main Results

Comparison with greedy constructive methods. With $K = 1$, DiffTP achieves average gaps of 7.6% on TA and 15.4% on DMU, improving over BOPO, the strongest greedy baseline, by 5.3 and 4.6 percentage points. It leads this group on seven of eight TA sizes and all eight DMU sizes. The exception is TA 100×20, where BOPO achieves 4.9% versus DiffTP's 5.6%. Here, $K = 1$ denotes one diffusion trajectory with multiple denoising updates followed by GT decoding.

Comparison with sampling-based methods. Selecting the best of $K = 32$ trajectories reduces DiffTP's average gaps to 5.2% on TA and 12.0% on DMU, improving on its single-candidate variant by 2.4 and 3.4 points. These are the lowest constructive averages in Table 2, compared with 7.5% and 12.9% for BOPO with 512 candidates. DiffTP leads on seven of eight TA sizes and six of eight DMU sizes. The advantage at $K = 1$ shows that the result is not solely due to sampling. Additional candidates improve both suites, although the smaller margin over BOPO on DMU indicates benchmark-dependent gains. Fewer candidates do not imply lower runtime, since per-candidate costs differ across methods.

Performance beyond training sizes. The 30×20 model achieves a 5.3% gap on unseen TA 50×20 instances, compared with BOPO's 7.4% at $K = 512$. However, competing constructive methods remain stronger on TA 100×20 and DMU 50×15 and 50×20. Among non-constructive methods, L2S$_{5k}$ has a higher average TA gap than DiffTP (7.4% versus 5.2%), while OR-Tools retains lower averages on both suites. DiffTP's advantage thus concerns the evaluated constructive methods at the reported budgets.

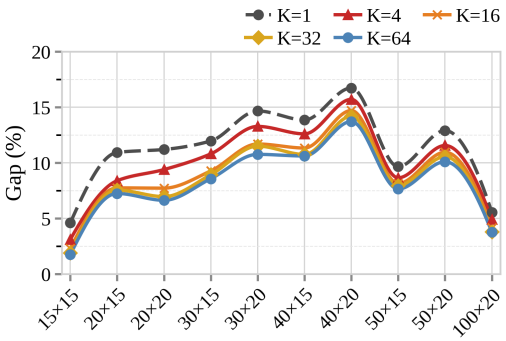

Figure 3: Average makespan gap by problem size for different candidate counts K with $S = 50$.

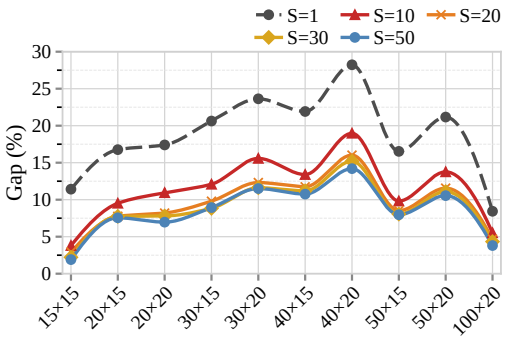

Figure 4: Average makespan gap by problem size for different numbers of denoising steps S with $K = 32$.

5.3 Hyperparameter Analysis

Figures 3 and 4 vary K and S separately, pooling benchmark instances by size.

Number of candidates. With $S = 50$, we evaluate $K \in \{1, 4, 16, 32, 64\}$. Increasing K reduces the gap across the plotted sizes. On 15×15, the gap decreases from 4.6% at $K = 1$ to 1.9% at $K = 32$, with a further reduction to 1.7% at $K = 64$. The corresponding values on 40×20

are 16.7%, 14.2%, and 13.7%. We use $K = 32$, as doubling this budget yields smaller additional gains. These diminishing gains accompany approximately linear runtime growth, as each candidate requires denoising and decoding (Appendix G).

Denoising steps. With $K = 32$, we evaluate $S \in \{1, 10, 20, 30, 50\}$. Iterative denoising improves over one-step prediction. On 15×15, the gap falls from 11.4% at $S = 1$ to 2.2% at $S = 30$, then to 1.9% at $S = 50$. On 40×20, it decreases from 28.2% to 15.2% and 14.2%, respectively. We use $S = 50$ for the additional quality gains, although it requires approximately 67% more sequential denoiser evaluations than $S = 30$.

5.4 Ablation Study

Table 3: Ablations grouped by representation, network components, and decoding. Mean gaps (%) use $K = 32$ and $S = 50$; Ours is the full DiffTP model.

	Representation	Network components		Decoding	Full model
Dataset	w/o TC	AGNN	w/o SC	Non-GT	Ours
TA	24.9	12.4	12.7	5.6	**5.2**
DMU	35.6	20.0	20.8	14.0	**12.0**

Table 3 groups the ablations into three categories, with $K = 32$ and $S = 50$. Appendix F.1 provides size-specific results, single-candidate representation results, and checkpoint-selection details.

(1) Representation. The variant without transitive closure (w/o TC) uses consecutive relations while retaining the model architecture and GT decoder. Its mean gaps reach 24.9% on TA and 35.6% on DMU, compared with 5.2% and 12.0% for DiffTP. This supports transitive precedence modeling within the adopted architecture and decoding scheme; the comparison does not isolate its effect from representation–decoder compatibility.

(2) Network components. Replacing the Graph Transformer with the anisotropic graph neural network (AGNN) used in DIFUSCO (Sun & Yang, 2023) increases the TA and DMU gaps by 7.2 and 8.0 percentage points. Removing self-conditioning (w/o SC) increases them by 7.5 and 8.8 points, supporting both the backbone and prediction reuse.

(3) Decoding. Replacing GT with Borda ranking and feasibility repair (Non-GT; Appendix D.1) increases the gaps by 0.4 points on TA and 2.0 on DMU. GT thus provides an additional improvement beyond the representation and network components. Together, these ablations support the combined design: the benefit of transitive targets depends on both information propagation and heatmap decoding.

6 Conclusions

DiffTP combines transitive precedence heatmaps, job–machine alternating graph denoising, and GT decoding for JSSP. Our experiments show that consecutive targets remain learnable, while transitive representations achieve better same-size performance and LA generalization under the evaluated protocols. On TA and DMU, DiffTP attains the lowest average gaps among the evaluated constructive methods at the reported sampling budgets.

A major limitation is computational cost: dense pairwise modeling increases memory requirements, and repeated denoising becomes expensive for large instances and multiple candidates. Future work will investigate latent diffusion models that generate compact schedule representations, and flow-based generative models that may enable accurate sampling with fewer network evaluations. Preserving precedence information and solution quality will be essential when assessing these alternatives. We also plan to extend the framework to related problems, such as flexible job shop scheduling, by adapting the representation and decoder to additional assignment decisions and resource constraints.

AI USE STATEMENT

We used generative AI tools to assist with drafting and revising manuscript text, literature retrieval, and research ideation and execution. We did not use these tools to generate synthetic datasets or prove mathematical claims. The authors reviewed all AI-assisted content for accuracy, consistency, and originality, including checking research ideas against the literature. AI-generated code was reviewed and tested by two authors. We take responsibility for the final content of this work, including its text, claims, and artifacts.

REFERENCES

Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, pp. 391–401, Mar 1988. doi: 10.1287/mnsc.34.3.391. URL http://dx.doi.org/10.1287/mnsc.34.3.391.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

Jacek Błażewicz, Erwin Pesch, and Małgorzata Sterna. The disjunctive graph machine representation of the job shop scheduling problem. *European Journal of Operational Research*, 127(2): 317–331, 2000.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog Bits: Generating discrete data using diffusion models with self-conditioning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2208.04202.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. *Advances in neural information processing systems*, 32, 2019.

Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. *arXiv preprint arXiv:2401.11849*, 2024.

Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

Imanol Echeverria, Maialen Murua, and Roberto Santana. Self-evaluation for job-shop scheduling. *arXiv preprint arXiv:2502.08684*, 2025.

B. Giffler and G. L. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503, 1960. doi: 10.1287/opre.8.4.487.

Zangir Iklassov, Dmitrii Medvedev, Ruben Solozabal Ochoa De Retana, and Martin Takac. On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In *IJCAI*, pp. 5350–5358, 2023.

Dazi Li, Yanyang Bao, and Xin Xu. Enhancing graph reconstruction: Uniting dual-level graph structure with graph reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 36(10):18218–18228, 2025. doi: 10.1109/TNNLS.2025.3585906.

Xingyu Li. GOAL: Graph-based objective-aligned diffusion solvers for dynamic multi-objective optimization. *arXiv preprint arXiv:2605.19119*, 2026.

Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:50020–50040, 2023.

Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. *Advances in Neural Information Processing Systems*, 37:30179–30206, 2024.

Zijun Liao, Jinbiao Chen, Debing Wang, Zizhen Zhang, and Jiahai Wang. BOPO: Neural combinatorial optimization via best-anchored and objective-guided preference optimization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 37456–37475. PMLR, 2025. URL https://proceedings.mlr.press/v267/liao25a.html.

Junyoung Park, Sanjar Bakhtiyar, and Jinkyoo Park. Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning. *arXiv preprint arXiv:2106.03051*, 2021.

Jonathan Pirnay and Dominik G Grimm. Self-improvement for neural combinatorial optimization: Sample without replacement, but improvement. *arXiv preprint arXiv:2403.15180*, 2024.

Karima Rihane, Adel Dabah, and Abdelhakim AitZai. Learning-based approaches for job shop scheduling problems: A review. *arXiv preprint arXiv:2505.04246*, 2025.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in neural information processing systems*, 36:3706–3731, 2023.

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

Jesse van Remmerden, Zaharah Bukhsh, and Yingqian Zhang. Offline reinforcement learning for learning to dispatch for job shop scheduling. *arXiv preprint arXiv:2409.10589*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. DiGress: Discrete denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=UaAD-Nu86WX.

Mingzhao Wang, You Zhou, Zhiguang Cao, Yubin Xiao, Xuan Wu, Wei Pang, Yuan Jiang, Hui Yang, Peng Zhao, and Yuanshu Li. An efficient diffusion-based non-autoregressive solver for traveling salesman problem. *arXiv preprint arXiv:2501.13767*, 2025.

ChaoYong Zhang, PeiGen Li, ZaiLin Guan, and YunQing Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229–3242, 2007.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in neural information processing systems*, 33:1621–1632, 2020.

Cong Zhang, Zhiguang Cao, Wen Song, Yaoxin Wu, and Jie Zhang. Deep reinforcement learning guided improvement heuristic for job shop scheduling. *arXiv preprint arXiv:2211.10936*, 2022.

Cong Zhang, Zhiguang Cao, Yaoxin Wu, Wen Song, and Jing Sun. Learning topological representations with bidirectional graph attention network for solving job shop scheduling problem. *arXiv preprint arXiv:2402.17606*, 2024.

A Small-Scale Representation Study

A.1 Why transitive precedence?

Experimental setting. We compare four configurations on 10×5 JSSP to distinguish the job message-passing graph from the machine denoising target. A uses consecutive job edges and direct-successor machine labels; B retains consecutive job edges but uses transitive machine labels; C uses transitive job edges and direct-successor machine labels; and D uses transitive relations for both. All configurations use the same alternating job–machine Transformer architecture, self-conditioning, and FastT2T training on the same fixed 200,000 training indices for ten epochs. The forward process uses $T = 1000$ steps with a cosine noise schedule.

Checkpoints are selected solely by mean gap on ten validation instances with $K = 4$ and $S = 30$. The selected models are then evaluated on 50 independent, nonoverlapping test instances with $K = 32$ and $S = 30$. Decoding is matched to the target semantics: A and C use greedy path merging, whereas B and D use Borda ranking; both are followed by feasibility repair (Appendix D.1). We consistently report the original end-of-training evaluations of the selected checkpoints, without mixing in later evaluations using different sampling random streams. The main-text Table 1 combines these same-size test results for A and D with their native-decoder LA averages from Appendix A.2.

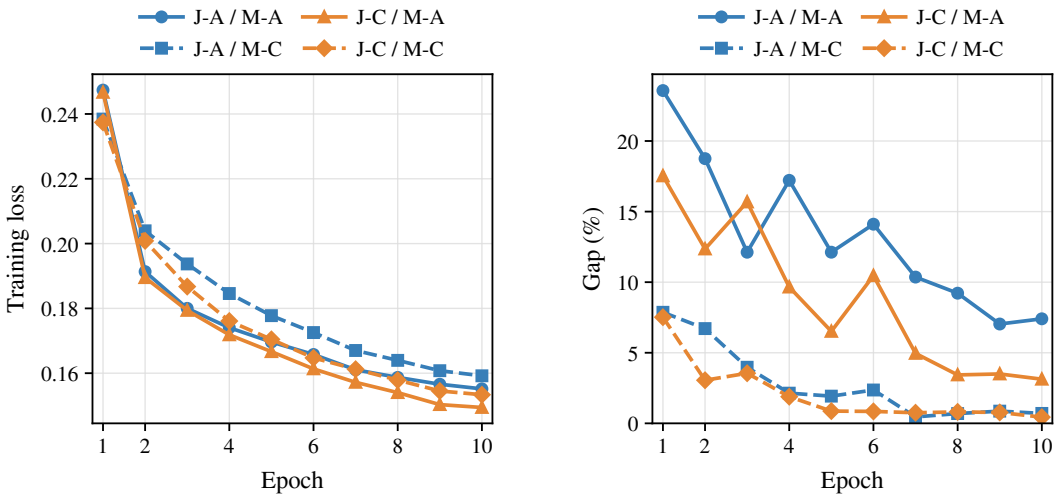

Figure 5: Training objective (left) and validation gap on ten instances (right) for the four representation configurations. In the legend, J/M denotes job/machine and A/C denotes adjacent/closure. Validation uses target-specific decoding with $K = 4$ and $S = 30$. Absolute losses are not directly comparable across targets with different label semantics and positive-label frequencies.

Table 4: Representation study on 10×5 instances. Validation uses ten instances and $K = 4$; independent testing uses 50 instances and $K = 32$, both with $S = 30$. Each ranking procedure is followed by feasibility repair.

	Job graph	Machine target	Ranking	Best epoch	Val. gap (%)	Test gap (%)
A	Consecutive	Direct successor	Path merging	9	7.04	2.69
B	Consecutive	Transitive	Borda	7	0.46	0.49
C	Transitive	Direct successor	Path merging	10	3.13	0.85
D	Transitive	Transitive	Borda	10	**0.44**	**0.15**

Same-size solution quality. With consecutive job edges, transitive machine targets reduce the independent test gap from 2.69% to 0.49% (A versus B). With transitive job edges, the gap decreases from 0.85% to 0.15% (C versus D). Job closure also improves performance for either machine target. The four configurations thus favor transitive machine targets under both job graphs, while the 0.85% gap of C demonstrates that direct-successor targets can also be learned effectively. Figure 5 shows lower validation gaps for transitive machine targets during training, consistent with the independent test results.

These comparisons use recovery procedures suited to each representation, so they assess the representation together with its corresponding decoder rather than an isolated change in supervision. A/B use physical batches of 128, whereas C/D accumulate gradients over two batches of 64; initialization-seed timing also differs across runs. The results support the observed trend under this training protocol without establishing statistical significance across training seeds.

A.2 LA GENERALIZATION AND DECODER COMPARISON

We further evaluate the four selected models on all 40 LA instances, with five instances per size and $K = 32$, $S = 30$. For each model and instance, the native decoder and a common GT decoder (Appendix D.2) receive exactly the same generated heatmaps. Native decoding uses greedy path merging for A/C and Borda ranking for B/D, followed by feasibility repair (Appendix D.1). Table 5 separates the two protocols. The LA40 column of Table 1 reports the native-decoder averages for A and D, using the same recovery procedures as their same-size tests. This comparison changes both job connectivity and machine targets; the four configurations provide the corresponding intermediate comparisons.

Table 5: Mean LA gaps (%) for models trained on 10×5 ($K = 32$, $S = 30$). Native and GT decoding reuse the same heatmaps for each model. A–D are defined in Table 4; Avg pools all 40 instances.

	Native decoder				Common GT decoder			
Size	A	B	C	D	A	B	C	D
10×5	6.20	1.14	2.63	0.00	13.0	1.14	13.0	0.00
15×5	2.88	0.00	0.57	0.00	7.37	0.00	5.32	0.00
20×5	108	0.00	95.4	0.00	6.74	0.00	6.27	0.00
10×10	46.3	24.9	44.9	13.7	15.6	11.0	18.2	9.20
15×10	140	32.5	159	21.9	19.6	15.1	25.4	11.9
20×10	277	36.0	270	28.8	27.5	17.5	27.3	13.1
30×10	327	25.1	317	21.4	15.5	4.51	16.1	3.69
15×15	256	81.1	277	70.9	28.4	20.7	28.5	18.9
Avg	145	25.1	146	19.6	16.7	8.74	17.5	7.11

Generalization beyond the training distribution. With the same recovery procedures used for the 10×5 tests, A and D achieve average LA gaps of 145% and 19.6%, respectively. The advantage of transitive precedence therefore appears in both same-size solution quality (2.69% versus 0.15%) and transfer to the LA benchmark. LA includes five 10×5 instances and 35 instances at unseen sizes, so this aggregate measures benchmark transfer across multiple sizes rather than exclusively cross-size generalization. The four-configuration results reinforce the trend: B improves over A and D improves over C at every reported LA size under native decoding. Direct-successor targets are thus learnable, but their low same-size gaps do not ensure equally strong transfer in this evaluation.

Additional decoding comparison. Under common GT decoding, machine closure reduces the average LA gap from 16.729% to 8.735% with consecutive job edges and from 17.516% to 7.113% with transitive job edges. The improvement holds at every reported size in both comparisons. Job closure alone does not improve the average GT result for direct-successor targets, whereas D improves over B; increasing job connectivity therefore does not substitute for transitive machine modeling in this experiment.

The results favor transitive precedence under the evaluated decoding protocols, while the same-size tests in Appendix A.1 show that direct-successor targets remain learnable. Decoder choice also affects generalization. Replacing native recovery with GT substantially reduces the overall gaps for the same heatmaps, showing that the large native-decoder gaps cannot be attributed entirely to the learned representation.

B Input Features and Network Details

B.1 Operation feature specification

We adapt the 15-dimensional operation features of Corsini et al. (2024), with the feature order and scaling specified below to match our implementation. For operation $O_{\ell,r}$, let $d_{\ell,r} > 0$ be its processing time, $m(\ell, r)$ its assigned machine, and $W_\ell = \sum_{s=1}^{m} d_{\ell,s}$ the total processing time of its job. Define the instance scale $d_{\max} = \max(1, \max_{\ell,r} d_{\ell,r})$ and normalized duration $\bar{d}_{\ell,r} = d_{\ell,r}/d_{\max}$. Machine quartiles retain the additional scaling by $d_{\max}$ used in the implementation. For $\alpha \in \{0.25, 0.50, 0.75\}$, the implemented job and machine statistics are

$$\begin{aligned} Q^{\mathrm{J}}_{\ell,\alpha} &= \mathrm{Quantile}_\alpha\{\bar{d}_{\ell,s} : 1 \le s \le m\}, \\ Q^{\mathrm{M}}_{k,\alpha} &= \mathrm{Quantile}_\alpha\{\bar{d}_{\ell,s}/d_{\max} : m(\ell, s) = k\}. \end{aligned} \tag{19}$$

Table 6 lists the components of $\mathbf{h}_{\ell,r} \in \mathbb{R}^{15}$ in their concatenation order. Each group indexed by α is ordered as $0.25, 0.50, 0.75$.

Table 6: Static operation features used by DiffTP. Here $k = m(\ell, r)$.

Dimensions	Feature group	Definition for $O_{\ell,r}$
1–3	Job quartiles	$Q^{\mathrm{J}}_{\ell,\alpha}$
4–6	Machine quartiles	$Q^{\mathrm{M}}_{k,\alpha}$
7	Normalized processing time	$\bar{d}_{\ell,r}$
8	Cumulative job-work fraction	$\sum_{s=1}^{r} d_{\ell,s}/W_\ell$
9	Remaining job-work fraction (including current operation)	$\sum_{s=r}^{m} d_{\ell,s}/W_\ell$
10–12	Deviation from job quartiles	$\bar{d}_{\ell,r} - Q^{\mathrm{J}}_{\ell,\alpha}$
13–15	Deviation from machine quartiles	$\bar{d}_{\ell,r} - Q^{\mathrm{M}}_{k,\alpha}$

B.2 Input embeddings

For a scalar input $x \in \mathbb{R}$ and an even encoding dimension d, let $\mathrm{PE}(x) \in \mathbb{R}^d$ denote its sinusoidal encoding. Using zero-based component indices $a = 0, \ldots, d/2 - 1$,

$$[\mathrm{PE}(x)]_{2a} = \sin\left(x/10000^{2a/d}\right), \qquad [\mathrm{PE}(x)]_{2a+1} = \cos\left(x/10000^{2a/d}\right). \tag{20}$$

The 15 node features from Table 6 are each encoded with $d = 16$, concatenated into 240 dimensions, and linearly projected to hidden dimension $d_h = 128$. For edge inputs, x is the fixed job attribute $E_{\mathrm{fix},ij} = 1$, the noisy machine attribute $E_{t,ij}$, or the self-conditioning probability $c_{t,ij}$ defined in Section 4.3; each uses PE with output dimension d_h. Job and machine inputs share an edge projection; the self-conditioning projection is separate, has no bias, is initialized to zero, and is added only to machine-edge embeddings.

The diffusion timestep is first encoded as a 128-dimensional sinusoidal vector and mapped to a 64-dimensional time embedding by a shared multilayer perceptron. Each machine layer projects this embedding back to 128 dimensions to match its edge features. After the layer update, the projected time embedding is added elementwise to every machine-edge feature of the instance, leaving the feature dimension unchanged. Job-edge features receive no time embedding.

B.3 Detailed attention updates

Job and machine layers use the same attention and feature-update rules, with separate learned parameters and their respective neighborhoods. Here l denotes the current network layer, and $h^l_i, e^l_{ij} \in \mathbb{R}^{d_h}$ are its input node and edge states. The neighborhood $\mathcal{N}_i$ contains operations whose messages are gathered at operation i through the active job or machine edges. Thus, for an edge pair (i, j), the message flows from j to i. Only the active edge type is updated in each layer.

For job and egde embeddings, the layer normalizations produce $\bar{h}_i = \mathrm{LN}_h(h^l_i)$ and $\bar{e}_{ij} = \mathrm{LN}_e(e^l_{ij})$. The query, key, and value vectors are

$$(q_i, k_i, v_i) = (W_Q \bar{h}_i, W_K \bar{h}_i, W_V \bar{h}_i). \tag{21}$$

Here $W_Q, W_K, W_V \in \mathbb{R}^{d_a \times d_h}$ are bias-free projections, with head dimension $d_a = d_h/H = 16$ for $H = 8$ heads. Two learnable affine maps $\phi_s, \phi_b : \mathbb{R}^{d_h} \to \mathbb{R}$ turn the edge state into a scale $\rho_{ij} = 1 + \tanh(\phi_s(\bar{e}_{ij}))$ and a shift $\eta_{ij} = \phi_b(\bar{e}_{ij})$. The attention logit b_{ij} and normalized weight α_{ij} are

$$b_{ij} = \rho_{ij} \frac{\langle q_i, k_j \rangle}{\sqrt{d_a}} + \eta_{ij}, \tag{22}$$

$$\alpha_{ij} = \frac{\exp(b_{ij})}{\sum_{j' \in \mathcal{N}_i} \exp(b_{ij'})}. \tag{23}$$

Each message combines the sender's value with the edge feature:

$$m_i = \sum_{j \in \mathcal{N}_i} \alpha_{ij}(v_j + W_E \bar{e}_{ij}), \tag{24}$$

where $W_E \in \mathbb{R}^{d_a \times d_h}$ is a bias-free projection and $m_i \in \mathbb{R}^{d_a}$. An empty neighborhood gives a zero message. Let $M_i \in \mathbb{R}^{d_h}$ concatenate these messages across all heads, and let $p_{ij} \in \mathbb{R}^{d_h}$ concatenate their endpoint interactions $q_i \odot k_j$, where $\odot$ denotes elementwise multiplication. The shared update form is

$$h_i^{l+1} = \mathcal{U}_h(h_i^l, M_i), \tag{25}$$

$$e_{ij}^{l+1} = \mathcal{U}_e(e_{ij}^l, \bar{h}_i, \bar{h}_j, p_{ij}). \tag{26}$$

The node update $\mathcal{U}_h$ adds an affine projection of M_i to the input node state, followed by a pre-normalized residual feed-forward sublayer. The edge update $\mathcal{U}_e$ forms $V_r \bar{h}_i + V_s \bar{h}_j + V_p p_{ij} + \bar{e}_{ij}$ using bias-free matrices $V_r, V_s, V_p \in \mathbb{R}^{d_h \times d_h}$, applies a SiLU–affine transform, and adds the result to the input edge state. A separate pre-normalized residual feed-forward sublayer then updates the edge state. Both feed-forward maps use dimensions $d_h \to 4d_h \to d_h$ with GELU activation, and both node and edge increments are computed from the same pre-update normalized states.

Machine layers additionally add the projected time embedding from Section B.2 to each updated edge state. Job layers apply the above updates without this addition. The message-passing mechanism is otherwise identical for the two relation types.

C Supplementary Diffusion Formulation

This section supplements the categorical diffusion definitions in Section 3 with the training and inference details used by DiffTP.

C.1 Forward corruption and training

For implementation, each full machine heatmap is stored using one entry per unordered pair; the reverse entry is its complement. Fix a target-independent enumeration $(i_{k,1}, \ldots, i_{k,n})$ on machine k and define $\mathcal{P} = \bigcup_{k=1}^{m} \{(i_{k,r}, i_{k,s}) : r < s\}$. The compact state used in the diffusion equations is

$$\mathbf{E}_0 = (E_{0,ij})_{(i,j) \in \mathcal{P}} \in \{0,1\}^{|\mathcal{P}|}, \qquad |\mathcal{P}| = |\mathcal{D}| = m\binom{n}{2}. \tag{27}$$

This vector and the full heatmaps describe the same pairwise orders. Noisy reverse entries are reconstructed by complementation, and the classification layer produces complementary probabilities. Averaging binary cross-entropy over $\mathcal{P}$ is equivalent to averaging over both directions in $\mathcal{E}_{\mathrm{M}}$ as in Equation 15, since the losses of complementary entries are equal.

Only canonical machine pairs are corrupted. Using the categorical forward process in Equation 1, its binary marginal can be written as

$$q(E_{t,uv} = 1 \mid E_{0,uv}) = \bar{\alpha}_t E_{0,uv} + (1 - \bar{\alpha}_t)/2, \tag{28}$$

where $\bar{\alpha}_t$ follows a cosine schedule, $\bar{\alpha}_0 = 1$, and $\bar{\alpha}_T \approx 0$. Thus, the terminal distribution is approximately uniform over binary pair states. We use $T = 1000$. Each training iteration samples a batch of labeled instances, constructs their canonical heatmaps, and samples the two corrupted states for Equation 16. Self-conditioning is independently enabled for the two noise levels, and AdamW updates the network using the batch-averaged loss. Algorithm 1 gives the procedure. The job relations and instance features remain unchanged throughout corruption.

Algorithm 1 DiffTP training

Require: Labeled instances $\mathcal{X}$, diffusion horizon T, network p_θ
while training is not complete **do**
  Sample a batch $\mathcal{B} \subset \mathcal{X}$; initialize $\mathcal{L}_\mathcal{B} \leftarrow 0$
  For each $(g, \pi) \in \mathcal{B}$, construct $\mathbf{E}_0$ using Equation 7
  Sample $t_1 \sim \mathrm{Uniform}\{1, \dots, T\}$ per instance; set $t_2 = \lfloor 0.5 t_1 \rfloor$
  for $r = 1, 2$ **do**
    Independently sample $\mathbf{E}_{t_r} \sim q(\cdot \mid \mathbf{E}_0)$ using Equation 28
    Initialize $\mathbf{c}_r \leftarrow \frac{1}{2}\mathbf{1}$
    if a Bernoulli(0.5) draw for this batch is 1 **then**
      $\mathbf{c}_r \leftarrow \mathrm{stopgrad}[p_\theta(\mathbf{E}_{t_r}, t_r, g; \mathbf{c}_r)]$
    end if
    $\mathbf{P}_{t_r} \leftarrow p_\theta(\mathbf{E}_{t_r}, t_r, g; \mathbf{c}_r)$
    $\mathcal{L}_\mathcal{B} \leftarrow \mathcal{L}_\mathcal{B} + |\mathcal{B}|^{-1} \sum_{g \in \mathcal{B}} \mathcal{L}_1(\mathbf{E}_0, \mathbf{P}_{t_r})$
  end for
  Update θ with AdamW using $\nabla_\theta \mathcal{L}_\mathcal{B}$
end while

C.2 DDIM-INSPIRED REVERSE INFERENCE

DDIM (Song et al., 2020) motivates reducing the number of denoising evaluations by selecting a subset of training timesteps. Following the discrete formulation discussed in Appendix G of DIFUSCO (Sun & Yang, 2023), we retain the categorical forward marginals and evaluate the reverse posterior directly between selected times.

Inference procedure. We generate K trajectories with independent initial states $\mathbf{E}_T^{(k)} \sim \mathrm{Bernoulli}(\frac{1}{2})^{|\mathcal{P}|}$. Let $1 \le S \le T$ be the number of reverse evaluations ($S = M$ in the preliminary notation), with $T = \tau_S > \cdots > \tau_1 \ge 1$ and $\tau_0 = 0$. For $S > 1$, the positive times are obtained by rounding S equally spaced values from 1 to T; for $S = 1$, the network is evaluated at T. At step i, the model predicts $\hat{\mathbf{E}}_0^{(k)} = p_\theta(\mathbf{E}_{\tau_i}^{(k)}, \tau_i, g; \mathbf{c}^{(k)})$ and updates the self-conditioning input. Following Equations 2 and 3, each pair uses the categorical posterior mixture

$$p_\theta(E_{\tau_{i-1},uv} \mid \mathbf{E}_{\tau_i}, g; \mathbf{c}) = \sum_{E_{0,uv} \in \{0,1\}} q(E_{\tau_{i-1},uv} \mid E_{\tau_i,uv}, E_{0,uv}) p_\theta(E_{0,uv} \mid \mathbf{E}_{\tau_i}, \tau_i, g; \mathbf{c}). \tag{29}$$

Intermediate pair states are sampled from the posterior mixture, whose skipped-timestep formulation is derived below. At the final step, the clean probabilities are retained for GT decoding, yielding one schedule per trajectory. The candidate with the smallest makespan is selected using Equation 17. Algorithm 2 summarizes this procedure.

Binary forward marginals. Let $\mathbf{U} = \frac{1}{2}\mathbf{1}\mathbf{1}^\top$ denote the uniform transition matrix on $\{0, 1\}$, where $\mathbf{1} \in \mathbb{R}^2$ is the all-ones vector and $\mathbf{I}$ below is the 2×2 identity matrix. Since $\mathbf{U}^2 = \mathbf{U}$, the symmetric transitions in Section 3 have the cumulative form

$$\overline{\mathbf{Q}}_t = \bar{\alpha}_t \mathbf{I} + (1 - \bar{\alpha}_t)\mathbf{U} = \frac{1}{2}\begin{pmatrix} 1 + \bar{\alpha}_t & 1 - \bar{\alpha}_t \\ 1 - \bar{\alpha}_t & 1 + \bar{\alpha}_t \end{pmatrix}, \qquad \bar{\alpha}_t = \prod_{r=1}^{t} (1 - 2\beta_r). \tag{30}$$

The state-1 entry of row $E_{0,uv}$ gives Equation 28. In the implementation, the retention coefficients are specified directly by the cosine schedule

$$\bar{\alpha}_t = \max\left\{ \frac{f(t)}{f(0)}, 10^{-8} \right\}, \qquad f(t) = \cos^2\left(\frac{t/T + 0.008}{1.008} \frac{\pi}{2} \right). \tag{31}$$

This choice induces $\beta_t = (1 - \bar{\alpha}_t / \bar{\alpha}_{t-1})/2$ and keeps the cumulative matrices invertible. The cosine schedule controls corruption strength; the inference timesteps themselves are linearly spaced as specified above.

Algorithm 2 DiffTP sampling and schedule construction

Require: Instance g, trained p_θ, K candidates, times $\tau_S > \cdots > \tau_1 \geq 1$, $\tau_0 = 0$
1: Initialize independent $\mathbf{E}^{(k)}_{\tau_S} \sim \text{Bernoulli}(\frac{1}{2})^{|\mathcal{P}|}$ and $\mathbf{c}^{(k)} \leftarrow \frac{1}{2}\mathbf{1}$
2: **for** $i = S, S-1, \ldots, 1$ **do**
3: For all k, predict $\hat{\mathbf{E}}^{(k)}_0 \leftarrow p_\theta(\mathbf{E}^{(k)}_{\tau_i}, \tau_i, g; \mathbf{c}^{(k)})$
4: Update $\mathbf{c}^{(k)} \leftarrow \hat{\mathbf{E}}^{(k)}_0$
5: **if** $i > 1$ **then**
6: Sample each canonical pair of $\mathbf{E}^{(k)}_{\tau_{i-1}}$ using Equation 29
7: **end if**
8: **end for**
9: Expand final probabilities with $\hat{E}^{(k)}_{0,vu} = 1 - \hat{E}^{(k)}_{0,uv}$
10: Decode $\pi^{(k)} \leftarrow \text{GT}(g, \hat{\mathbf{E}}^{(k)}_0)$ using Algorithm 3
11: $k^* \leftarrow \arg\min_k C_{\max}(\pi^{(k)})$
12: **return** $\pi^{(k^*)}$

Posterior between selected timesteps. For two consecutive evaluation times $s = \tau_{i-1} < t = \tau_i$, marginalizing the omitted forward states gives

$$\mathbf{Q}_{s,t} = \prod_{r=s+1}^{t} \mathbf{Q}_r = \overline{\mathbf{Q}}_s^{-1}\overline{\mathbf{Q}}_t = \frac{\bar{\alpha}_t}{\bar{\alpha}_s}\mathbf{I} + \left(1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_s}\right)\mathbf{U}. \tag{32}$$

For a canonical pair $(u, v) \in \mathcal{P}$ and states $a, b, c \in \{0, 1\}$, Bayes' rule then yields

$$q(E_{s,uv} = b \mid E_{t,uv} = c, E_{0,uv} = a) = \frac{[\overline{\mathbf{Q}}_s]_{ab}[\mathbf{Q}_{s,t}]_{bc}}{[\overline{\mathbf{Q}}_t]_{ac}}. \tag{33}$$

The denominator normalizes the two possible states at time s, because $\overline{\mathbf{Q}}_s\mathbf{Q}_{s,t} = \overline{\mathbf{Q}}_t$. This expression reduces to the one-step posterior when $s = t - 1$ and applies without evaluating the denoiser at any omitted timestep.

Learned reverse update. Omitting the trajectory index k, write $\hat{p}_{t,uv} = p_\theta(E_{0,uv} = 1 \mid \mathbf{E}_t, t, g; \mathbf{c}_t)$ for the (u, v) entry of the predicted clean heatmap $\hat{\mathbf{E}}_0$. Averaging the posterior over its two clean states gives the probability used by the sampler:

$$r_{s|t,uv} = \sum_{a=0}^{1} (\hat{p}_{t,uv})^a (1 - \hat{p}_{t,uv})^{1-a} \frac{[\overline{\mathbf{Q}}_s]_{a1}[\mathbf{Q}_{s,t}]_{1,E_{t,uv}}}{[\overline{\mathbf{Q}}_t]_{a,E_{t,uv}}}. \tag{34}$$

For $s > 0$, the next state and self-conditioning input are updated as

$$E_{s,uv} \sim \text{Bernoulli}(r_{s|t,uv}), \qquad \mathbf{c}_s = \hat{\mathbf{p}}_t. \tag{35}$$

Self-conditioning starts from $\mathbf{c}_T = \frac{1}{2}\mathbf{1}$ and supplies the previous clean prediction to the next evaluation. At $s = 0$, $\overline{\mathbf{Q}}_0 = \mathbf{I}$ implies $r_{0|t,uv} = \hat{p}_{t,uv}$; we therefore retain the final clean probabilities for GT decoding without a terminal binary sample. This requires S denoiser evaluations per trajectory instead of T, while using the same trained network.

Only canonical pairs are sampled: the opposite direction is reconstructed as $E_{t,vu} = 1 - E_{t,uv}$, and predicted heatmap entries satisfy $\hat{p}_{t,vu} = 1 - \hat{p}_{t,uv}$. Complementation preserves mutually exclusive pairwise orders but does not enforce transitivity or global acyclicity, which motivates the feasible decoder in Section D.2. Self-conditioning carries predictions between reverse evaluations; the shared-target self-consistency objective in Equation 16 instead supervises training at two noise levels and introduces no additional inference update.

D Schedule Decoding and Recovery

D.1 Order-then-repair decoding

We refer to the two-stage recovery procedure as *order-then-repair decoding*. First, the predicted heatmap is converted into a complete operation sequence for each machine, using a ranking or path-construction rule appropriate to its edge semantics. Second, these machine sequences are scheduled

jointly with the fixed job precedences, repairing global conflicts when necessary. The first stage differs between consecutive and transitive representations; the second follows the same deadlock-repair principle.

Greedy path merging for consecutive arcs. The consecutive-arc target assigns a positive label only to immediate neighbors in a machine sequence. For a reference sequence π_k, its entries are

$$A^{\text{con}}_{uv} = \mathbb{I}\big[\text{pos}_{\pi_k}(v) = \text{pos}_{\pi_k}(u) + 1\big], \qquad u, v \in \mathcal{O}_k, \quad u \neq v. \tag{36}$$

For this consecutive representation, we adapt the heatmap-guided greedy edge-merging design used for TSP decoding in DIFUSCO (Sun & Yang, 2023). We call this procedure *greedy path merging*: candidate arcs are considered in descending score order and accepted only when they join distinct paths without violating their degree constraints. Let $\hat{A}^{\text{con}}_{uv}$ denote the predicted probability that v immediately follows u on the same machine. Candidate arcs are ranked by

$$R_{uv} = \frac{\hat{A}^{\text{con}}_{uv}}{c_u}, \qquad c_u = \sum_{\substack{v: j(v)=j(u) \\ \text{pos}_{\text{J}}(v) \leq \text{pos}_{\text{J}}(u)}} d_v, \tag{37}$$

Here $j(u)$, $m(u)$, and d_u denote the job, machine, and processing time of operation u, respectively; $\text{pos}_{\text{J}}(u)$ is its position in its job, and $\text{pos}_{\pi_k}(u)$ its position in machine sequence π_k. Thus, c_u is the cumulative processing time of its job up to and including u.

Initially, each operation is a separate path. Candidate arcs are scanned in descending score order, and an arc $u \to v$ is accepted only if u is the tail of one path and v is the head of a different path. This rule preserves at most one predecessor and one successor per operation and prevents cycles within a machine. After $n - 1$ accepted arcs, the resulting path specifies the machine sequence. The decoder scans all directed candidate arcs without a probability threshold, so lower-scoring connections can complete paths when high-scoring predictions conflict or are insufficient. In this way, the greedy path-merging decoder constructs all machine sequences from the predicted consecutive-arc heatmap.

Borda ranking for transitive precedence. For machine k with operation set $\mathcal{O}_k$, Borda decoding assigns each operation its expected number of pairwise precedence wins:

$$B_k(u) = \sum_{v \in \mathcal{O}_k \setminus \{u\}} \hat{E}_{0,uv}, \qquad \pi_k = \text{argsort}^{\text{descending}}_{u \in \mathcal{O}_k} B_k(u). \tag{38}$$

For an exact transitive precedence heatmap, the operation at position r has score $n-r$, so this ranking recovers its machine sequence. For predicted probabilities, it aggregates pairwise preferences into a complete permutation. However, independently ranked machine sequences may form directed cycles when combined with job precedence constraints. The decoder therefore includes feasibility repair before makespan evaluation.

Global scheduling and conflict repair. Once a sequence has been obtained for each machine, the decoder constructs a schedule through the following three steps.

(1) Schedule operations following the machine sequences. For each machine k, a pointer h_k marks the first unscheduled operation in its sequence π_k. The decoder repeatedly scans the machines and considers the operation at each pointer. If all earlier operations of its job have already been scheduled, this operation is assigned the earliest start time allowed by its job and machine availability (Equation 40), and the pointer advances by one position. Otherwise, the machine is skipped for the current scan. When the machine orders are compatible with job precedence, this procedure schedules all operations without changing the orders.

(2) Swap an eligible operation forward when blocked. Repair is triggered only when operations remain but a complete scan schedules none. In this case, the current machine orders and job precedences create a circular dependency: every machine with unfinished operations is waiting for a job predecessor that has not yet been scheduled. The implementation detects this deadlock through the lack of progress, rather than explicitly searching for a cycle. Let $\mathcal{U}$ contain the next unscheduled operation of each unfinished job. Every operation in $\mathcal{U}$ has all its job predecessors scheduled. Among these operations, choose the one closest to the first unscheduled position on its assigned machine:

$$u^* = \arg\min_{u \in \mathcal{U}} \Big(\text{pos}_{\pi_{m(u)}}(u) - h_{m(u)},\ m(u),\ \text{pos}_{\pi_{m(u)}}(u) \Big). \tag{39}$$

Algorithm 3 Heatmap-guided Giffler–Thompson recovery

Require: Instance g, complementary precedence probabilities $\hat{\mathbf{E}}_0$
Set all job and machine ready times to 0; initialize empty machine sequences π_m
Let $\mathcal{U}$ contain the first operation of every job
while $\mathcal{U} \neq \emptyset$ **do**
  Compute $\mathrm{EST}(u)$ and $\mathrm{ECT}(u)$ for $u \in \mathcal{U}$ using Equation 40
  Select $u^* \in \arg\min_{u \in \mathcal{U}} \mathrm{ECT}(u)$
  Form the conflict set $\mathcal{F}$ using Equation 41
  Select $\hat{u} \in \arg\max_{u \in \mathcal{F}} \sum_{v \in \mathcal{F} \setminus \{u\}} \hat{E}_{0,uv}$
  Set $s_{\hat{u}} \leftarrow \mathrm{EST}(\hat{u})$; append $\hat{u}$ to $\pi_{m(\hat{u})}$
  $R_{j(\hat{u})} \leftarrow s_{\hat{u}} + d_{\hat{u}}$; $R_{m(\hat{u})} \leftarrow s_{\hat{u}} + d_{\hat{u}}$
  Remove $\hat{u}$ from $\mathcal{U}$; add its next job operation, if any
end while
return Feasible schedule $\pi = (\pi_m)_{m \in \mathcal{M}}$ with start times s

The tuple is minimized lexicographically. The first component measures the number of sequence positions between u and its machine pointer; machine index and position break ties. The decoder swaps u^* with the operation at that pointer. This is a direct exchange of two positions, not an insertion that shifts all intervening operations. After the swap, u^* is both first in its machine's unscheduled sequence and eligible under job precedence, so it can be scheduled at its earliest feasible start time.

(3) Resume scheduling and repeat. The decoder resumes scheduling from the current machine pointers and performs another swap only if a later scan is blocked. This process continues until all operations have been scheduled, at which point the largest completion time gives the makespan.

D.2 GT DECODING

The predicted heatmap provides preferences that may violate transitivity or global acyclicity. We convert it into a feasible schedule using Giffler–Thompson (GT) construction (Giffler & Thompson, 1960). Let $\mathcal{U}$ contain the next unscheduled operation of every unfinished job. For $u \in \mathcal{U}$, let $R_{j(u)}$ and $R_{m(u)}$ be the completion times of the last scheduled operations in its job and on its machine, respectively. The ready times are initially zero. Its earliest start and completion times are

$$\mathrm{EST}(u) = \max\{R_{j(u)}, R_{m(u)}\}, \qquad \mathrm{ECT}(u) = \mathrm{EST}(u) + d_u. \tag{40}$$

Choose a pivot u^* with minimum ECT and form the conflict set on its machine:

$$\mathcal{F} = \{u \in \mathcal{U} : m(u) = m(u^*), \quad \mathrm{EST}(u) < \mathrm{ECT}(u^*)\}. \tag{41}$$

The heatmap selects the operation with the strongest aggregate preference to precede the other conflicting operations:

$$\hat{u} = \arg\max_{u \in \mathcal{F}} \sum_{v \in \mathcal{F} \setminus \{u\}} \hat{E}_{0,uv}. \tag{42}$$

We schedule $\hat{u}$ at $\mathrm{EST}(\hat{u})$, append it to its machine sequence, update both ready times, and advance its job to the next operation. Pivot ties are broken by earlier start time and then operation index; score ties are broken by earlier completion time and then operation index. Repeating this procedure schedules all operations. Job eligibility enforces job precedence, and machine availability prevents overlap. The GT conflict-set rule yields an active schedule, in which no operation can start earlier without delaying another operation. Feasibility therefore does not depend on the predicted heatmap satisfying all ordering constraints. Algorithm 3 summarizes this decoder. Although decoding is sequential, it requires no additional neural evaluations: non-autoregressive generation refers to the joint prediction of precedence variables.

GT decoding across solution generators. Table 7 evaluates the effect of GT on SLIM (Corsini et al., 2024), BOPO (Liao et al., 2025), and DiffTP. SLIM and BOPO are evaluated on all 80 TA and 80 DMU instances using fixed pretrained checkpoints and seed 12345, with either one random trajectory ($K = 1$) or the best of 512 random trajectories ($K = 512$). Both settings use stochastic action selection: BOPO's default argmax branch at $K = 1$ is replaced by categorical sampling, and

its additional greedy trajectory is disabled at $K = 512$. The benchmark instances and reference makespans match those used for DiffTP. These locally evaluated baselines are reported separately from the literature values in Table 2.

For SLIM and BOPO, +GT restricts each action distribution to the conflict set in Equation 41: logits outside the set are masked, and the remaining probabilities are renormalized before sampling. Model weights, state features, and state-transition rules remain unchanged. The two variants use the same instances and reset the same random seed, but the modified action mask changes subsequent states and hence the generated trajectories; GT is not applied as post-processing to identical completed schedules. In contrast, DiffTP uses its jointly predicted precedence heatmap to select operations within the GT conflict set, without further neural evaluations during decoding.

Table 7: Effect of GT decoding on mean makespan gaps (%) across 80 instances per benchmark. Sampling uses $K = 512$ for SLIM and BOPO and $K = 32$ for DiffTP; DiffTP uses $S = 50$. Bold indicates the lowest gap in each column.

		Single candidate ($K = 1$)		Sampling	
Method	Decoding	TA	DMU	TA	DMU
SLIM	Original	14.69	22.25	7.70	13.33
	+GT	14.33	21.02	7.69	13.14
BOPO	Original	13.99	21.07	7.50	13.17
	+GT	14.29	21.12	7.41	13.27
DiffTP	Borda	8.72	20.48	5.60	14.00
	GT	**7.60**	**15.40**	**5.22**	**11.99**

Note. SLIM and BOPO use stochastic rollouts in both settings. DiffTP single-candidate GT results follow Table 2; Borda sampling follows Table 3, and the remaining DiffTP results use archived evaluations. The single-candidate Borda and GT evaluations are not paired on identical heatmaps.

Effect of GT. For a single sampled trajectory, GT reduces SLIM's mean gap by 0.36 percentage points on TA and 1.23 points on DMU, whereas BOPO's gaps increase by 0.30 and 0.05 points, respectively. With 512 candidates, the changes are smaller: SLIM improves by 0.01 points on TA and 0.19 points on DMU, while BOPO improves by 0.08 points on TA but deteriorates by 0.10 points on DMU (computed before rounding the table entries). Thus, GT can benefit individual autoregressive configurations, but it does not provide consistent improvements across the two pretrained samplers, and its additional benefit is limited under extensive sampling.

For DiffTP with $K = 1$, the archived Borda results are 8.72% on TA and 20.48% on DMU, compared with the GT results of 7.60% and 15.40% reported in Table 2. These values come from separate evaluations and do not constitute a paired comparison on identical sampled heatmaps. With $K = 32$, replacing Borda with GT reduces the mean gap from 5.60% to 5.22% on TA and from 14.00% to 11.99% on DMU. These reductions are larger than those observed for either autoregressive baseline with $K = 512$, supporting the compatibility of GT decoding with the learned precedence heatmap. This comparison concerns inference-time adaptations at the evaluated candidate budgets and seed; it does not establish that GT cannot benefit autoregressive methods trained with the same constraint.

E Training Data Generation

E.1 CP-based Large Neighborhood Search

We generate reference schedules using constraint programming (CP), applying full-instance optimization to smaller instances and CP-based large neighborhood search (CP-LNS) to larger ones. CP-LNS starts from a shortest-processing-time (SPT) schedule and repeatedly reoptimizes a neighborhood of operations using CP. At each iteration, operations are sorted by their current start times. A critical-path operation anchors a contiguous window of up to w operations in this ordering. CP reoptimizes this subproblem subject to boundary conditions imposed by operations outside the window. After a feasible update, the schedule and its critical path are recomputed for the next iteration,

while the best schedule found is retained separately. The procedure continues until the iteration limit or wall-clock budget is reached.

The CP subproblem reoptimizes only the selected operations, while keeping the start times of all other operations fixed during the solve. Let $\mathcal{W}$ denote the selected operation set, and let s_u^{cur} be the current start time of operation u. On each machine k, the selected operations occupy a consecutive block in the current machine order. We define a machine-specific execution interval $[L_k, U_k]$: L_k is the completion time of the outside operation immediately preceding this block, and U_k is the start time of the outside operation immediately following it. If either boundary operation is absent, the corresponding bound is set to zero or the current makespan, respectively. CP may reorder the selected operations on machine k, but each must start no earlier than L_k and finish no later than U_k. Thus, reoptimization preserves their position after the preceding outside operations and before the following outside operations, without changing the order or timing of those outside operations during the subproblem solve. In addition to these machine boundaries, job predecessors and successors outside $\mathcal{W}$ impose release times and completion deadlines. Accordingly, the integer start variable s_u for each $u \in \mathcal{W}$ satisfies

$$L_{m(u)} \le s_u \le U_{m(u)} - d_u, \tag{43}$$

$$s_u \ge s^{\mathrm{cur}}_{\mathrm{pred_J}(u)} + d_{\mathrm{pred_J}(u)}, \quad \mathrm{pred_J}(u) \notin \mathcal{W}, \tag{44}$$

$$s_u + d_u \le s^{\mathrm{cur}}_{\mathrm{succ_J}(u)}, \quad \mathrm{succ_J}(u) \notin \mathcal{W}. \tag{45}$$

The first constraint keeps the entire processing interval of u within its machine-specific execution interval. The latter two constraints preserve job precedence across the subproblem boundary and apply only when the corresponding outside predecessor or successor exists. Within the window, CP also enforces job precedence and non-overlap of operations assigned to the same machine. The operators $\mathrm{pred_J}$ and $\mathrm{succ_J}$ denote immediate job predecessors and successors. The subproblem minimizes $C_{\mathcal{W}}$, the latest completion time among window operations:

$$\min C_{\mathcal{W}}, \qquad C_{\mathcal{W}} \ge s_u + d_u \quad \text{for all } u \in \mathcal{W}. \tag{46}$$

This objective encourages earlier completion of the selected operations within their feasible intervals. After a feasible CP solve, the generator extracts the resulting machine orders and reconstructs a global earliest-start schedule, so operations outside the window may also start earlier at this reconstruction stage. The best schedule found across iterations is retained; the procedure does not certify global optimality.

E.2 Dataset construction and generation time

Each synthetic instance contains n jobs with m operations per job. For each job, the generator independently samples a permutation of the m machines and integer processing times uniformly from $\{1, \ldots, 99\}$, following the Taillard-style distribution in Section 5.1. The training datasets contain 1,000,000 instances of size 10×10, 150,000 distinct instances of size 15×15, and 50,000 instances of size 30×20. For 10×10 instances, full-instance CP terminates upon proving optimality, with a measured mean generation time of 0.674 seconds per instance. For 15×15 and 30×20, CP-LNS is allotted 60 and 120 seconds per instance, respectively, with a window size of $w = 150$ for 15×15 and $w = 200$ for 30×20. Table 8 reports the estimated generation time using 64 CPU workers. The resulting estimates are approximately 3.0, 39.1, and 26.1 hours, for 10×10, 15×15, and 30×20, respectively, or about 68 hours in total.

Table 8: Training dataset sizes and estimated generation times with 64 CPU workers. The 15×15 dataset contains 150,000 distinct instances.

Instance size	Training instances	Estimated time (h)
10×10	1,000,000	3.0
15×15	150,000	39.1
30×20	50,000	26.1

F MORE RESULTS

F.1 DETAILED ABLATION RESULTS

We distinguish three ablation categories: *representation* (w/o TC), *network components* (AGNN and w/o SC), and *decoding* (Non-GT). AGNN denotes the anisotropic graph neural network used in DIFUSCO (Sun & Yang, 2023); w/o SC removes self-conditioning, and Non-GT replaces GT with Borda ranking and feasibility repair. Table 9 expands Table 3 by instance size at $K = 32$ and $S = 50$.

Table 9: Average makespan gaps (%) by instance size in the ablation study. Ours denotes the full DiffTP model. Lower is better.

		Representation	Network components		Decoding	Full model
Dataset	Size	w/o TC	AGNN	w/o SC	Non-GT	Ours
TA	15×15	19.8	6.6	7.1	2.3	**1.9**
	20×15	24.8	9.8	9.2	**5.4**	5.5
	20×20	22.6	15.2	14.1	5.5	**5.1**
	30×15	31.0	16.3	14.7	6.6	**6.5**
	30×20	34.7	17.9	17.7	9.2	**9.0**
	50×15	21.8	12.6	13.8	4.7	**4.4**
	50×20	28.1	13.4	15.8	5.8	**5.3**
	100×20	16.4	7.6	9.4	5.6	**3.8**
	Avg	24.9	12.4	12.7	5.6	**5.2**
DMU	20×15	33.5	16.0	14.1	9.8	**9.6**
	20×20	29.4	19.1	14.7	11.8	**8.8**
	30×15	38.0	20.2	20.6	12.1	**11.3**
	30×20	37.4	23.6	21.5	16.8	**14.0**
	40×15	33.9	18.0	20.5	12.1	**10.7**
	40×20	39.9	23.4	25.4	17.3	**14.2**
	50×15	32.5	17.0	21.5	13.3	**11.5**
	50×20	40.4	22.5	28.2	19.1	**15.7**
	Avg	35.6	20.0	20.8	14.0	**12.0**

Representation ablation protocol. The w/o TC variant retains the model architecture, self-conditioning, optimization-consistency training, and GT decoding, but uses consecutive job relations and direct-successor machine labels in place of transitive closure. The job–machine alternating update rule is unchanged; the job relation support and machine target semantics change with the representation. Both candidate budgets use $S = 50$, and the $K = 32$ set contains the $K = 1$ candidate. All 80 TA and 80 DMU instances are evaluated. The 15×15 and 30×20 stages both use their validation-selected `best-initial` checkpoints rather than their final-epoch checkpoints. These results therefore describe the selected models under the stated protocol, rather than final-epoch performance.

Increasing the candidate budget reduces the w/o TC gap from 38.144% to 24.895% on TA and from 46.351% to 35.633% on DMU, but the $K = 32$ gaps remain above those of the full model at every reported size. This supports the importance of transitive precedence modeling within the evaluated framework. Because GT is designed to aggregate pairwise precedence preferences, changing its input to direct-successor predictions also changes the compatibility between the representation and decoder. The results thus assess removal of transitive closure from the full pipeline, rather than separating the effects of supervision, job connectivity, and decoding. Appendix D.1 describes greedy path merging for consecutive-arc predictions; its evaluation appears in Appendix A.1.

F.2 RESULTS ON THE LAWRENCE BENCHMARK

Table 11 reports the LA results omitted from Table 2, covering eight problem sizes from 10×5 to 30×10. The comparison includes both single-candidate and sampling settings. The method

Table 10: Representation ablation without transitive closure, using the same model architecture and GT decoding with $S = 50$. Mean gaps (%) are shown for one and 32 candidates; — indicates a size absent from the suite.

	TA		DMU	
Size	$K = 1$	$K = 32$	$K = 1$	$K = 32$
15×15	34.259	19.813	—	—
20×15	42.338	24.827	45.904	33.508
20×20	42.235	22.608	41.749	29.439
30×15	45.273	30.983	49.682	38.026
30×20	47.263	34.685	52.149	37.449
40×15	—	—	46.066	33.871
40×20	—	—	48.239	39.879
50×15	33.109	21.778	40.032	32.515
50×20	38.101	28.075	46.989	40.379
100×20	22.571	16.388	—	—
Avg	38.144	24.895	46.351	35.633

Table 11: Average gaps (%) of evaluated methods on the Lawrence (LA) benchmark. "-" indicates unavailable results from the corresponding paper. An asterisk marks problem sizes absent from DiffTP training. The best constructive results are highlighted in bold.

		Non-Constructive				Greedy Constructive							Sampling Constructive							
		Exact Solver	RL-based Improvement			Traditional PDR			RL		SLL		Our	$\text{RL}_{K=128}$		$\text{SLL}_{K=128}$		$\text{SLL}_{K=512}$		$\text{Our}_{K=32}$
	Shape	OR-Tools	L2S_{500}	TGA_{500}	L2S_{5k}	SPT	MOR	MWR	L2D	CL	SLIM	BOPO	DiffTP	L2D	CL	SLIM	BOPO	SLIM	BOPO	DiffTP
LA	10×5*	0.0	2.1	2.1	1.8	14.8	16.0	16.0	14.3	-	9.3	6.0	7.8	8.8	-	1.9	2.7	**1.1**	2.1	4.5
	10×10	0.0	4.4	1.8	0.9	15.7	18.1	12.2	23.7	-	8.9	8.2	6.6	10.4	-	3.1	2.3	2.5	**2.1**	2.8
	15×5*	0.0	0.0	0.0	0.0	14.9	3.9	5.5	7.8	-	2.6	1.1	3.4	2.8	-	0.0	0.0	**0.0**	**0.0**	0.5
	15×10*	0.0	6.4	3.6	3.4	28.7	23.7	17.8	27.2	-	11.6	11.0	6.7	16.2	-	5.2	5.8	5.0	4.9	**2.2**
	15×15	0.0	7.3	5.5	5.9	24.6	18.1	18.2	27.1	-	13.6	12.2	5.9	17.4	-	6.8	6.5	5.6	4.9	**2.2**
	20×5*	0.0	0.0	0.0	0.0	13.7	3.8	5.2	6.3	-	2.1	0.4	2.7	3.1	-	0.0	0.0	**0.0**	**0.0**	1.1
	20×10*	0.0	7.0	5.0	2.6	33.4	20.9	17.2	24.6	-	12.1	12.2	6.9	18.3	-	6.9	5.9	5.6	4.6	**4.2**
	30×10*	0.0	0.2	0.0	0.0	13.9	6.5	8.6	8.4	-	2	2.4	2.1	6.8	-	0.0	0.0	**0.0**	**0.0**	0.8
	Avg	0.0	3.4	2.3	1.8	20.0	13.9	12.6	17.4	-	7.8	6.7	5.3	10.6	-	3.0	2.9	2.5	**2.3**	**2.3**

categories follow the main comparison, and the candidate counts are stated explicitly: DiffTP uses $K = 32$, whereas the sampling baselines use $K = 128$ or 512.

DiffTP achieves an average gap of 5.3% with a single candidate, compared with 6.7% for greedy BOPO and 7.8% for greedy SLIM. With 32 candidates, its average gap decreases to 2.3%, matching BOPO with 512 candidates at the reported precision and improving on SLIM's 2.5% with the same baseline sampling budget. DiffTP obtains the lowest constructive gaps on 15×10, 15×15, and 20×10, while the sampling baselines perform better on the remaining five sizes. These results indicate that its advantage varies across LA problem sizes; the comparison does not establish an equivalent runtime budget. Among the non-constructive methods, OR-Tools reports a zero gap on every size, and L2S with 5,000 iterations achieves an average gap of 1.8%.

F.3 Instance-level means and standard deviations

Table 12 reproduces all eight baseline columns from Table 7 of Corsini et al. (2024) and adds DiffTP with $K = 32$ and $S = 50$. DiffTP statistics use the archived candidate/step sweep; its 50×20 mean rounds to 5.4%, whereas Table 2 reports 5.3% from the main evaluation. Baseline statistics retain the source's reported precision and conventions; the source does not specify the standard-deviation denominator. DiffTP attains a pooled gap of $5.2 \pm 2.4\%$, compared with $7.8 \pm 3.6\%$ for SLIM with 512 candidates. It achieves the lowest constructive mean on seven of the eight sizes in this comparison, with 100×20 as the exception. These descriptive statistics do not establish statistical significance.

Table 12: Mean ± standard deviation of instance-level gaps (%) on TA. Baseline values are reproduced from Table 7 of Corsini et al. (2024); their GM is denoted SLIM here. DiffTP uses $K = 32$, $S = 50$. Each size contains 10 instances; Avg pools all 80 instances. Bold marks the lowest constructive mean.

	Non-Constructive				Sampling Constructive				
Shape	NLS_A	$L2S_{5k}$	MIP	CP	MOR	CL	$SLIM_{128}$	$SLIM_{512}$	$DiffTP_{32}$
15×15	7.7 ± 3.0	6.2 ± 1.4	0.1 ± 0.2	0.1 ± 0.1	12.5 ± 2.6	9.0 ± 2.0	7.2 ± 1.6	6.5 ± 1.3	$\mathbf{1.9 \pm 0.7}$
20×15	12.2 ± 1.7	8.3 ± 1.7	3.2 ± 1.9	0.2 ± 0.4	16.4 ± 1.5	10.6 ± 1.6	9.3 ± 1.8	8.8 ± 1.4	$\mathbf{5.5 \pm 1.0}$
20×20	11.5 ± 1.5	9.0 ± 2.2	2.9 ± 1.6	0.7 ± 0.7	14.7 ± 1.7	10.9 ± 1.6	10.0 ± 1.3	9.0 ± 1.0	$\mathbf{5.1 \pm 0.9}$
30×15	14.1 ± 3.5	9.0 ± 3.0	10.7 ± 3.1	2.1 ± 2.2	17.3 ± 4.5	14.0 ± 3.8	11.0 ± 3.8	10.6 ± 3.6	$\mathbf{6.5 \pm 2.2}$
30×20	16.4 ± 2.4	12.6 ± 2.1	13.2 ± 2.9	2.8 ± 1.5	20.4 ± 2.2	16.1 ± 2.0	13.4 ± 1.6	12.7 ± 1.4	$\mathbf{9.0 \pm 1.4}$
50×15	11.0 ± 4.3	4.6 ± 3.1	12.3 ± 3.9	3.0 ± 0.0	13.5 ± 4.5	9.3 ± 3.8	5.5 ± 3.0	4.9 ± 2.7	$\mathbf{4.4 \pm 2.1}$
50×20	11.3 ± 2.1	6.5 ± 1.9	13.6 ± 4.1	2.8 ± 1.9	14.0 ± 1.5	9.9 ± 2.2	8.4 ± 2.0	7.6 ± 2.0	$\mathbf{5.3 \pm 1.4}$
100×20	5.9 ± 2.3	3.0 ± 1.7	11.0 ± 3.5	3.9 ± 1.5	7.1 ± 2.9	4.0 ± 2.2	2.3 ± 1.1	$\mathbf{2.1 \pm 1.2}$	3.8 ± 1.7
Avg	11.3 ± 4.1	7.4 ± 3.7	8.4 ± 5.9	2.0 ± 1.9	14.5 ± 4.6	10.5 ± 4.2	8.4 ± 3.8	7.8 ± 3.6	$\mathbf{5.2 \pm 2.4}$

G Inference Time and Training Cost

G.1 Timing protocol

We report the archived single-GPU RTX 4090 measurements on 80 TA and 80 DMU instances. The grid contains 4,000 records covering $K \in \{1, 4, 16, 32, 64\}$ and $S \in \{1, 10, 20, 30, 50\}$. Results are pooled by problem size across the two suites. The 15×15 checkpoint handles 15×15, 20×15, and 20×20 instances; the 30×20 checkpoint handles the remaining sizes. Each reported time covers all K candidates for one instance, with the minimum-makespan schedule selected after decoding. The measured time includes denoising, heatmap transfer, GT decoding, and schedule evaluation.

G.2 Runtime comparison, scaling, and component costs

Testing on an RTX 4090 yields runtimes that differ from the original reports; CL remains the exception, with its time taken from Liao et al. (2025). We group the methods in Table 13 into non-constructive, greedy constructive, and sampling methods to distinguish the cost of iterative improvement, single-candidate construction, and multi-candidate generation. Sampling budgets match those used for the solution-quality results: 512 candidates for SLIM and BOPO, and 32 for DiffTP, which uses $S = 50$ in both inference settings. The Greedy group denotes single-candidate evaluation; the local SLIM/BOPO runs sample one trajectory, and DiffTP greedily decodes one denoised heatmap. Thus, the runtime table preserves the candidate budgets of the quality comparison, although single-candidate action selection differs from the literature greedy baselines.

Table 13: Runtime comparison (seconds per instance).

	Non-constructive			Greedy constructive						Sampling		
Shape	$L2S_{500}$	TGA_{500}	$L2S_{5k}$	PDR	L2D	CL	SLIM	BOPO	DiffTP	SLIM	BOPO	DiffTP
15×15	6.67	8.66	56.9	0.0	0.17	0.8	0.39	0.48	0.23	0.37	0.44	2.45
20×15	7.42	9.35	65.6	0.0	0.24	1.1	0.53	0.63	0.23	0.5	0.61	4.19
20×20	7.33	10.6	74.9	0.0	0.35	1.39	0.71	0.83	0.25	0.67	0.79	6.02
30×15	8.54	11.6	77.8	0.01	0.42	1.49	0.79	0.93	0.33	0.81	0.9	8.59
30×20	8.5	13	84	0.02	0.63	1.72	1.05	1.24	0.43	1.08	1.19	13.2
50×15	12.5	16.8	110	0.03	0.94	2.82	1.3	1.53	0.71	1.69	1.48	23.2
50×20	12.1	17.8	118	0.04	1.69	3.93	1.73	2.04	0.97	2.25	1.97	31.7
100×20	25.6	30.3	259	0.14	15.5	9.58	3.45	4.07	3.67	7.26	4.12	120
Average	11.1	14.8	106	0.03	2.49	2.85	1.15	1.35	0.7	1.55	1.32	21.2

In the Greedy group, DiffTP provides the strongest overall runtime performance among the evaluated learning-based constructive methods, attaining the lowest time on six of the eight sizes. The exceptions are 15×15, where L2D is faster, and 100×20, where SLIM is faster. Across the

140 instances evaluated by DiffTP, SLIM, and BOPO, their single-candidate mean times are 0.70, 1.15, and 1.35 seconds, respectively. These results highlight the efficiency of predicting operation relations jointly and constructing a schedule without additional neural evaluations during decoding.

This advantage diminishes as the number of sampled candidates increases. At a fixed S, every candidate requires a complete denoising trajectory and its own decoding, so the total computational work grows with K. Batching can amortize overhead, but does not eliminate this work; as GPU utilization and memory requirements increase, additional candidates lead to approximately linear runtime growth. Table 14 illustrates this trend: for 15×15 at $S = 50$, increasing K from 16 to 32 and 64 raises runtime from 1.21 to 2.45 and 5.07 seconds. The scaling is not strictly linear for small batches or across different instance sizes. Increasing S also adds sequential denoiser evaluations: at $K = 32$, increasing S from 30 to 50 raises the same size's runtime from 1.51 to 2.45 seconds.

Table 14: Mean measured inference time (seconds per instance). Each entry includes all K candidates. TA and DMU instances of the same size are pooled.

	Candidate count K ($S = 50$)					Reverse steps S ($K = 32$)				
Shape	1	4	16	32	64	1	10	20	30	50
15×15	0.23	0.36	1.21	2.45	5.07	0.13	0.56	1.03	1.51	2.45
20×15	0.23	0.55	2.03	4.19	9.38	0.22	0.95	1.76	2.57	4.19
20×20	0.25	0.75	2.92	6.02	12.03	0.30	1.35	2.52	3.69	6.02
30×15	0.33	1.05	4.31	8.59	18.18	0.44	1.93	3.60	6.28	8.59
30×20	0.43	1.48	6.08	13.21	26.18	0.60	2.73	5.09	8.47	13.21
40×15	0.50	1.81	7.29	14.62	29.17	0.74	3.28	6.12	8.95	14.62
40×20	0.65	2.55	12.20	20.39	44.87	1.00	6.57	8.51	12.48	20.39
50×15	0.71	2.81	11.05	23.15	47.30	1.11	5.97	10.26	13.55	23.15
50×20	0.97	3.89	17.41	31.73	64.33	1.59	6.93	13.85	20.82	31.73
100×20	3.67	14.19	61.10	120.21	244.36	5.51	29.45	51.64	79.87	120.21

Table 15 shows that denoising accounts for most of the runtime at $K = 32$, $S = 50$. Memory constraints further affect large instances: 100×20 requires four batches of eight candidates, whereas the other sizes process all 32 candidates in one batch. The resulting time reaches 120 seconds per instance, compared with 2.45 seconds for 15×15. Both repeated denoiser evaluations and candidate batching therefore contribute to the cost of sampling.

Table 15: Inference-time breakdown at $K = 32$, $S = 50$ (seconds per instance). Post-processing includes heatmap transfer, GT decoding, and schedule evaluation. Batch denotes candidates per batch $\times$ number of batches.

Shape	Instances	Denoising	Post-processing	Total	Batch
15×15	10	2.37	0.08	2.45	32×1
20×15	20	4.05	0.13	4.19	32×1
20×20	20	5.85	0.18	6.02	32×1
30×15	20	8.33	0.26	8.59	32×1
30×20	20	11.86	1.35	13.21	32×1
40×15	10	14.18	0.44	14.62	32×1
40×20	10	19.81	0.58	20.39	32×1
50×15	20	21.48	1.67	23.15	32×1
50×20	20	29.85	1.88	31.73	32×1
100×20	10	110.89	9.32	120.21	8×4

Nevertheless, DiffTP with 32 candidates remains faster than the measured L2S_{5k} configuration at every reported size, requiring 2.45–120 seconds compared with approximately 56.9–259 seconds. It does not retain a runtime advantage over SLIM or BOPO with 512 candidates, despite its smaller candidate budget. DiffTP thus offers efficient single-candidate inference, while sampling efficiency remains a limitation. Reducing the number of denoising evaluations and improving the computational and memory efficiency of candidate generation are important directions for future work.

G.3 TRAINING COSTS

Table 16 summarizes approximate training costs for the three dataset sizes described in the main text. The runs use one RTX 4090, a 12-layer Graph Transformer with hidden dimension 128, and AdamW with learning rate 5×10^{-5} and weight decay 10^{-4}. Gradient accumulation maintains an effective batch size near 192, and validation uses 50 instances with GT decoding results as the evaluation metric.

Table 16: Training costs on one RTX 4090. Dataset sizes match the main text. Approximate wall-clock times include validation, checkpointing and testing.

Shape	Training instances	Batch	Effective batch	Target epochs	Time (h)
10×10	1,000,000	96×2	192	20	$\approx$14.2
15×15	150,000	28×7	196	20	≈ 9.4
30×20	50,000	6×32	192	10	$\approx$7.0

H VISUALIZATION OF THE DENOISING PROCESS

H.1 EXPERIMENTAL SETTINGS

We visualize one fixed trajectory for each of the eight DMU problem sizes. Instances are selected as the first file in lexicographic order within each size: DMU01, DMU06, DMU11, DMU16, DMU21, DMU26, DMU31, and DMU36. Neither instances nor trajectories are selected according to solution quality. Each run uses $K = 1$, $S = 50$, $T = 1000$, and the timestep-subsampling procedure described in Appendix C.2.

H.2 PRECEDENCE HEATMAPS ALONG A TRAJECTORY

Figures 7–10 show states after 0, 1, 10, 25, 40, and 50 reverse updates, corresponding to $t = 1000, 980, 796, 490, 184, 0$. Each panel displays the first, middle, and last machine. Entry (u, v) represents precedence of the row operation over the column operation. Both axes retain a fixed job-based ordering. All panels use the same white-to-blue scale from zero to one, with the diagonal excluded.

Intermediate columns show sampled binary states $\mathbf{E}_t$, while the final column shows clean probabilities retained for GT decoding. The black-bordered reference column gives the transitive precedence heatmap of the archived label schedule and is used only for comparison.

As noise decreases, the heatmaps develop more coherent row and column patterns, although the final predictions still differ from the reference. Entries across the diagonal are complementary because opposite precedence directions are mutually exclusive; this property alone does not ensure transitivity across operation triples. An exact transitive precedence heatmap becomes upper triangular when reordered by its machine sequence, but need not have that appearance under the fixed axes used here.

H.3 RECOVERED SCHEDULES

Table 17 reports the final schedule obtained by applying GT to each trajectory's last clean prediction. The eight observed gaps range from 5.52% to 13.82%. For DMU01, GT produces a feasible schedule with makespan 2,751, compared with the reference value 2,563, giving a gap of 7.34%; Figure 6 shows its machine-wise allocation.

Table 17: Final GT-decoded schedules for the fixed visualization instances ($K = 1$, $S = 50$). Reference denotes the makespan of the label schedule.

Shape	Instance	Reference	DiffTP	Gap (%)
20×15	DMU01	2563	2751	7.34
20×20	DMU06	3244	3423	5.52
30×15	DMU11	3430	3904	13.82
30×20	DMU16	3751	4113	9.65
40×15	DMU21	4380	4687	7.01
40×20	DMU26	4647	5164	11.13
50×15	DMU31	5640	6060	7.45
50×20	DMU36	5621	6146	9.34

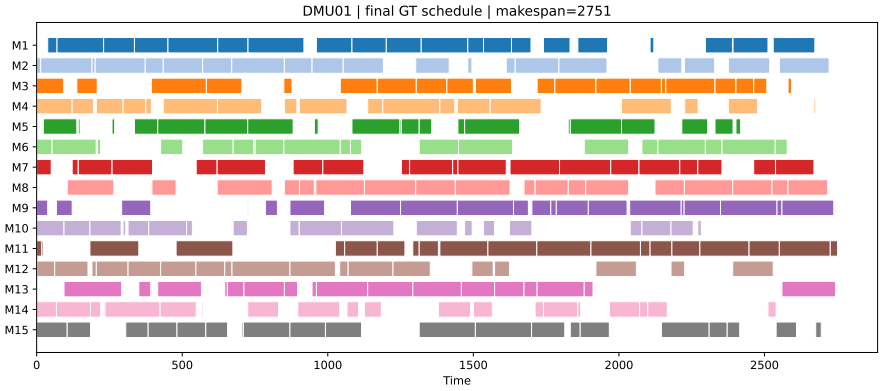

Figure 6: Final GT-decoded schedule for DMU01 (20×15), obtained from the displayed $K = 1$, $S = 50$ trajectory. Rows and colors identify machines, horizontal bars represent operations, and the horizontal axis gives processing time. The makespan is 2,751.

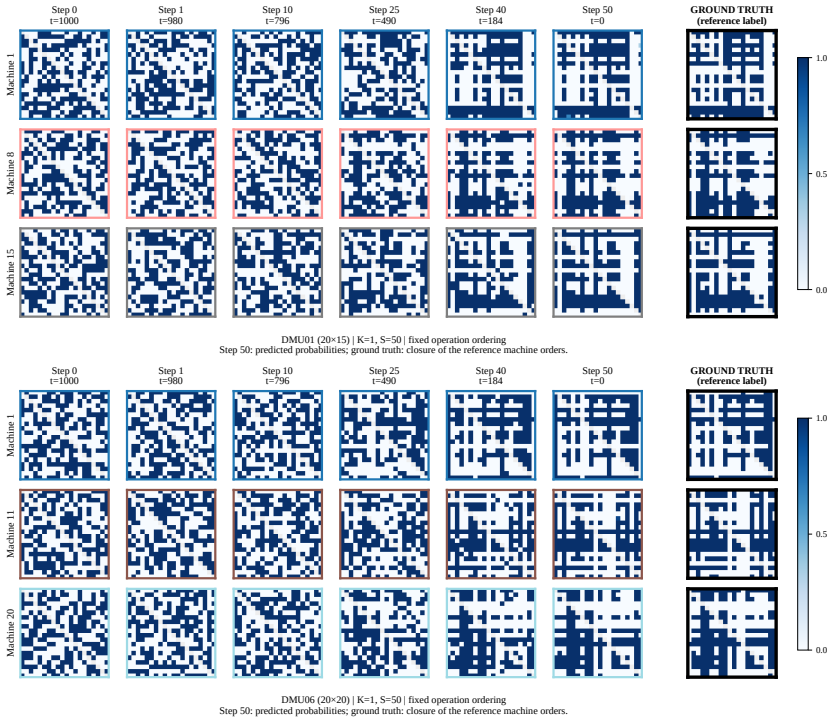

Figure 7: Reverse-state snapshots for DMU01 (20×15, top) and DMU06 (20×20, bottom), with $K = 1$ and $S = 50$. The final step shows clean probabilities; black-bordered reference labels are excluded from sampling.

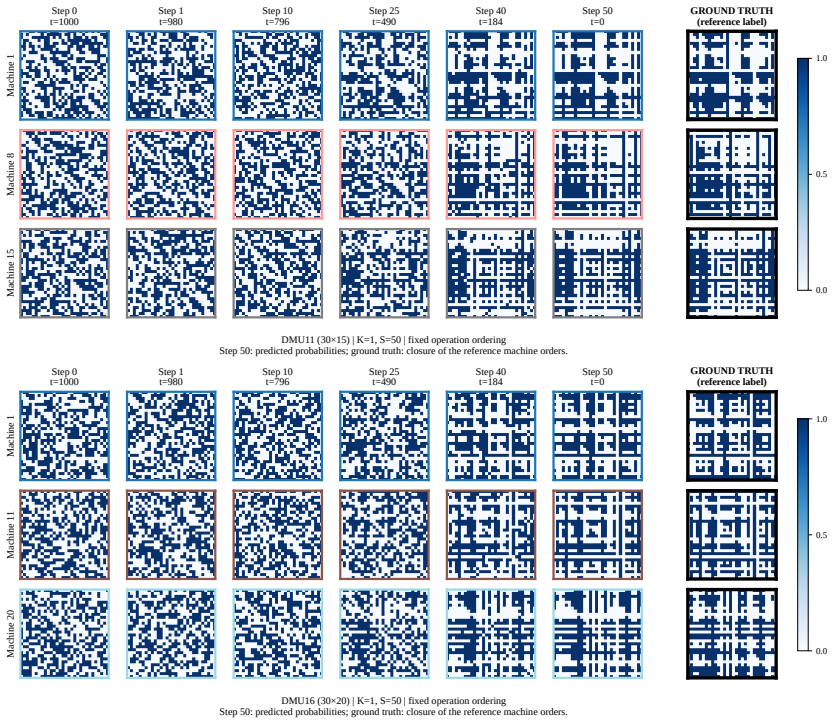

Figure 8: Reverse-state snapshots for DMU11 (30×15, top) and DMU16 (30×20, bottom), with $K = 1$ and $S = 50$. The final step shows clean probabilities; black-bordered reference labels are excluded from sampling.

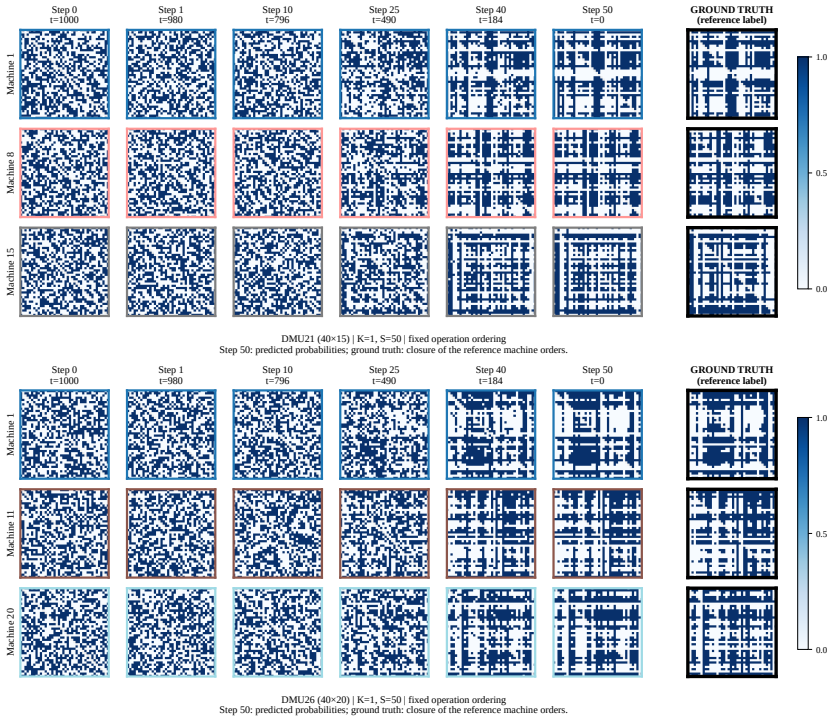

Figure 9: Reverse-state snapshots for DMU21 (40×15, top) and DMU26 (40×20, bottom), with $K = 1$ and $S = 50$. The final step shows clean probabilities; black-bordered reference labels are excluded from sampling.

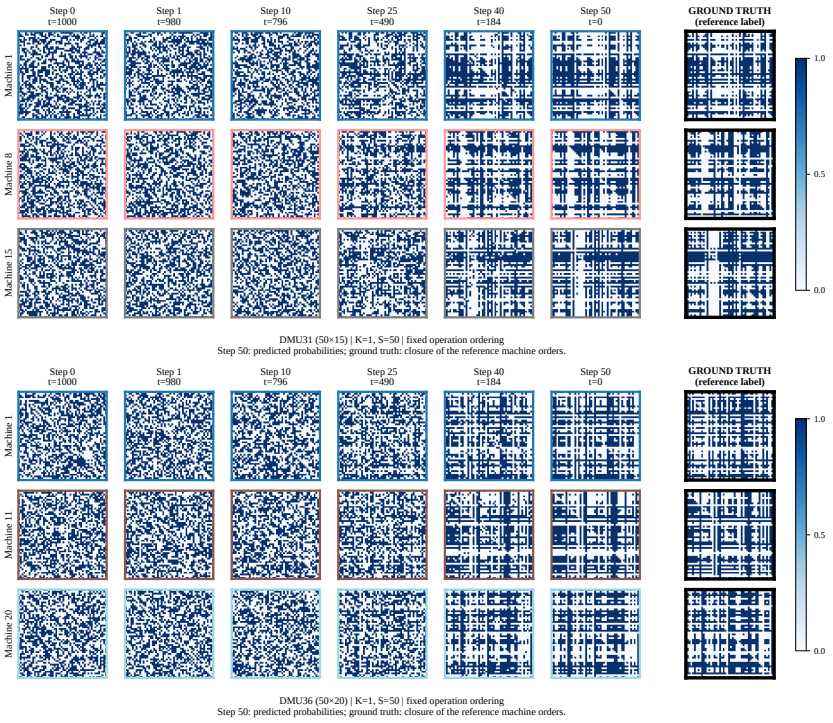

Figure 10: Reverse-state snapshots for DMU31 (50×15, top) and DMU36 (50×20, bottom), with $K = 1$ and $S = 50$. The final step shows clean probabilities; black-bordered reference labels are excluded from sampling.