Opening the Black Box: Topologically Guided Latent Steering for Neural Combinatorial Optimization

Henrik Abgaryan, Tristan Cazenave, and Ararat Harutyunyan

LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France
{henrik.abgaryan, tristan.cazenave, ararat.harutyunyan}@dauphine.psl.eu

Abstract. Neural Combinatorial Optimization (NCO) has seen significant advancements through methods like COMPASS, which condition pre-trained policies on a continuous latent space to enable inference-time search. However, existing approaches treat the neural policy as a black box, employing derivative-free optimization algorithms (e.g., CMA-ES) that ignore the structural properties of the generated solutions. We introduce **Topologically Guided Latent Steering (TGLS)**, an inference-time adaptation mechanism for the Job Shop Scheduling Problem (JSSP). TGLS integrates the topological structure of the solution, specifically the critical path, with the differentiability of the neural policy. By calculating the gradient of specific topological edits with respect to the latent code, TGLS actively "steers" the search toward superior regions of the policy space. We demonstrate that TGLS consistently improves over the COMPASS baseline under a fixed small budget on randomly generated JSSP instances, achieving tighter optimality gaps.

Keywords: Neural Combinatorial Optimization, Job Shop Scheduling, Latent Space Search, Policy Adaptation, Topological Gradient, Critical Path Method

1 Introduction

Combinatorial Optimization (CO) problems, such as the Job Shop Scheduling Problem (JSSP), are fundamental to industrial efficiency but remain NP-hard [6]. Recent "Constructive" Neural Combinatorial Optimization (NCO) approaches learn a policy π_θ to construct solutions sequentially [16,10,11]. While these methods offer fast inference, their zero-shot performance often lags behind exact solvers and industrial heuristics.

To bridge this gap, state-of-the-art methods like COMPASS [5] introduce *inference-time policy adaptation*. By conditioning the policy on a latent vector z and optimizing z via Covariance Matrix Adaptation Evolution Strategy (CMA-ES)[8], COMPASS finds diverse and high-quality solutions. However, this optimization process is fundamentally inefficient: it treats the mapping from z to solution quality as a black box, relying on random perturbations to stumble upon improvements.

In this work, we use **COMPASS+CMA-ES** as our baseline: at test time we repeatedly sample latent codes, decode schedules, and update the CMA-ES distribution using only the scalar objective value (makespan). Our proposed method **TGLS** differs in one key aspect: it augments the same latent-space search with a *topology-guided, gradient-based* update that biases the policy toward specific schedule edits.

We argue that the structure of the generated solution contains rich signals that are currently discarded. In JSSP, schedule quality is governed by the *Critical Path* that determines the makespan; consequently, meaningful local improvements often correspond to modifying operation orders inside "critical blocks" along that path [13].

We propose **Topologically Guided Latent Steering (TGLS)**. TGLS combines Operations Research (OR) insights with the differentiability of neural networks, but it does *not* directly apply a discrete swap operator in schedule space. Instead, TGLS heuristically identifies an operation pair (u, v) in a critical block, maps (u, v) to a decode step t where u's job was selected, and then steers the latent vector z to increase $\log \pi_\theta\big(a_t = \mathrm{job}(v) \mid s_t, z\big)$. We then perform gradient ascent in the latent space, effectively "steering" the generative policy toward schedules that are more likely to realize this swap-inspired intervention.

Our contributions are:

1. **Gray-Box Search:** We formulate a search strategy that utilizes both the scalar reward and the structural topology of the solution.
2. **Latent Steering:** We derive a gradient-based update rule for continuous latent codes that promotes discrete combinatorial changes.
3. **Inference-time Improvement over COMPASS:** We show that TGLS improves over the COMPASS baseline on JSSP benchmarks (10x10, 15x15, 20x15) in relative gap under our small inference-time budget.

2 Problem Definition: Job Shop Scheduling

A Job Shop Scheduling Problem (JSSP) instance is defined by a set of jobs $\mathcal{J} = \{1, \ldots, J\}$ and machines $\mathcal{M} = \{1, \ldots, M\}$. Each job $j \in \mathcal{J}$ consists of an ordered sequence of M operations $(j, 1), \ldots, (j, M)$. Operation (j, k) must be processed on a specified machine $m_{j,k} \in \mathcal{M}$ for a processing time $p_{j,k} > 0$.

A schedule assigns a start time $S_{j,k}$ to each operation subject to: (i) *precedence constraints* within each job,

$$S_{j,k+1} \geq S_{j,k} + p_{j,k}, \quad \forall j,\ k \in \{1, \ldots, M-1\}, \tag{1}$$

(ii) *machine capacity constraints* (each machine processes at most one operation at a time), and (iii) non-preemption.

The objective is to minimize the makespan

$$C_{\max} \ = \ \max_{j \in \mathcal{J}} \left(S_{j,M} + p_{j,M}\right). \tag{2}$$

3 Related Work

Classic JSSP heuristics and benchmarks. Much of the classical Operations Research literature represents JSSP solutions via disjunctive graphs and constructs schedules via priority-rule based procedures such as the Giffler–Thompson algorithm [7]. Standard benchmark sets (often referred to as Taillard instances) have become a common evaluation bedrock for comparing heuristic and learning-based solvers [15].

Neural Combinatorial Optimization for scheduling. Since the introduction of Pointer Networks [16], attention-based models [10] have become the standard for routing and sequencing tasks. For JSSP specifically, learning-to-dispatch approaches such as L2D [18] demonstrate that graph-based state representations can support strong scheduling policies. Other learning-based JSSP solvers combine GNN representations with reinforcement learning to learn transferable scheduling policies [14], and recent topology-aware GNN architectures explicitly exploit disjunctive-graph structure for improved performance [17]. Attention-based RL has also been applied to JSSP using modified Transformer backbones [12].

Inference-time search and per-instance adaptation. To improve zero-shot solutions, recent works employ inference-time search and adaptation strategies at test time. POMO [11] uses stochastic sampling with instance augmentations. EAS [9] employs active search by fine-tuning model weights via gradient descent on each instance. However, EAS is computationally expensive due to the high number of parameters updated. SchedulExpert combines graph attention with mixture-of-experts for JSSP [3].

Latent space search and structured neighborhoods. COMPASS [5] unifies these approaches by learning a latent space of diverse policies. It freezes the model weights and searches only the low-dimensional latent space using CMA-ES [8]. In parallel, classical local-search neighborhoods for JSSP (e.g., critical-block based move operators) remain highly effective and motivate hybrid approaches that incorporate structural information during search [13].

Recent work also explores the use of large language models for scheduling and combinatorial optimization: Abgaryan et al. investigate LLM-based approaches for JSSP [4]. STARJOB introduces a supervised dataset for LLM-driven JSSP [2]. ACCORD studies autoregressive, constraint-satisfying generation for combinatorial optimization [1].

3.1 Neural Policy and Latent Space

We utilize the Compass[5] framework, which trains an encoder-decoder modification of the pointer network. The policy $\pi_\theta(a_t|s_t, z)$ generates a schedule autoregressively, conditioned on the current state s_t and a latent variable $z \in \mathbb{R}^d$.

- **State** s_t: Current partial schedule, available machines, and remaining operations.

- **Action** a_t: A discrete job index $a_t \in \{1, \ldots, J\}$. Executing $a_t = j$ schedules the *next unscheduled operation* of job j (subject to an action mask enforcing feasibility) and updates the partial schedule.
- **Latent** z: A continuous vector ("behavior marker") that modulates the decoder's logits, allowing for diverse trajectory generation.

4 Methodology: Topologically Guided Latent Steering

TGLS follows a "Generate-Analyze-Steer" loop. It uses Covariance Matrix Adaptation Evolution Strategy (CMA-ES)[8] to explore the latent space z, but augments it with a gradient-based correction derived from topological analysis. If the **Analyze** and **Steer** steps are removed, the procedure reduces to the standard COMPASS inference-time latent search (CMA-ES over z with frozen network weights) [5]. Let $z \sim \mathcal{N}(\mu, \Sigma)$ be a candidate latent vector. The policy generates a trajectory $\tau = \{(s_0, a_0), \ldots, (s_T, a_T)\}$ with makespan $C_{\max}(\tau)$. TGLS operates in generations. In each generation g:

1. **Sample**: Generate a population $Z_g = \{z_1, \ldots, z_\lambda\}$ via CMA-ES.
2. **Evaluate**: Decode each z_i to obtain trajectories and makespans. Identify the elite candidate z_{best}.
3. **Analyze**: From the decoded schedule of z_{best}, identify an adjacent operation pair (u, v) on a critical block (a swap-inspired intervention).
4. **Steer**: TGLS does not execute this swap directly; instead it maps (u, v) to a timestep t where u's job was selected and computes z_{steer} to increase $\log \pi_\theta\big(a_t = \text{job}(v) \mid s_t, z\big)$.
5. **Update**: Update the CMA-ES distribution using $Z_g \cup \{z_{\text{steer}}\}$.

Given a schedule, we compute the critical path. A *critical block* is a maximal sequence of operations on the same machine belonging to the critical path. Motivated by classic critical-block neighborhoods in the JSSP literature [13], we heuristically select an adjacent critical-block pair and steer the policy toward choosing the job corresponding to the second operation when precedence readiness suggests feasibility. We identify a target pair (u, v) such that u and v are adjacent in a critical block (currently scheduled $u \to v$). We then map this pair to a target action a_{target} at the timestep t where u was originally scheduled: "select the job of v instead of the job of u".

4.1 Latent Steering via Direct Gradient

We do not simply force action a_{target}, as this naturally degrades the autoregressive policy's performance on subsequent steps (forcing it off-distribution). Instead, we modify the latent context z to make the policy *want* to take the action. We define the steering loss as the log-likelihood of the target action:

$$\mathcal{L}_{\text{steer}}(z) = \log \pi_\theta(a_{\text{target}} | s_t, z)$$

We compute the gradient $\nabla_z \mathcal{L}_{\text{steer}}(z)$ via backpropagation through the frozen policy network. The steered latent vector is obtained by gradient ascent:

$$z_{\text{steer}} = z_{\text{best}} + \alpha \cdot \nabla_z \mathcal{L}_{\text{steer}}(z_{\text{best}})$$

where α is a hyperparameter (Steering Coefficient). This new candidate z_{steer} is then evaluated and included in the population for the CMA-ES update.

Algorithm 1: TGLS Step (Single Generation)

Input: CMA-ES parameters (μ, Σ), Policy π_θ
Sample population $z_1, \ldots, z_\lambda \sim \mathcal{N}(\mu, \Sigma)$
$\tau_i \leftarrow \text{Rollout}(\pi_\theta, z_i)$ for $i = 1 \ldots \lambda$
$z_{\text{best}} \leftarrow \arg\min_{z_i} C_{\max}(\tau_i)$
$\text{Swap}(t, a_{\text{target}}) \leftarrow \text{AnalyzeCriticalPath}(\tau_{\text{best}})$
if Swap found **then**
    $g \leftarrow \nabla_z \log \pi_\theta(a_{\text{target}} | s_t(\tau_{\text{best}}), z_{\text{best}})$
    $z_{\text{steer}} \leftarrow z_{\text{best}} + \alpha \cdot g$
    $\tau_{\text{steer}} \leftarrow \text{Rollout}(\pi_\theta, z_{\text{steer}})$
    Add $(z_{\text{steer}}, C_{\max}(\tau_{\text{steer}}))$ to population
end if
Update (μ, Σ) using best candidates

5 Experimental Setup

5.1 Datasets, objective, and evaluation metric

We evaluate on standard Taillard-like JSSP validation sets of sizes 10×10, 15×15, and 20×15. For each instance, the policy rollout induces a schedule with makespan $C_{\max}$. We report the **optimality gap (%)** with respect to a fixed reference makespan C_{ref} (taken from the OR-Tools benchmark values used in our experimental scripts):

$$\text{Gap} = \frac{C_{\max} - C_{\text{ref}}}{C_{\text{ref}}} \times 100.$$

5.2 Inference-time search budget

All methods are evaluated under a small inference-time rollout budget per instance. The CMA-ES baseline uses population size $\lambda = 4$ and runs for 8 generations, resulting in $4 \times 8 = 32$ policy rollouts per instance. TGLS uses the same population size and, in addition, may evaluate one extra *steered* candidate per generation when a valid critical-block intervention is identified. Due to the integer population size and the termination condition in our current implementation, **TGLS may use up to 34 rollouts** (population rollouts plus the occasional steered rollout).

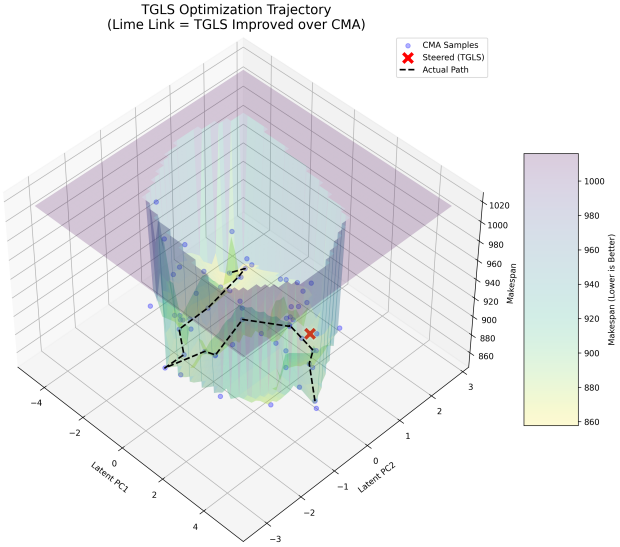


Fig. 1: Latent-space visualization of inference-time search. We collect CMA-ES population samples (blue circles) and gradient-steered candidates produced by TGLS (red crosses) across iterations, project the latent codes to two principal components (PC1/PC2), and plot the corresponding rollout makespan as height (lower is better). The translucent surface is an interpolation from these sampled points and is shown only as a visual aid (it does not represent a true latent energy landscape). Green links highlight iterations where the steered candidate improves over the best CMA sample, illustrating how topology-guided steering can induce directed jumps between sampled regions.

5.3 Implementation details for reproducibility

Latent search hyperparameters. Unless stated otherwise, we use population size $\lambda = 4$, steering coefficient $\alpha = 1.5$, and CMA-ES latent initialization with standard deviation $\sigma = 0.5$ and box constraints $z \in [-1, 1]^d$.

Episode horizon and truncation. Rollouts are executed with a fixed episode horizon that depends on the instance size: $T = 1250$ for 10x10, $T = 3000$ for 15x15, and $T = 4000$ for 20x15. This horizon acts as an explicit truncation limit and is held constant across all methods.

Hardware configuration. All experiments are run on CPU-only JAX execution. For reproducibility, the provided scripts explicitly disable GPU usage (e.g., by setting `CUDA_VISIBLE_DEVICES=""` and configuring JAX to run on CPU).

Random seeds. We use fixed per-instance random seeds (e.g., $seed = instance\_idx \times 1000$) to make the search procedure deterministic given the codebase.

Decoding protocol and fairness. In our implementation, the COMPASS/CMA-ES baseline evaluates each latent code using *deterministic (greedy) decoding* (taking the mode action at each timestep), whereas TGLS evaluates population candidates using *stochastic decoding* (sampling actions from the policy distribution). We keep this distinction because (i) the baseline is intended to measure the best performance achievable by latent search under a stable, low-variance evaluation function, and (ii) TGLS relies on trajectory diversity to discover informative critical-block edits and to produce a meaningful steering gradient. Overall, both methods use the same frozen policy network, the same latent search space and initialization, and the same population size, so the comparison *primarily* isolates the effect of topology-guided steering; nevertheless, we note that the decoding stochasticity differs and affects objective noise and exploration, and we retain the implementation-faithful protocol used in our scripts.

6 Results

6.1 Latent-space visualization

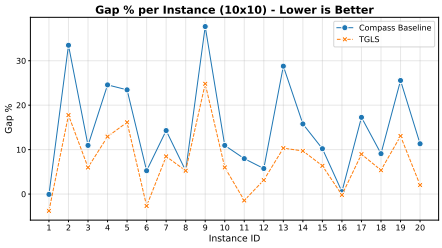
Figure 1 provides a qualitative view of how TGLS navigates the COMPASS latent space during inference-time search. We record all sampled latent codes (CMA-ES population samples) and the additional gradient-steered candidates proposed by TGLS, then project the resulting set of latent vectors to two dimensions using PCA (PC1/PC2). Each point is plotted with height equal to the rollout objective (makespan; lower is better). This visualization can be interpreted as an empirical slice of the latent "energy landscape": CMA-ES explores locally via stochastic sampling, while TGLS adds directed steps that aim to increase the probability of a specific discrete scheduling decision (a job-selection action corresponding to a critical-block intervention). When a steered candidate improves upon the best CMA sample in that iteration, the plot highlights this jump (green link), illustrating how gradient information can move the search to a lower-makespan region that is unlikely to be reached by undirected sampling at the same budget.

6.2 Sample efficiency, optimality gap, and runtime

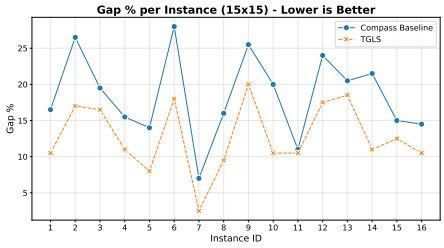
TGLS demonstrates superior sample efficiency. By injecting gradient-derived candidates, TGLS achieves lower optimality gaps within the same fixed budget compared to the baseline's blind evolution. Table 1 summarizes aggregate performance, while Figure 2 shows per-instance gaps and Figure 3 reports per-instance wall-clock time on CPU.

Table 1: Comparison of Mean Optimality Gap (%) on JSSP Benchmarks under a small inference-time rollout budget (baseline: `BUDGET` = 32 rollouts per instance; TGLS: up to 34 rollouts in our current implementation due to the additional steered rollout).

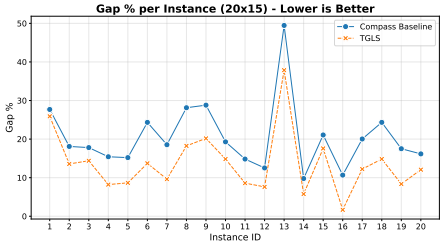
Instance Size	COMPASS Baseline	TGLS (Ours)	Relative Improv.
10x10	14.36%	**7.05%**	+50.9%
15x15	18.19%	**12.06%**	+33.7%
20x15	19.93%	**13.43%**	+32.6%



(a) 10x10

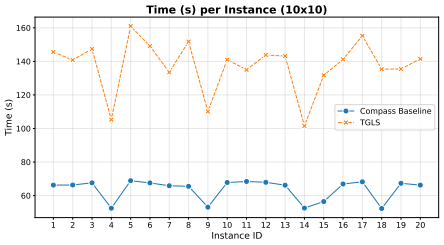


(b) 15x15

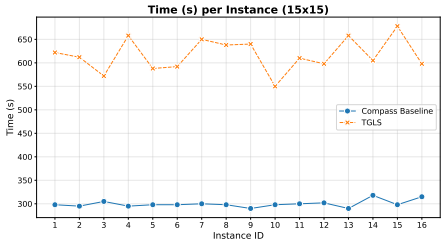


(c) 20x15

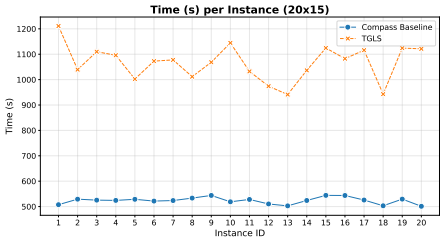
Fig. 2: Per-instance optimality gap comparison between the COMPASS/CMA-ES baseline and TGLS at the end of the inference-time search. Each point corresponds to one validation instance (instances are indexed consistently by hash), and lower gap is better.



(a) 10x10



(b) 15x15



(c) 20x15

Fig. 3: Per-instance wall-clock time comparison (CPU execution) for the COMPASS/CMA-ES baseline and TGLS. Times correspond to the total search time per instance as recorded by the experiment script.

7 Conclusion, Limitations and Future work

The success of TGLS highlights the limitations of treating neural policies as black boxes during inference. While latent spaces provide a powerful manifold for search, navigating them blindly is inefficient. TGLS proves that the **generative process contains differentiable structural information** that can be leveraged. By steering the latent code to maximize the probability of critical path swaps, we effectively perform "local search in latent space." A limitation is the dependency on the policy's conditioning. If the policy ignores the latent z (posterior collapse) or if the landscape is highly non-convex, the gradient step may not yield the desired action change. Furthermore, the topological analysis assumes the Critical Path method, which is standard for JSSP but may require adaptation for other domains.

References

1. Abgaryan, H., Cazenave, T., Harutyunyan, A.: Accord: Autoregressive constraint-satisfying generation for combinatorial optimization with routing and dynamic attention. arXiv preprint arXiv:2506.11052 (2025)
2. Abgaryan, H., Cazenave, T., Harutyunyan, A.: Starjob: Dataset for llm-driven job shop scheduling. arXiv preprint arXiv:2503.01877 (2025)
3. Abgaryan, H., Cazenave, T., Harutyunyan, A.: Schedulexpert: Graph attention meets mixture-of-experts for jssp. Lecture Notes in Computer Science, vol. 15744, pp. 281–297 (2026). https://doi.org/10.1007/978-3-032-09156-7_19
4. Abgaryan, H., Harutyunyan, A., Cazenave, T.: Llms can schedule. arXiv preprint arXiv:2408.06993 (2024)
5. Chalumeau, F., Bonnet, C., Pretorius, A., Surana, S., Grinsztajn, N., Laterre, A., Barrett, T.D.: Combinatorial optimization with policy adaptation using latent space search. In: Advances in Neural Information Processing Systems. vol. 36 (2023)
6. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman (1979)
7. Giffler, B., Thompson, G.L.: Algorithms for solving production-scheduling problems. Operations Research **8**(4), 487–503 (1960)
8. Hansen, N.: The CMA evolution strategy: A tutorial. arXiv preprint arXiv:1604.00772 (2016)
9. Hottung, A., Kwon, Y.D., Tierney, K.: Efficient active search for combinatorial optimization problems. In: International Conference on Learning Representations (2022)
10. Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (2019)
11. Kwon, Y.D., Choo, J., Kim, B., Yoon, I., Gwon, Y., Min, S.: Pomo: Policy optimization with multiple optima for reinforcement learning. In: Advances in Neural Information Processing Systems. vol. 33, pp. 21188–21198 (2020)
12. Lee, J., Kee, S., Janakiram, M., Runger, G.: Attention-based reinforcement learning for combinatorial optimization: Application to job shop scheduling problem. arXiv preprint arXiv:2401.16580 (2024)
13. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the job shop problem. Management Science **42**(6), 797–813 (1996)
14. Park, J., Chun, J., Kim, S.H., Kim, Y., Park, J.: Learning to schedule job-shop problems: Representation and policy learning using graph neural network and reinforcement learning. arXiv preprint arXiv:2106.01086 (2021)
15. Taillard, É.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993)
16. Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Advances in Neural Information Processing Systems. vol. 28 (2015)
17. Zhang, C., Cao, Z., Wu, Y., Song, W., Sun, J.: Learning topological representations with bidirectional graph attention network for solving job shop scheduling problem. In: Kiyavash, N., Mooij, J.M. (eds.) Proceedings of the Fortieth Conference on Uncertainty in Artificial Intelligence. Proceedings of Machine Learning Research, vol. 244, pp. 4192–4208. PMLR (15–19 Jul 2024)
18. Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P.S., Xu, C.: Learning to dispatch for job shop scheduling via deep reinforcement learning. In: Advances in Neural Information Processing Systems. vol. 33, pp. 1621–1632 (2020)