LEARNING TO CONSTRUCT AND IMPROVE FOR LARGE-SCALE JOB SHOP SCHEDULING VIA DEEP REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Learning-based Job Shop Scheduling Problem (JSSP) solvers have made substantial progress through graph representations, size-generalizable policies, and learned improvement. Yet learning directly on large scheduling instances remains challenging: the solver must represent complex precedence and resource dependencies over long horizons while efficiently allocating computation across large decision and search spaces. We study how to make large-scale JSSP instances first-class training states and propose Learning to Construct and Improve (L2CI), a deep reinforcement learning framework that jointly learns schedule construction and improvement at scale. For scalable representation learning, we develop a structure-aware self-attention Transformer that captures job and machine dependencies, while inducing-point compression provides fixed-size global representations. During construction, successor-state bounds supervise soft pruning of unpromising actions. After construction, an RL-guided critical-path search improves complete schedules and recycles improved solutions as demonstrations for the constructive policy. We train L2CI directly on instances with up to 12,000 operations and achieve strong performance on standard TAI and DMU benchmarks, with learned search further improving the constructed schedules.

1 INTRODUCTION

The Job Shop Scheduling Problem (JSSP) is a canonical combinatorial optimization problem in manufacturing and production planning. Given a set of jobs, each consisting of an ordered sequence of operations competing for machines, the goal is to construct a feasible schedule that minimizes the makespan. JSSP is notoriously challenging because precedence constraints and machine conflicts couple local sequencing decisions into an exponentially large search space Garey et al. (1976). Classical exact methods become expensive as the problem grows, while dispatching rules and metaheuristics typically trade solution quality for efficiency or require substantial domain-specific design (Jain & Meeran, 1999).

Learning-based scheduling offers an alternative approach by amortizing combinatorial optimization into a neural policy. Typical examples include L2D (Zhang et al., 2020), which formulates JSSP as sequential decision-making over a disjunctive graph, and subsequent work has developed stronger graph representations, training strategies, attention mechanisms, and self-improvement strategies (Park et al., 2021; Iklassov et al., 2023; Lee et al., 2024; Pirnay & Grimm, 2024).

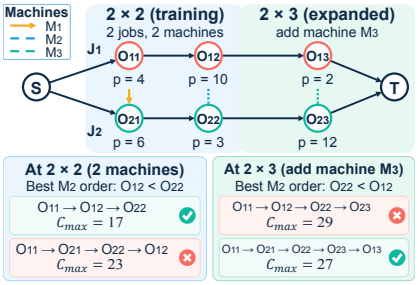


Figure 1: A toy example where an optimal decision can change across scales.

An important property of these methods is *size generalization*: policies trained on small or medium-sized instances can often be applied to substantially larger problems. However, *generalization to large instances is different from learning directly on large instances*. As illustrated in Fig. 1, the disjunctive graph representation of JSSP encodes precedence constraints with directed solid edges and machine conflicts with disjunctive dashed edges, whose

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.

orientations determine scheduling decisions. A policy trained on smaller problems does not experience the state distributions, long decision horizons, and exploration difficulty of the full large-scale optimization process, e.g., the optimal machine order changes from $O_{12} \rightarrow O_{22}$ in the 2×2 instance to $O_{22} \rightarrow O_{12}$ in the scaled 2×3 instance due to the scale expansion. Indeed, Iklassov et al. (2023) have observed that directly training reinforcement-learning schedulers becomes increasingly difficult as instance complexity grows. We therefore study a complementary question: *how should representation and search themselves be redesigned when policy learning is performed directly at scale?*

Direct large-scale learning raises two coupled challenges. The first is representation. As JSSP instances grow, the effect of a single scheduling decision propagates through an increasingly complex dependency graph, creating critical long-range dependencies. Standard Graph Neural Networks (GNNs), however, suffer from over-squashing bottlenecks when capturing such long-range dependencies (Alon & Yahav, 2021; Topping et al., 2022). While recent Transformer-based methods alleviate this limitation to some extent by adopting operation-centric or factorized job/machine representations (Lee et al., 2024; Xiao et al., 2026), we choose to directly model a unified operation–machine space, with masks explicitly encoding job and machine relations. This allows each representation to jointly account for precedence and machine-contention constraints, enabling the model to better capture their coupled effects on scheduling decisions. We further use learnable inducing points (Lee et al., 2019) to compress the representation for efficient policy and value prediction at scale.

The second challenge is searching. As the number of operations grows, uniformly exploring the resulting decision space rapidly becomes ineffective. Inspired by the complementary notions of causal sufficiency and necessity Pearl (1999); Yu et al. (2025), we posit that an effective searching paradigm should answer two questions: whether a partial solution can still lead to a sufficiently good schedule, and whether a candidate decision is consequential for reaching such an outcome. To wit, we employ certified successor-state bounds for soft pruning to filter out branches lacking sufficiency during the construction process, coupled with an improvement loop that learns to allocate a limited budget over critical-path topological swaps (specifically, N5 moves Nowicki & Smutnicki (1996)) to enforce necessity. Ultimately, the best search incumbent serves as the final solution, while the resulting search trajectories are simultaneously recycled as imitation supervision to further refine the constructor.

The resulting framework, L2CI, therefore learns both sides of the solver, equipped with a transformer-based representation architecture where a constructive policy provides strong initial schedules and an improvement policy continues optimizing them.

Our main contributions are fourfold:

- We develop a deep reinforcement learning framework for directly learning on large-scale JSSP instances, and jointly learn schedule construction and improvement.
- We propose a structure-aware Transformer and inducing-point compression, providing more effective representations of job and machine dependencies at large scales.
- We develop a search–policy co-improvement scheme that combines bound-supervised soft pruning with RL-guided critical-path N5 search, and recycles improved solutions to refine the constructive policy.
- We train our framework on JSSP instances with up to 12,000 operations and conduct extensive experiments on standard TAI and DMU benchmarks, demonstrating strong construction performance and further improvements through learned search.

2 Related Work

Learning-based JSSP. Learning-based JSSP solvers commonly formulate scheduling as sequential decision making. L2D (Zhang et al., 2020) learns a size-agnostic dispatching policy over disjunctive graphs, while subsequent work improves graph representations, policy architectures, and training strategies (Park et al., 2021; Iklassov et al., 2023). A recurring focus is size generalization from smaller training instances to larger problems. Iklassov et al. (2023) further show that training becomes harder as instances grow and use curriculum learning to expand the training scale. Our work focuses on direct large-scale learning, incorporating large target instances into policy training.

Representation for JSSP. GNNs are widely used to encode precedence and machine relations in JSSP (Zhang et al., 2020; Park et al., 2021), while recent methods increasingly adopt attention-based architectures. Lee et al. (2024) develop a Transformer-based model with strong size generalization. Rather than introducing Transformer scheduling itself, we combine structure-masked attention to jointly capture job and machine dependencies with inducing-point compression (Lee et al., 2019) to efficiently represent large instances.

Learning to construct and improve. Beyond policy-gradient training, recent work improves constructive policies using solutions generated by the model itself. SLIM (Corsini et al., 2024) uses high-quality sampled schedules as pseudo-labels, while Pirnay & Grimm (2024) iteratively improve the policy using stronger generated solutions. Multi-policy self-evolution further exploits diverse policies to obtain improved supervision (Choi et al., 2025). A complementary line improves complete schedules through local search, with critical-path neighborhoods such as N5 underpinning effective tabu search (Nowicki & Smutnicki, 1996). Recent work learns to navigate these neighborhoods: Zhang et al. (2024a) use reinforcement learning to select local moves, while TBGAT (Zhang et al., 2024b) introduces topology-aware representations for learned neighborhood search. We connect these two lines by using learned critical-path N5 search to improve constructed schedules and turn the improved solutions into demonstrations for subsequent constructive learning.

3 METHOD

L2CI is organized around a closed learning loop. A constructive policy first builds a complete machine ordering from a partial schedule; an improvement policy then spends a bounded search budget on critical-path revisions; finally, the best improved ordering is converted back into a constructive demonstration. We first define the two decision processes and then describe how each stage represents and searches its state space.

3.1 PROBLEM FORMULATION

A JSSP instance I consists of jobs $\mathcal{J}(I) = \{J_1, \dots, J_n\}$, machines $\mathcal{M}(I) = \{M_1, \dots, M_m\}$, and operations $\mathcal{O}(I)$. In the $n \times m$ JSSP, job J_j follows the chain $O_{j1} \to \cdots \to O_{jm}$ and visits every machine exactly once. Here $O_{j\ell}$ is its ℓth operation, so $\mathcal{O}(I) = \{O_{j\ell} : 1 \le j \le n,\ 1 \le \ell \le m\}$ contains nm operations. Each operation $o = O_{j\ell}$ requires uninterrupted processing for $p_o = p_{j\ell} > 0$ time units on its assigned machine $\mu(o) \in \mathcal{M}(I)$. We use o, v as generic operation labels, with analogous subscripts for timing variables and embeddings. A feasible schedule respects these job precedences and processes at most one operation at a time on each machine.

To represent the sequencing decisions, let $\sigma = (\sigma_M)_{M \in \mathcal{M}(I)}$ collect the processing orders on all machines, where σ_M is a total order of the operations assigned to machine M. Let E_J contain the immediate job-precedence arcs and E_M^σ the arcs between consecutive operations on each machine under σ. We introduce zero-duration source and sink nodes $\mathtt{S}$ and $\mathtt{T}$, and let E_0^σ contain the arcs from $\mathtt{S}$ to operations with no incoming arc in $E_J \cup E_M^\sigma$, together with arcs from operations with no outgoing arc in that set to $\mathtt{T}$. The resulting schedule graph is $G(I, \sigma) = \big(\mathcal{O}(I) \cup \{\mathtt{S}, \mathtt{T}\}, E_J \cup E_M^\sigma \cup E_0^\sigma\big)$, with operation durations as node weights. We call σ feasible when $G(I, \sigma)$ is acyclic.

Once a feasible σ is fixed, the earliest-start schedule follows by processing $G(I, \sigma)$ in topological order. Let $c_{j\ell}^{\mathrm{M}}$ denote the completion time of the immediate predecessor of $O_{j\ell}$ on machine $\mu(O_{j\ell})$ under σ, or zero if no such predecessor exists. With $c_{j0} = 0$, the start and completion times are $b_{j\ell} = \max\{c_{j,\ell-1}, c_{j\ell}^{\mathrm{M}}\}$ and $c_{j\ell} = b_{j\ell} + p_{j\ell}$, respectively. The makespan $C(I, \sigma) = C_{\max} = \max_{o \in \mathcal{O}(I)} c_o$ is the minimum achievable under these machine orders. The JSSP objective is to minimize $C(I, \sigma)$ over feasible σ.

For scale normalization, we use the local processing-time scale $\kappa_I = \max_{o \in \mathcal{O}(I)} p_o$ for construction-side time features, rewards, and screening targets. The processing-time features on operation–machine edges also use κ_I in both encoders. For complete-schedule timing features and improvement rewards, we instead use the initial workload lower bound $L_I^{(0)} = \max\{\max_{J_j \in \mathcal{J}(I)} \sum_{\ell=1}^{m} p_{j\ell},\ \max_{M \in \mathcal{M}(I)} \sum_{o \in \mathcal{O}(I):\mu(o)=M} p_o\}$, the largest total processing time assigned to any job or machine. Both factors depend only on the instance and remain fixed through-

out construction and search. This keeps local dispatching quantities on an operation-level scale while expressing complete-schedule improvement relative to the instance workload.

Construction process. We formulate schedule construction as a finite-horizon Markov decision process. At step t, state s_t contains the instance, the dispatched operation set $O_t^{\mathrm{disp}} \subseteq \mathcal{O}(I)$, the partial machine sequences, and their timing information. Since O_t^{disp} forms a prefix of each job, the legal action set $\mathcal{A}_{\mathrm{c}}(s_t) = \{O_{j\ell} \in \mathcal{O}(I) \setminus O_t^{\mathrm{disp}} : \ell = 1 \text{ or } O_{j,\ell-1} \in O_t^{\mathrm{disp}}\}$ contains the first unscheduled operation of every unfinished job. The constructive policy is denoted by $\pi_\theta(a \mid s)$. It selects $a_t \in \mathcal{A}_{\mathrm{c}}(s_t)$ and appends it to the sequence of machine $\mu(a_t)$. Its start time follows the recurrence above, with the current machine tail as its machine predecessor. The transition $s_{t+1} = T_{\mathrm{c}}(s_t, a_t)$ updates the sequence, timing information, and $O_{t+1}^{\mathrm{disp}} = O_t^{\mathrm{disp}} \cup \{a_t\}$. Starting from $O_0^{\mathrm{disp}} = \varnothing$, the process terminates after nm decisions. The resulting trajectory $\tau = (s_0, a_0, \ldots, a_{nm-1}, s_{nm})$ induces the complete machine ordering $\sigma(\tau)$, which preserves the dispatch order on each machine.

Improvement process. Starting from $\sigma_0 = \sigma(\tau)$, improvement is a second finite-horizon MDP with a budget of K search steps. At search step k, starting from $k = 0$, state x_k contains the instance, the current ordering σ_k, the best incumbent schedule, tabu memory, and the remaining search budget. An action $u = (o, v)$ swaps two adjacent operations on the same machine. Candidate pairs are selected at critical-block boundaries along a longest path in $G(I, \sigma_k)$, forming the N5 neighborhood $\mathcal{N}_5(\sigma_k)$; the boundary rules are given in Section 3.3. Let B_k contain the reverse of the most recently accepted swap, with $B_0 = \varnothing$. The legal action set is $\mathcal{A}_{\mathrm{i}}(x_k) = \{u = (o, v) \in \mathcal{N}_5(\sigma_k) : u \notin B_k\}$. The improvement policy is denoted by $\rho_\psi(u \mid x)$. It selects $u_k \in \mathcal{A}_{\mathrm{i}}(x_k)$, and the transition $x_{k+1} = T_{\mathrm{i}}(x_k, u_k)$ applies the swap if the resulting graph is acyclic, retaining the current ordering otherwise. It also updates the schedule, incumbent, tabu memory, and remaining budget. Search ends when the budget is exhausted or no admissible swap remains. Let $k_{\mathrm{end}} \leq K$ denote the final search step. The returned ordering is $\sigma^+ \in \arg\min_{\sigma \in \{\sigma_0, \ldots, \sigma_{k_{\mathrm{end}}}\}} C(I, \sigma)$, which need not coincide with the final ordering $\sigma_{k_{\mathrm{end}}}$ because intermediate swaps may increase makespan. We use c and i to distinguish construction and improvement quantities, respectively.

Policy optimization. We train both policies with clipped proximal policy optimization (PPO) (Schulman et al., 2017), using a separate actor–critic model for each process, where actor defines the action distribution, and critic estimates expected future return to guide policy updates.

3.2 Learning to Construct

A size-agnostic policy can be applied to larger instances, yet a decision that is optimal at one scale can become suboptimal after the instance expands. An extension I' enlarges I while preserving the processing times, machine assignments, and job-precedence relations of its original operations. For a construction state s on I, let $F_I^*(s, a)$ denote the minimum terminal makespan attainable after taking action a, optimizing all subsequent construction decisions.

Theorem 1 (Scale-shift decision reversal)**.** *There exist an instance I and an extension I', with reachable construction states s and s' obtained by dispatching the same sequence of original operations, and two operations $o, v \in \mathcal{A}_{\mathrm{c}}(s) \cap \mathcal{A}_{\mathrm{c}}(s')$ with $\mu(o) = \mu(v)$, such that $F_I^*(s, o) < F_I^*(s, v)$ while $F_{I'}^*(s', o) > F_{I'}^*(s', v)$. Thus the preferred decision for the same machine conflict strictly reverses under the extension, even with optimal subsequent choices.*

The proof in Appendix B formalizes the decision reversal across scales illustrated in Fig. 1. Although useful inductive biases may still enable extrapolation, this reversal motivates exposing the constructor to progressively larger scheduling states, so it can learn how larger-scale dependencies reshape decision quality. Doing so, however, makes both representation and exploration harder; the remaining design of L2CI addresses these two issues directly.

Structure-Aware Residual-State Representation. Direct learning on large instances requires repeatedly reasoning over a large but progressively shrinking scheduling problem. We therefore represent each construction state through its *residual* subproblem using a *Structure-Aware Self-Attention Transformer* (SSAT), as illustrated in Fig. 2(a). SSAT embeds operations and machines in a unified representation space, with job-precedence relations connecting operations of the same job and

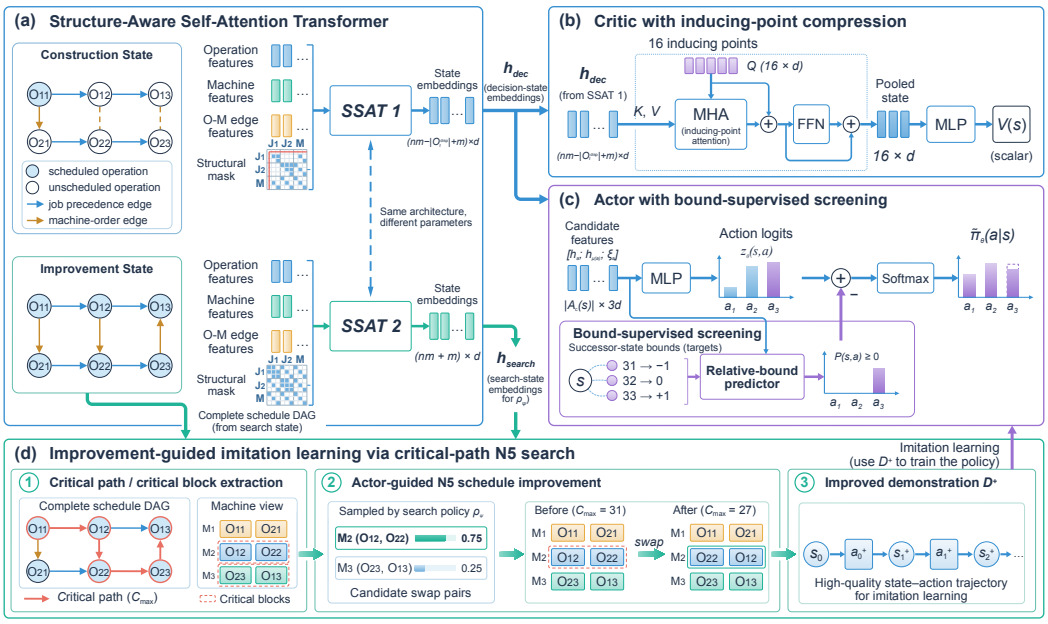


Figure 2: **Overview of L2CI.** (a) Two separately parameterized SSAT encoders represent partial construction DAGs and complete search DAGs. (b) A fixed set of inducing points compresses the variable-size state representation for value estimation. (c) Successor- state lower bounds supervise soft pruning of the constructive actor. (d) An RL-guided critical-path N5 policy improves complete schedules; the best search incumbent is used at inference, and search-improved schedules are linearized into imitation trajectories for construction learning.

operation–machine relations linking each operation to its assigned machine. Once an operation is dispatched, its embedding is removed from the active graph, while its effect is retained through the updated precedence and timing information. The encoder thus focuses on the unresolved part of the scheduling problem.

The active graph determines the attention mask matrix Φ^{c}. During construction, an operation representation may attend to later unscheduled operations in the same job and to its assigned machine. For active query and key nodes v and w, we set $\Phi^{\mathrm{c}}_{vw} = 0$ if $v = w$ or (v, w) is connected by an allowed structural relation, and $\Phi^{\mathrm{c}}_{vw} = -\infty$ otherwise. Scheduled operation embeddings are excluded, whereas machine embeddings remain active throughout construction. Each SSAT layer applies masked multi-head attention followed by residual connections, normalization, and a feed-forward network, propagating information through job and machine dependencies.

Operation features encode processing time and readiness, while machine features encode availability. For each assignment edge $(o, \mu(o))$, we embed p_o/κ_I as $\xi_o \in \mathbb{R}^d$, where d is the embedding dimension. Let $h_v \in \mathbb{R}^d$ denote the final SSAT embedding of operation or machine v at the current state, omitting state arguments for brevity. At construction state s_t, the output matrix $h_{\mathrm{dec}} = [h_v^\top]_{v \in (\mathcal{O}(I) \setminus O_t^{\mathrm{disp}}) \cup \mathcal{M}(I)}$ stacks the active node embeddings as rows and has shape $(nm - |O_t^{\mathrm{disp}}| + m) \times d$. Rows are indexed by node identifiers, with h_v the transpose of row v.

Critic with inducing-point compression. The number of operation and machine representations grows with the instance, whereas the critic requires a compact representation of the whole state. As shown in Fig. 2(b), we use $r = 16$ learnable inducing points $Q \in \mathbb{R}^{r \times d}$ as queries in a multi-head attention block (MAB) over h_{dec}, yielding $Z = \mathrm{MAB}(Q, h_{\mathrm{dec}}) \in \mathbb{R}^{r \times d}$. We average the inducing outputs to obtain the global vector $g_{\mathrm{c}} = r^{-1} \sum_{\nu=1}^{r} Z_{\nu,:}^\top \in \mathbb{R}^d$, where $Z_{\nu,:}$ is the νth row of Z, and pass g_{c} to an MLP value head that outputs the state-value estimate $V(s)$. The fixed number of inducing points decouples the size of the critic input from the number of active operations.

Bound-supervised soft pruning. Direct large-scale construction also enlarges the action-selection problem. Two legal operations may receive similar actor scores even when the partial schedules produced by dispatching them admit very different lower bounds. For each legal action $a \in \mathcal{A}_{\mathrm{c}}(s)$, we compute an admissible lower bound $\underline{C}_{\mathrm{scr}}(T_{\mathrm{c}}(s,a))$ on the makespan achievable from its successor state, using precedence/job-tail information, mandatory machine workload, and selected small ordering relaxations. We normalize these values by κ_I and center them within the current legal action set, $y(s,a) = \frac{1}{\kappa_I}\big[\underline{C}_{\mathrm{scr}}(T_{\mathrm{c}}(s,a)) - \frac{1}{|\mathcal{A}_{\mathrm{c}}(s)|}\sum_{a' \in \mathcal{A}_{\mathrm{c}}(s)} \underline{C}_{\mathrm{scr}}(T_{\mathrm{c}}(s,a'))\big]$, and train a predictor $\widehat{y}_\omega(s,a)$ with a screening loss $\mathcal{L}_{\mathrm{scr}}$ that preserves both the centered values and their relative ordering.

The construction actor scores the concatenated features $[h_a; h_{\mu(a)}; \xi_a]$ to obtain logits $z_\theta(s,a)$, whose softmax over $\mathcal{A}_{\mathrm{c}}(s)$ defines $\pi_\theta(a \mid s)$. We then pass the predicted screening signal through a fixed rectified $\tanh$ transformation, yielding a finite nonnegative penalty $P(s,a) \geq 0$ for adjusting action-selection probabilities. Rollouts use the screened policy $\widetilde{\pi}_\theta(a \mid s) = \frac{\pi_\theta(a|s)\exp[-P(s,a)]}{\sum_{a' \in \mathcal{A}_{\mathrm{c}}(s)} \pi_\theta(a'|s)\exp[-P(s,a')]}$. Soft pruning thus redistributes probability among legal actions while preserving the full action set, such that every legal action retains positive probability. Appendix C details the bound computation and its use in soft pruning.

Construction objective. To obtain step-wise feedback, let $q_j(s)$ be the completion time of the most recently dispatched operation of job J_j (zero if none has been dispatched). Summing its remaining processing times gives the job-tail lower bound $\underline{C}_{\mathrm{job}}(s_t) = \max_{J_j \in \mathcal{J}(I)}\big(q_j(s_t) + \sum_{\ell:\, O_{j\ell} \notin O_t^{\mathrm{disp}}} p_{j\ell}\big)$. The base construction reward is $r_t^{\mathrm{c}} = [\underline{C}_{\mathrm{job}}(s_t) - \underline{C}_{\mathrm{job}}(s_{t+1})]/\kappa_I$ with discount factor $\gamma = 1$. At termination, the bound equals $C(I, \sigma(\tau))$, so the base return telescopes and maximizing it is equivalent to minimizing makespan for a fixed instance. In training, we additionally use the learned residual correction associated with the screening bound as a difference-based shaping signal between successive states to provide finer local feedback. We denote the clipped PPO actor/critic objective, including value regression and entropy regularization, by $\mathcal{L}_{\mathrm{RL}}^{\mathrm{c}}$ (Appendix D).

3.3 Learning to Improve

Once construction produces a complete schedule, L2CI switches from selecting unscheduled operations to revising the machine orders fixed by the constructor.

Complete-schedule representation. Unlike construction, improvement operates on a complete schedule, where both job and machine orderings have been determined. We therefore use the same SSAT architecture with separate parameters to represent the complete schedule graph, as illustrated in Fig. 2(a). In particular, all consecutive machine-precedence relations $E_M^{\sigma_k}$ are available, allowing the structural attention mask matrix Φ^{i} to be directly derived from the complete schedule DAG. Operation–machine assignment relations are incorporated in the same manner.

To support local improvement decisions, operation features additionally characterize the realized schedule structure, including processing time, earliest and latest start times, slack, machine position, and critical-path membership. At state x_k, SSAT2 produces the matrix $h_{\mathrm{search}} = [h_v^\top]_{v \in \mathcal{O}(I) \cup \mathcal{M}(I)} \in \mathbb{R}^{(nm+m)\times d}$, whose rows are the final embeddings of all operations and machines. Applying the inducing-point compression and mean pooling described above, with separate parameters and h_{search} as input, yields the global context vector $g_{\mathrm{i}} \in \mathbb{R}^d$.

Critical-path N5 search. To focus on operations that determine makespan, we restrict search to moves along a longest S–T path in $G(I,\sigma)$, as shown in Fig. 2(d). A maximal consecutive sequence of operations on this critical path processed on the same machine forms a critical block. The N1 neighborhood $\mathcal{N}_1(\sigma)$ swaps adjacent pairs within these blocks, while the Nowicki–Smutnicki N5 neighborhood $\mathcal{N}_5(\sigma) \subseteq \mathcal{N}_1(\sigma)$ retains only their boundary swaps: an interior block contributes its first and last adjacent pair, the first block only its last pair, and the last block only its first pair (Nowicki & Smutnicki, 1996). Classical critical-block dominance shows that the discarded N1 moves cannot yield a strict one-step makespan improvement on the selected critical path. N5 thus offers the learned policy a smaller candidate set than N1, without implying global search completeness.

Improvement policy. The N5 neighborhood determines which local changes are considered; SSAT2 then provides the state representation used to choose among them. For an admissible can-

didate swap $u = (o, v) \in \mathcal{A}_{\mathrm{i}}(x_k)$ on machine M, the actor combines the endpoint vectors h_o, h_v and machine vector h_M, read from the corresponding rows of h_{search}, move features χ_u, and the compressed global context g_{i}: $z_\psi(x_k, u) = f^{\mathrm{i}}_{\mathrm{act}}([h_o; h_v; h_M; \chi_u; g_{\mathrm{i}}])$, where $f^{\mathrm{i}}_{\mathrm{act}}$ is an MLP scoring head that outputs a scalar logit. A softmax over $\mathcal{A}_{\mathrm{i}}(x_k)$ defines $\rho_\psi(u \mid x_k)$. The move features χ_u encode endpoint processing times, critical-block length and boundary side, and search status, including current and best makespans and remaining budget.

Local search may deliberately pass through neutral or worsening schedules to escape a local optimum. We therefore define its reward by improvements to the best schedule found so far. Let $C^\star_k = \min_{0 \le \nu \le k} C(I, \sigma_\nu)$ denote the best makespan retained at search step k. The reward is $r^{\mathrm{i}}_k = (C^\star_k - C^\star_{k+1})/L^{(0)}_I$, with discount factor $\gamma = 1$. A move receives a positive reward only upon finding a new best makespan; otherwise, reward is zero and the incumbent is unchanged.

Theorem 2. *Fix an instance I, an initial schedule σ_0, and a search horizon K. For any realized feasible search trajectory ending at $k_{\mathrm{end}} \le K$ with retained ordering σ^+, $\sum_{k=0}^{k_{\mathrm{end}}-1} r^{\mathrm{i}}_k = [C(I, \sigma_0) - C(I, \sigma^+)]/L^{(0)}_I$. Thus maximizing expected undiscounted return is equivalent to minimizing the expected makespan of the final retained schedule. Moreover, $r^{\mathrm{i}}_k \ge 0$ and $C^\star_{k+1} \le C^\star_k$, so extending the same realized trajectory cannot worsen its retained incumbent.*

The proof is given in Appendix B. This result justifies the reward design that cumulative reward measures exactly the normalized improvement in the returned schedule, aligning reward maximization with final solution quality. The improvement actor and critic are trained with clipped PPO.

Search-guided imitation. Beyond refining the final solution, search provides supervision for the constructor, closing the learning loop in Fig. 2(d). We turn the retained ordering σ^+ into a demonstration by topologically linearizing $G(I, \sigma^+)$. However, the same ordering may admit multiple valid construction sequences. At step t, let s^+_t denote the reconstructed construction state and let $\mathcal{A}^+_{\mathrm{c}}(s^+_t) \subseteq \mathcal{A}_{\mathrm{c}}(s^+_t)$ contain the operations whose machine predecessors in σ^+, if any, have already been emitted. When several operations are admissible, we choose the one preferred by the current constructor, $a^+_t = \arg\max_{a \in \mathcal{A}^+_{\mathrm{c}}(s^+_t)} \pi_\theta(a \mid s^+_t)$. The resulting state–action pairs form the demonstration buffer D^+ and define $\mathcal{L}_{\mathrm{imit}} = -\mathbb{E}_{(s,a)\sim D^+} \log \pi_\theta(a \mid s)$.

Construction learning combines PPO, bound-supervised screening, and search-guided imitation; Appendix D specifies the PPO objective and auxiliary training details. Training alternates construction, N5 improvement, demonstration generation, and constructor updates. At inference, the constructor first produces σ_0; if an improvement budget is available, the learned N5 policy continues from σ_0 and returns the best incumbent encountered. Thus improvement plays two roles in L2CI: it refines the constructed schedule at inference time and provides stronger supervision for subsequent construction learning.

4 Experiments

We evaluate L2CI from three perspectives: (i) greedy construction quality; (ii) the effectiveness of learned improvement and its complementarity with construction; and (iii) construction scalability across instance sizes. For convenience, we denote the construction and improvement components by L2C and L2I, respectively, and their combination by L2CI.

4.1 Experimental Setup

Datasets. We evaluate on eight Taillard (TAI) sizes from 15×15 to 100×20 (Taillard, 1993) and eight Demirkol (DMU) sizes from 20×15 to 50×20 (Demirkol et al., 1998), with ten instances per size. The search-policy evaluation covers TAI, SWV, ORB, and YN (Storer et al., 1992; Applegate & Cook, 1991; Yamada & Nakano, 1992), giving 13 size groups and 114 instances. Scale transfer is evaluated on eight synthetic sizes spanning 2,000–25,000 operations, with ten instances per size. These instances follow Taillard's random generation scheme (Taillard, 1993).

Baselines. We consider the priority dispatching rules SPT, MWR, and MOR; the learned constructors L2D (Zhang et al., 2020), ReSched (Xiao et al., 2026), CL (Iklassov et al., 2023), and SLIM

Table 1: Mean gap (%) on TAI and DMU.

	Shape	PDRs			Construction RL					Improvement RL	
		SPT	MWR	MOR	L2D	ReSched	CL	SLIM	L2C	L2S-500	TBGAT-500
TAI	15×15	26.10	19.10	20.40	26.00	15.74	14.30	13.80	**6.49**	9.30	<u>8.00</u>
	20×15	32.30	23.30	24.90	30.00	19.70	16.50	15.00	**6.32**	11.60	<u>9.90</u>
	20×20	28.30	21.80	22.90	31.60	16.30	17.30	15.20	**8.08**	12.40	<u>10.00</u>
	30×15	35.00	24.10	22.90	33.00	21.50	18.50	17.10	**6.03**	14.70	<u>13.30</u>
	30×20	33.40	24.80	26.80	33.60	22.50	21.50	18.50	**7.57**	17.50	<u>16.40</u>
	50×15	24.00	16.40	17.60	22.40	16.10	12.20	10.10	**0.51**	11.00	<u>9.60</u>
	50×20	25.60	17.80	16.80	26.50	15.60	13.20	<u>11.60</u>	**1.92**	13.00	11.90
	100×20	14.00	8.30	8.70	13.60	9.60	5.90	<u>5.80</u>	**0.25**	7.90	6.40
	Avg.	27.40	19.50	20.10	27.10	17.13	14.90	13.40	**4.65**	12.18	<u>10.69</u>
DMU	20×15	27.90	27.50	30.70	39.00	23.72	-	<u>18.00</u>	**9.37**	21.96	-
	20×20	33.20	26.80	26.30	37.70	22.20	-	19.40	**9.76**	<u>19.26</u>	-
	30×15	31.20	31.90	36.90	42.00	27.80	-	<u>21.80</u>	**8.34**	25.71	-
	30×20	34.40	32.10	32.30	39.70	28.30	-	<u>25.70</u>	**11.29**	26.81	-
	40×15	25.30	27.00	35.80	35.60	26.50	-	<u>17.50</u>	**7.48**	25.97	-
	40×20	33.80	32.30	35.90	39.60	29.20	-	<u>22.20</u>	**10.75**	30.21	-
	50×15	24.30	27.60	34.90	36.50	26.30	-	<u>15.70</u>	**7.13**	26.81	-
	50×20	30.00	30.30	36.80	39.50	31.80	-	<u>22.40</u>	**8.55**	32.76	-
	Avg.	30.00	29.40	33.70	38.70	26.98	-	<u>20.30</u>	**9.09**	26.18	-

(Corsini et al., 2024); and the learned improvement methods L2S (Zhang et al., 2024a) and TBGAT (Zhang et al., 2024b). We use published baseline results where available and otherwise evaluate publicly released checkpoints. Constructors use greedy decoding; improvement methods use matched budgets of 500 moves in Table 1 and 500–5,000 moves in Table 2.

Metrics and references. We report the mean instance-relative gap over an evaluation set D, $\mathrm{Gap}(D) = \frac{100}{|D|} \sum_{i \in D} \left(\frac{C_i}{C_i^{\mathrm{ref}}} - 1 \right)$, where C_i and C_i^{ref} are the method's and reference makespans. All methods in each comparison use the same reference makespan for each instance. Construction and scale-transfer evaluations use 60-minute CP-SAT solutions; search-policy evaluation uses the common benchmark reference values. Lower gaps indicate better schedules; negative gaps indicate makespans below the references. Bold and underline mark the best and second-best results in each comparison, with search-policy results ranked at each size and move budget using unrounded gaps.

Implementation and budgets. L2CI uses 12 layers with embedding dimension 512 and eight attention heads. The constructor trains on instances up to 12,000 operations (400×30) for 300 hours on eight NVIDIA RTX 4090 GPUs. The N5 policy is trained for 2,500 PPO updates using initial schedules from successive stages of constructive learning.

4.2 Experiment Results

Greedy construction quality. Table 1 compares greedy L2C with dispatching rules, learned constructors, and learned improvement methods given 500 N5 moves. Some DMU results are marked as − because the comparable reported results and suitable released checkpoints were unavailable.

Greedy L2C delivers the best solution quality in all 16 TAI and DMU size groups. The gains are substantial: it more than halves the mean gap of the strongest reported baseline on each family, TBGAT-500 on TAI and SLIM on DMU. On DMU, this reduces the gap from 20.30% to 9.09%. The advantage extends to the largest TAI size, 100×20, where L2C attains a mean gap of just 0.25%. Thus, high-quality schedules are already available from greedy construction, before any budget is allocated to improvement.

Learned improvement and its complementarity with construction. We next assess the quality of learned N5 decisions and the contribution of the initial schedule. L2I uses FDD/MWKR initialization, as do L2S and TBGAT. L2CI uses the same learned improvement policy initialized by L2C. Table 2 reports both variants at 500, 1,000, 2,000, and 5,000 moves.

Table 2: Mean gap (%) under equal N5 move budgets.

Dataset	Scale	L2S				TBGAT				L2I				L2CI			
		500	1K	2K	5K	500	1K	2K	5K	500	1K	2K	5K	500	1K	2K	5K
TAI	15×15	9.30	8.60	7.10	6.20	8.00	6.10	5.10	4.60	4.09	3.36	2.77	2.29	**3.18**	**2.67**	**2.35**	**1.97**
	20×15	11.60	10.40	9.40	8.30	9.90	8.70	7.30	6.30	6.75	5.75	4.48	3.75	**3.54**	**3.31**	**3.05**	**2.53**
	20×20	12.40	11.40	10.20	9.00	10.00	9.00	7.90	7.00	6.95	5.57	4.75	3.83	**4.85**	**4.10**	**3.69**	**3.22**
	30×15	14.70	12.90	11.00	9.00	13.30	10.90	9.40	6.80	10.26	8.69	7.51	5.85	**3.68**	**3.44**	**3.06**	**2.48**
	30×20	17.50	15.70	14.00	12.60	16.40	14.00	12.10	9.70	12.55	10.98	9.13	7.42	**5.99**	**5.64**	**5.15**	**4.70**
	50×15	11.00	9.00	6.90	4.60	9.60	7.50	5.60	2.60	7.90	5.67	3.82	1.81	**0.06**	**0.06**	**0.04**	**0.04**
	50×20	13.00	11.40	9.30	6.50	11.90	9.40	7.50	5.00	10.49	8.40	6.31	4.53	**1.02**	**0.95**	**0.87**	**0.64**
	100×20	7.90	6.60	5.10	3.00	6.40	4.90	3.00	1.20	6.52	4.88	3.34	1.76	**0.00**	**0.00**	**0.00**	**0.00**
SWV	20×10	29.60	24.50	21.80	17.80	23.00	19.20	14.60	11.10	19.81	15.58	12.55	9.83	**5.06**	**4.74**	**4.65**	**4.54**
	20×15	25.50	23.50	21.70	17.00	23.70	20.10	18.10	15.70	17.25	14.18	11.48	8.50	**7.76**	**7.20**	**6.61**	**5.90**
	50×10	21.40	20.10	19.00	17.10	20.30	18.50	16.60	14.50	19.66	17.80	15.99	13.54	**3.32**	**2.96**	**2.57**	**2.13**
ORB	10×10	8.20	6.60	5.70	3.80	7.00	5.70	5.10	4.50	3.75	3.06	2.61	2.44	**2.41**	**1.77**	**1.62**	**1.57**
YN	20×20	12.40	10.50	9.60	8.70	9.60	8.00	7.00	5.70	6.39	5.35	3.86	2.78	**4.30**	**3.41**	**2.91**	**2.30**

Table 3: Mean gap (%) on eight synthetic sizes.

Scale	PDRs			Construction RL			Improvement RL
	SPT	MWR	MOR	L2D	SLIM	L2C	L2S-500
100×20	14.66	5.13	5.23	8.01	2.19	**0.18**	4.72
100×40	17.40	7.81	7.59	14.62	3.63	**-2.20**	10.56
100×100	13.41	8.10	7.99	18.51	2.79	**2.19**	14.14
150×100	9.41	3.03	2.78	13.04	-2.83	**-3.25**	9.34
200×20	6.85	1.79	1.17	3.09	0.99	**0.76**	1.13
200×50	8.49	1.61	1.80	5.84	-0.83	**-2.26**	4.68
400×30	3.67	0.94	0.69	3.04	0.73	**0.18**	1.51
500×50	3.42	0.17	0.02	2.89	-0.28	**-0.55**	1.75

Starting from heuristic solutions, L2I already outperforms both L2S and TBGAT in most size–budget comparisons. More importantly, initializing the same policy with L2C yields state-of-the-art performance among the compared methods at every tested size and budget across all four benchmark families. At 5,000 moves, L2C initialization cuts the mean gap by about half on TAI and two-thirds on SWV relative to heuristic initialization. With the policy and move budget held fixed, these gains demonstrate how stronger construction complements learned search.

Scalability across instance sizes. We apply the constructor trained up to 400×30 to eight synthetic sizes without fine-tuning or size-specific parameter changes. The largest, 500×50, contains 25,000 operations, exceeding the maximum operation count seen during training. L2C retains its quality advantage across different instance sizes and job-to-machine ratios, achieving the best mean gap on all eight tested sizes in Table 3. Notably, greedy L2C also improves on the 60-minute CP-SAT reference in four of the eight size groups, as indicated by negative mean gaps.

5 Conclusion

We presented L2CI, a reinforcement learning framework for learning to construct and improve JSSP schedules directly at large scales. Our analysis shows that learning across progressively larger scheduling states is necessary for large-scale JSSP. To achieve this, we propose structure-aware attention and inducing-point compression to represent coupled job and machine dependencies, while bound-supervised soft pruning guides construction. Learned critical-path N5 search further improves schedules and feeds them back as demonstrations for the constructor. Trained on instances with up to 12,000 operations, L2CI achieves state-of-the-art results among the compared learning-based methods, with strong greedy construction, complementary gains from learned improvement, and effective construction on instances with up to 25,000 operations. These results demonstrate the effectiveness of direct large-scale learning with mutually reinforcing construction and improvement.

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

David Applegate and William Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991. doi: 10.1287/ijoc.3.2.149.

Inguk Choi, Woo-Jin Shin, Sang-Hyun Cho, and Hyun-Jung Kim. Towards generalizable multi-policy optimization with self-evolution for job scheduling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024.

Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998. doi: 10.1016/S0377-2217(97)00019-2.

Michael R. Garey, David S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976. doi: 10.1287/moor.1.2.117.

Zangir Iklassov, Dmitrii Medvedev, Ruben Solozabal Ochoa de Retana, and Martin Takác. On the study of curriculum learning for inferring dispatching policies on the job shop scheduling. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 5350–5358. International Joint Conferences on Artificial Intelligence Organization, 2023. doi: 10.24963/ijcai.2023/594.

A. S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2):390–434, 1999. doi: 10.1016/S0377-2217(98)00113-1.

Jaejin Lee, Seho Kee, Mani Janakiram, and George Runger. Attention-based reinforcement learning for combinatorial optimization: Application to job shop scheduling problem. *arXiv preprint*, 2024.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pp. 3744–3753. PMLR, 2019.

Eugeniusz Nowicki and Czeslaw Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996. doi: 10.1287/mnsc.42.6.797.

Junyoung Park, J. Chun, S. H. Kim, Y. Kim, and J. Park. Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. *International Journal of Production Research*, 59(11):3360–3377, 2021. doi: 10.1080/00207543.2020.1870013.

Judea Pearl. Probabilities of causation: Three counterfactual interpretations and their identification. *Synthese*, 121(1-2):93–149, 1999. doi: 10.1023/A:1005233831499.

Jonathan Pirnay and Dominik G. Grimm. Self-improvement for neural combinatorial optimization: Sample without replacement, but improvement. *Transactions on Machine Learning Research (TMLR)*, 2024, 2024. ISSN 2835-8856.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL `https://arxiv.org/abs/1707.06347`.

Robert H. Storer, S. David Wu, and Renzo Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992. doi: 10.1287/mnsc.38.10.1495.

Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993. doi: 10.1016/0377-2217(93)90182-M.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022.

Xiangjie Xiao, Cong Zhang, Wen Song, and Zhiguang Cao. RESCHED: Rethinking flexible job shop scheduling from a transformer-based architecture with simplified states. In *The Fourteenth International Conference on Learning Representations*, 2026.

Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop problems. In Reinhard Manner and Bernard Manderick (eds.), *Parallel Problem Solving from Nature 2 (PPSN II)*, pp. 281–290. North-Holland, 1992.

Xiangning Yu, Zhuohan Wang, Linyi Yang, Haoxuan Li, Anjie Liu, Xiao Xue, Jun Wang, and Mengyue Yang. Causal sufficiency and necessity improves chain-of-thought reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 1621–1632. Curran Associates, Inc., 2020.

Cong Zhang, Zhiguang Cao, Wen Song, Yaoxin Wu, and Jie Zhang. Deep reinforcement learning guided improvement heuristic for job shop scheduling. In *International Conference on Learning Representations (ICLR)*, 2024a.

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. In *Proceedings of the 40th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2024b.

A Verification of the Toy Example in Figure 1

Proof. Consider the two-job, two-machine instance I with job chains $O_{11} \to O_{12}$ and $O_{21} \to O_{22}$. Operations $O_{1\ell}$ and $O_{2\ell}$ use machine M_ℓ, and their processing times are

$$(p_{11}, p_{12}) = (4, 10), \qquad (p_{21}, p_{22}) = (6, 3). \tag{A.1}$$

Construct I' by appending O_{13} and O_{23} on a new machine M_3, with $p_{13} = 2$ and $p_{23} = 12$. All original operations, precedence relations, processing times, and machine assignments are retained, so I' is an extension of I.

As in Fig. 1, compare the next decision after dispatching O_{11} and then O_{21} on both instances. This common construction prefix produces $O_{11} \prec O_{21}$ on M_1 and leaves O_{12} and O_{22} as the two legal actions. We evaluate each choice with its best subsequent completion; on I', this includes choosing the order on the new machine M_3.

On machine M_ℓ, use $+$ to denote $O_{1\ell} \prec O_{2\ell}$ and $-$ to denote $O_{2\ell} \prec O_{1\ell}$. For each feasible machine ordering, we compute the earliest completion times using the recurrence in Section 3.1. Every feasible schedule with the same ordering satisfies the job and machine precedence constraints, so its completion times cannot be smaller. Enumerating these orderings gives Table 4.

Table 4: Exhaustive machine-order enumeration for the scale-reversal witness. C_I is the makespan of I; $C^+_{I'}$ and $C^-_{I'}$ are the makespans of I' under $O_{13} \prec O_{23}$ and $O_{23} \prec O_{13}$, respectively. The first two rows correspond to the common construction prefix (O_{11}, O_{21}) used in Fig. 1 and Theorem 1.

M_1	M_2	C_I	$C^+_{I'}$	$C^-_{I'}$
$+$	$+$	17	29	31
$+$	$-$	23	37	27
$-$	$+$	23	35	37
$-$	$-$	20	34	23

For I, choosing O_{12} next gives makespan 17, with critical path $O_{11} \to O_{12} \to O_{22}$; choosing O_{22} gives 23, with critical path $O_{11} \to O_{21} \to O_{22} \to O_{12}$. Thus O_{12} is the optimal next action after the common prefix.

For I', keeping this decision gives makespans 29 and 31 under $O_{13} \prec O_{23}$ and $O_{23} \prec O_{13}$, respectively. Choosing O_{22} instead gives 37 and 27 under those same two orders. The best completion after O_{12} therefore has makespan 29, whereas the best completion after O_{22} has makespan 27: the preferred next action has reversed. Optimizing the new machine order alone cannot compensate for retaining the original M_2 decision.

These best completions are the lower-right comparison in Fig. 1. Their critical paths are $O_{11} \to O_{12} \to O_{22} \to O_{23}$ for the makespan-29 schedule and $O_{11} \to O_{21} \to O_{22} \to O_{23} \to O_{13}$ for the makespan-27 schedule. The undirected M_3 edge indicates that its order remains available for optimization in both cases. The example thus isolates a reversal caused by instance extension while keeping the earlier construction decisions unchanged, providing the witness for Theorem 1. □

B Proofs of the Main Results

B.1 Proof of Theorem 1

Proof. Take the instance extension and common construction prefix (O_{11}, O_{21}) verified in Appendix A, and denote the resulting states by s and s'. Both states admit O_{12} and O_{22} on M_2. The first two rows of Table 4 exhaust all machine orders consistent with this prefix. Each ordering has a topological construction sequence extending the prefix, so minimizing these entries gives the best attainable continuation after each action:

$$\begin{aligned} F^*_I(s, O_{12}) &= 17 < 23 = F^*_I(s, O_{22}), \\ F^*_{I'}(s', O_{12}) &= 29 > 27 = F^*_{I'}(s', O_{22}). \end{aligned} \tag{B.1}$$

The extension preserves all original processing times, machine assignments, and job precedences. Taking $o = O_{12}$ and $v = O_{22}$ therefore satisfies every condition of the theorem and establishes the strict reversal. □

B.2 Proof of Theorem 2

Proof. For every step before termination, the incumbent satisfies

$$C^\star_{k+1} = \min_{0\le\nu\le k+1} C(I,\sigma_\nu) = \min\{C^\star_k, C(I,\sigma_{k+1})\} \le C^\star_k. \tag{B.2}$$

Thus $r^{\mathrm{i}}_k \ge 0$, and the retained makespan cannot increase along the same trajectory. Summing until the actual termination step telescopes:

$$\begin{aligned}\sum_{k=0}^{k_{\mathrm{end}}-1} r^{\mathrm{i}}_k &= \frac{1}{L_I^{(0)}} \sum_{k=0}^{k_{\mathrm{end}}-1} (C^\star_k - C^\star_{k+1}) \\ &= \frac{C^\star_0 - C^\star_{k_{\mathrm{end}}}}{L_I^{(0)}} = \frac{C(I,\sigma_0) - C(I,\sigma^+)}{L_I^{(0)}}.\end{aligned} \tag{B.3}$$

Here $C^\star_{k_{\mathrm{end}}} = C(I,\sigma^+)$ by the definition of the returned ordering. This identity holds whether search exhausts its budget or terminates early. Since I and σ_0 are fixed, taking expectations over search trajectories gives

$$\mathbb{E}\left[\sum_{k=0}^{k_{\mathrm{end}}-1} r^{\mathrm{i}}_k\right] = \frac{C(I,\sigma_0) - \mathbb{E}[C(I,\sigma^+)]}{L_I^{(0)}}. \tag{B.4}$$

Because $L_I^{(0)} > 0$, maximizing expected undiscounted return is equivalent to minimizing the expected makespan of the final retained schedule. □

C Bound-Supervised Soft Pruning

The screening bound combines remaining machine workload with small ordering relaxations. The former captures unavoidable processing requirements, while the latter accounts for conflicts that aggregate workload alone may overlook. We first describe this bound and then explain how it supervises action selection.

C.1 Screening-Bound Computation

For a construction state s, the bound is

$$\underline{C}_{\mathrm{scr}}(s) = \max\{\underline{C}_{\mathrm{RT}}(s), \underline{C}_{4\mathrm{op}}(s)\}. \tag{C.1}$$

The two terms share the release times and job tails defined below. Evaluating the same expression at $T_{\mathrm{c}}(s,a)$ gives the successor-state bound used to supervise action a.

Release times and job tails. Let $U_M(s)$ contain the undispatched operations on machine M, $w_M(s)$ be its last dispatched completion time, and $C_{\mathrm{disp}}(s)$ be the largest dispatched completion time; missing completion times are zero. For an unfinished job J_j whose first undispatched operation is $O_{j\ell_0}$, propagate release-time lower bounds along its remaining operations:

$$\begin{aligned} r_{j\ell_0} &= \max\big\{q_j(s), w_{\mu(O_{j\ell_0})}(s)\big\}, \\ r_{j\ell} &= \max\big\{r_{j,\ell-1} + p_{j,\ell-1}, w_{\mu(O_{j\ell})}(s)\big\}, && \ell_0 < \ell \le m, \\ \eta_{j\ell} &= \sum_{\nu=\ell+1}^{m} p_{j\nu}, && \ell_0 \le \ell \le m. \end{aligned} \tag{C.2}$$

Here $q_j(s)$ is the completion time of the last dispatched operation in J_j. The release time r_o bounds when o can start, while η_o accounts for the processing its job still requires after o.

Machine-workload bound. Operations with late release times and long job tails impose a joint restriction on the makespan. For $o, v \in U_M(s)$, define $W_M(o,v) = \{w \in U_M(s) : r_w \geq r_o,\ \eta_w \geq \eta_v\}$. All operations in this set must be processed on M after time r_o, and the last to finish still requires a job tail of at least η_v. Thus

$$\underline{C}_{\mathrm{RT}}(s) = \max\Big\{C_{\mathrm{disp}}(s),\ \max_{M\in\mathcal{M}(I)}\ \max_{\substack{o,v\in U_M(s)\\ W_M(o,v)\neq\varnothing}}\Big(r_o + \sum_{w\in W_M(o,v)} p_w + \eta_v\Big)\Big\}. \tag{C.3}$$

Small ordering relaxations. The workload term does not resolve all ordering conflicts. We therefore select subsets S of at most four operations on one machine and solve each release–tail ordering problem exactly. For an ordering $\varpi = (\varpi_1, \ldots, \varpi_{|S|})$ of S, let c_ν^ϖ denote the earliest completion time of its νth operation ϖ_ν in this relaxed single-machine problem. Starting from $c_0^\varpi = 0$, we compute

$$\begin{aligned} c_\nu^\varpi &= \max\{c_{\nu-1}^\varpi, r_{\varpi_\nu}\} + p_{\varpi_\nu}, \qquad 1 \leq \nu \leq |S|, \\ B(S) &= \min_{\varpi\in\mathrm{Perm}(S)}\ \max_{1\leq\nu\leq|S|}\big(c_\nu^\varpi + \eta_{\varpi_\nu}\big). \end{aligned} \tag{C.4}$$

Here $\mathrm{Perm}(S)$ denotes all permutations of S; the completion times are recomputed for each one to obtain the optimal relaxed value $B(S)$. Each subset requires at most $4! = 24$ orders. On each nonempty machine, we select five subsets by taking the top $\min\{4, |U_M(s)|\}$ operations using p_o, r_o, η_o, the individual-operation bound $\beta_o = r_o + p_o + \eta_o$, and the pairwise contention score

$$d_o = \sum_{v\in U_{\mu(o)}(s)\setminus\{o\}} \big[B(\{o,v\}) - \max\{\beta_o, \beta_v\}\big]_+. \tag{C.5}$$

Here $[z]_+ = \max\{z, 0\}$, and d_o measures the extra cost of pairwise machine contention beyond the individual operation bounds. Computing $B(\{o,v\})$ requires only the two possible orders; ties in subset selection follow job index. Denoting the selected subsets by $S_M^{\mathrm{sel}}(s)$, we obtain

$$\underline{C}_{4\mathrm{op}}(s) = \max\Big\{C_{\mathrm{disp}}(s),\ \max_{M\in\mathcal{M}(I)}\ \max_{S\in S_M^{\mathrm{sel}}(s)} B(S)\Big\}. \tag{C.6}$$

Empty machines are skipped. At termination, both bounds reduce to $C_{\mathrm{disp}}(s)$.

Proposition 1 (Admissibility of the screening bound)**.** *For any reachable construction state s, every complete schedule extending its dispatched machine prefixes has makespan at least $\underline{C}_{\mathrm{scr}}(s)$.*

Proof. Release times and tails bound the required job timing. Each workload term follows from non-overlapping processing on one machine. Each $B(S)$ optimizes a relaxation retaining only the selected machine operations and their release times and tails, so it cannot exceed the makespan of any feasible completion. The fixed-prefix completion time $C_{\mathrm{disp}}(s)$ is also a lower bound; taking their maximum preserves admissibility. □

C.2 Learning and Applying the Screening Signal

Screening supervision serves two purposes: fitting the centered bound of each action and preserving the differences between candidates. Using the targets $y(s,a)$ defined in Section 3.2, we center the predictions and define their errors as

$$\begin{aligned} \bar{y}_\omega(s,a) &= \widehat{y}_\omega(s,a) - \frac{1}{|\mathcal{A}_{\mathrm{c}}(s)|}\sum_{b\in\mathcal{A}_{\mathrm{c}}(s)} \widehat{y}_\omega(s,b), \\ e_\omega(s,a) &= \bar{y}_\omega(s,a) - y(s,a). \end{aligned} \tag{C.7}$$

The pairwise term focuses on actions whose target values differ by at least $\epsilon_{\mathrm{pair}} > 0$. These unordered pairs form the set

$$B_{\mathrm{pair}}(s) = \big\{\{a,b\} \subseteq \mathcal{A}_{\mathrm{c}}(s) : a \neq b,\ |y(s,a) - y(s,b)| \geq \epsilon_{\mathrm{pair}}\big\}. \tag{C.8}$$

Counting each pair once, we minimize

$$\begin{aligned} \mathcal{L}_{\mathrm{scr}} = \mathbb{E}_s\Bigg[\frac{1}{|\mathcal{A}_{\mathrm{c}}(s)|}\sum_{a\in\mathcal{A}_{\mathrm{c}}(s)} H_{\delta_{\mathrm{scr}}}\big(e_\omega(s,a)\big)\Bigg] \\ + \lambda_{\mathrm{pair}}\mathbb{E}_s\Bigg[\frac{1}{\max\{1, |B_{\mathrm{pair}}(s)|\}}\sum_{\{a,b\}\in B_{\mathrm{pair}}(s)} H_{\delta_{\mathrm{scr}}}\big(e_\omega(s,a) - e_\omega(s,b)\big)\Bigg]. \end{aligned} \tag{C.9}$$

Here $\mathbb{E}_s$ denotes averaging over sampled nonterminal construction states. The coefficient $\lambda_{\text{pair}} \geq 0$ weights the pairwise contribution, and H_δ is the Huber loss,

$$H_\delta(e) = \begin{cases} \frac{1}{2}e^2, & |e| \leq \delta, \\ \delta\big(|e| - \frac{1}{2}\delta\big), & |e| > \delta. \end{cases} \tag{C.10}$$

The second term vanishes when no pair qualifies. Normalizing within each state gives sampled states equal weight regardless of their candidate counts. We use $\delta_{\text{scr}} = \epsilon_{\text{pair}} = 0.01$ and $\lambda_{\text{pair}} = 0.25$; during guided updates, $\mathcal{L}_{\text{scr}}$ enters the construction objective with unit weight.

Probability adjustment. The centered prediction becomes a bounded action penalty through

$$P(s,a) = \lambda_{\text{p}} \tanh\left(\frac{[\bar{y}_\omega(s,a) - \epsilon_{\text{p}}]_+}{\tau_{\text{p}}}\right). \tag{C.11}$$

The strength $\lambda_{\text{p}} \geq 0$ limits the penalty magnitude, $\epsilon_{\text{p}} \geq 0$ sets a tolerance above the legal-action mean, and $\tau_{\text{p}} > 0$ controls how smoothly the penalty saturates. Subtracting $P(s,a)$ from the actor logits yields the screened policy in Section 3.2, with $0 \leq P(s,a) \leq \lambda_{\text{p}}$. We use $\epsilon_{\text{p}} = 0.01$, $\tau_{\text{p}} = 0.1$, and $\lambda_{\text{p}} \in [0, 0.25]$.

D Training Details

D.1 Policy Optimization

Construction and improvement use clipped PPO with separate actor and critic parameters. We use (s_t, a_t) and V generically for the state–action pair and critic of either process, with π_{train} denoting $\widetilde{\pi}_\theta$ for construction or ρ_ψ for improvement. The superscript old identifies the parameters used to collect the rollout, and $R_t^\pi = \pi_{\text{train}}(a_t \mid s_t)/\pi_{\text{train}}^{\text{old}}(a_t \mid s_t)$ is the probability ratio. For construction, the recorded penalties $P^{\text{old}}(s_t, a)$ are held fixed in both policies entering this ratio throughout the PPO updates. The objective, denoted by $\mathcal{L}_{\text{RL}}^{\text{c}}$ for construction, is

$$\begin{aligned} \mathcal{L}_{\text{RL}} = & -\mathbb{E}_t\Big[\min\big\{R_t^\pi \widehat{A}_t, \text{clip}(R_t^\pi, 1-\epsilon, 1+\epsilon)\widehat{A}_t\big\}\Big] \\ & + c_V \mathbb{E}_t\Big[\max\big\{(V(s_t) - \widehat{G}_t)^2, (V_t^{\text{clip}} - \widehat{G}_t)^2\big\}\Big] \\ & - c_H \mathbb{E}_t\Big[\mathcal{H}\big(\pi_{\text{train}}(\cdot \mid s_t)\big)\Big], \end{aligned} \tag{D.1}$$

where $\mathbb{E}_t$ averages over valid rollout transitions, $\widehat{A}_t$ is the normalized advantage, $\widehat{G}_t$ is the value target, and $\mathcal{H}$ is categorical entropy over legal actions. The coefficients $c_V = 0.5$ and $c_H = 0.01$ weight value regression and entropy, respectively, in both processes. Value clipping uses

$$V_t^{\text{clip}} = V^{\text{old}}(s_t) + \text{clip}\big(V(s_t) - V^{\text{old}}(s_t), -\epsilon, \epsilon\big), \tag{D.2}$$

with the same clip range $\epsilon = 0.2$ as the policy objective.

Targets are computed by generalized advantage estimation (GAE). Let $d_t \in \{0, 1\}$ indicate termination and let r_t denote the shaped construction reward $\widetilde{r}_t^{\text{c}}$ defined below, or the improvement reward r_t^{i}. The recursion is

$$\begin{aligned} \delta_t &= r_t + \gamma(1 - d_t)V^{\text{old}}(s_{t+1}) - V^{\text{old}}(s_t), \\ A_t^{\text{GAE}} &= \delta_t + \gamma\lambda_{\text{GAE}}(1 - d_t)A_{t+1}^{\text{GAE}}, \\ \widehat{G}_t &= V^{\text{old}}(s_t) + A_t^{\text{GAE}}. \end{aligned} \tag{D.3}$$

Here δ_t is the temporal-difference residual and A_t^{GAE} is the unnormalized advantage estimate. The discount factor is $\gamma = 1$, and $\lambda_{\text{GAE}} = 0.98$ controls the advantage trace. We standardize A_t^{GAE} over active transitions to obtain $\widehat{A}_t$, keeping $\widehat{G}_t$ unchanged. The backward recursion starts from zero beyond the rollout. Terminal states have zero bootstrap value; nonterminal improvement rollout boundaries bootstrap from the critic. All loss averages exclude inactive transitions, and rollout targets remain fixed during PPO updates.

D.2 Auxiliary Training Signals

Reward correction. To supplement the job-tail bound, a scalar head $f_{\mathrm{res}}(s)$ taking the constructor's compressed global representation g_{c} as input learns the residual $[\underline{C}_{\mathrm{scr}}(s) - \underline{C}_{\mathrm{job}}(s)]/\kappa_I$ by Huber regression with threshold 1 and unit loss weight. Its change supplies finer local feedback through

$$\widetilde{r}_t^{\mathrm{c}} = r_t^{\mathrm{c}} + \alpha_{\mathrm{res}}\big[f_{\mathrm{res}}^{\mathrm{old}}(s_t) - f_{\mathrm{res}}^{\mathrm{old}}(s_{t+1})\big], \tag{D.4}$$

where $\alpha_{\mathrm{res}} \in [0, 1]$ is the correction strength. The correction function and weight are fixed within each rollout, and the recorded predictions remain fixed during PPO updates. Setting the terminal prediction to zero gives

$$\sum_{t=0}^{nm-1} \widetilde{r}_t^{\mathrm{c}} = \frac{\underline{C}_{\mathrm{job}}(s_0) - C(I, \sigma(\tau))}{\kappa_I} + \alpha_{\mathrm{res}} f_{\mathrm{res}}^{\mathrm{old}}(s_0). \tag{D.5}$$

For a fixed initial state, the added term is independent of the chosen actions. Thus shaping provides local feedback while preserving the makespan objective.

Demonstration returns. Demonstrations also supervise the critic using

$$G_t^+ = \sum_{\nu=t}^{nm-1} r_\nu^{\mathrm{c},+} + \alpha_{\mathrm{res}} f_{\mathrm{res}}^{\mathrm{old}}(s_t^+), \tag{D.6}$$

where $r_\nu^{\mathrm{c},+}$ is the base reward from replaying the demonstration. The correction aligns these targets with the shaped construction return. A one-sided Huber regression with threshold 1 updates the critic only where $G_t^+ > V(s_t^+)$, averaging over qualifying samples and contributing zero when none qualify. The targets and selection condition are treated as constants for differentiation. The demonstration value loss has weight 0.02, while the actor imitation weight lies in $[0, 0.6]$.

E Additional Experimental Results

E.1 Performance across Training-Scale Stages

We examine how construction quality develops as the training curriculum expands from 10×10 to 400×30. Figure 3 evaluates L2C-10, L2C-15, L2C-20, L2C-30, L2C-50, L2C-100, L2C-200, and L2C-400 on the same eight-size TAI panel. Gaps are measured against 60-minute CP-SAT reference solutions. The architecture and greedy decoding rule remain fixed, allowing the panel to track cross-scale performance throughout the expanding-scale training run.

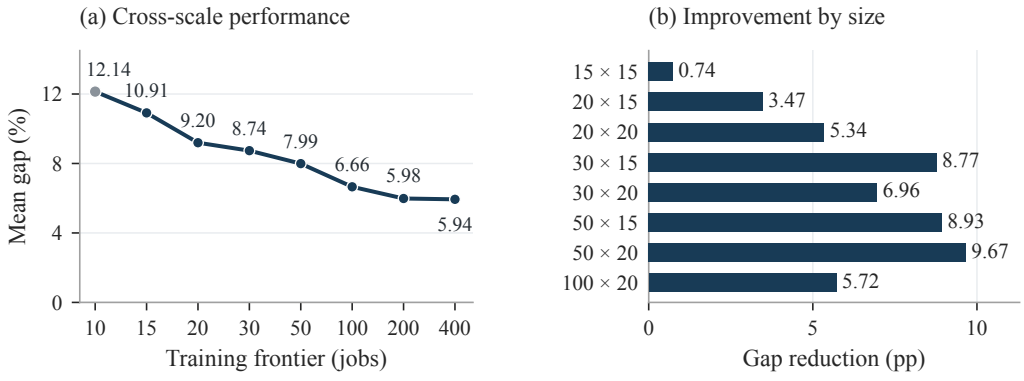


Figure 3: Construction quality as the training scale expands. (a) Mean gap on the same eight-size TAI panel at eight training frontiers. (b) Per-size gap reduction from L2C-10 to L2C-400, measured in percentage points (pp). L2C-n denotes the policy when the curriculum reaches n jobs.

As the training curriculum advances to larger scales, L2C develops broader construction capability (Figure 3). The final L2C-400 policy reduces L2C-10's panel-average gap by 51.1%. Importantly,

the improvement extends across all eight TAI sizes, covering both small and large instances in the panel. The largest gain occurs on 50×20, with a gap reduction of 9.67 percentage points.

E.2 Solution Quality under Wall-Clock Budgets

We compare the complete L2CI pipeline with CP-SAT at matched 30- and 60-minute budgets on four instance sizes: 100×40, 100×60, 150×50, and 200×50, with ten instances per size. We report mean makespan at each budget. L2CI uses one GPU, while CP-SAT uses eight CPU threads per instance.

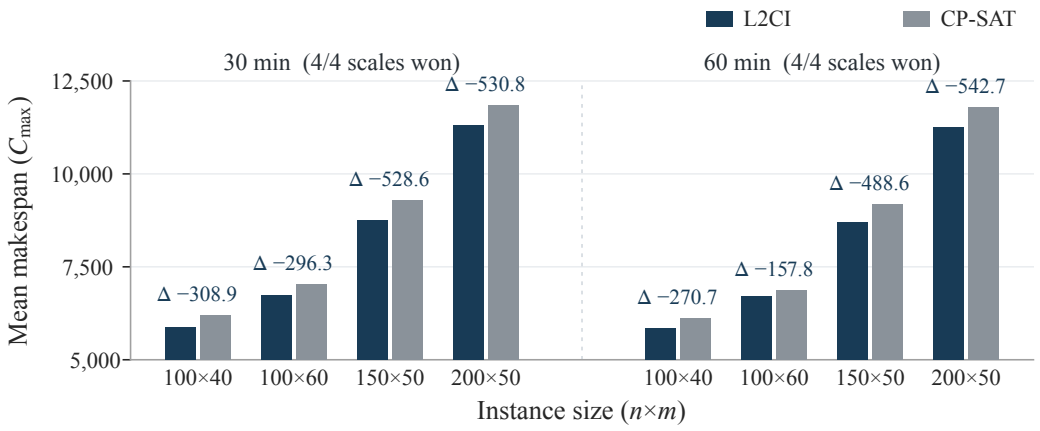


Figure 4: Mean makespan at equal 30- and 60-minute budgets on the four displayed sizes, with ten instances per size. Labels show $\Delta\bar{C}_{\max} = \bar{C}_{\max}^{\text{L2CI}} - \bar{C}_{\max}^{\text{CP-SAT}}$. The vertical axis starts at 5,000. Makespan uses instance processing-time units; minutes denote the wall-clock solving budget.

L2CI consistently outperforms CP-SAT in mean makespan on all four displayed sizes at both time budgets (Figure 4). At 30 minutes, it reduces mean makespan by 4.21–5.69%. More notably, its 30-minute mean makespans already beat CP-SAT's 60-minute results on every displayed size. Under the stated hardware configurations, L2CI thus delivers better schedules with half the time budget.

F LLM Use Statement

We used large language models (LLMs) as general-purpose assistants for drafting and polishing parts of the manuscript, providing feedback on methodological ideas, suggesting code snippets, and checking tabulated results. All substantive decisions regarding the research problem, methodology, and experimental design were made by us. We reviewed and revised the AI-assisted content and take full responsibility for the final manuscript.