JUST INITIALIZE: A TRAINING-FREE INITIALIZATION COMPONENT FOR LARGE-SCALE ROUTING OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale routing problems are difficult to solve efficiently as their search spaces grow rapidly with problem size. Existing approaches primarily improve the optimization procedure itself, often at increasing computational cost. We instead shift the focus to a useful initialization that can be refined into a high-quality solution with limited downstream refinement. We propose Just Initialize, a training-free and solver-agnostic initialization component for large-scale routing optimization. Just Initialize compresses a large routing instance into a compact surrogate space, optimizes its global routing structure, and recovers the resulting solution as an optimization-friendly starting point in the original space. Extensive experiments on Traveling Salesman Problems (TSPs), Capacitated Vehicle Routing Problems (CVRPs), Vehicle Routing Problems with Time Windows (VRPTWs), and Prize-Collecting Traveling Salesman Problems (PCTSPs) demonstrate that Just Initialize achieves high-quality solutions comparable to or better than state-of-the-art methods while substantially reducing computational cost across instances ranging from 1K to 100K nodes, including an average speedup of approximately 70×, sub-second runtimes on 10K-node instances, and runtimes within tens of seconds on 100K-node instances.

1 INTRODUCTION

Routing problems are widely encountered in real-world systems such as transportation, autonomous delivery, and industrial workshops (Golden et al., 2023; Elshaer & Awad, 2020). However, their NP-hard nature makes solving large-scale instances computationally expensive and time-consuming. Traditional optimization methods have achieved remarkable success in obtaining optimal or near-optimal solutions. Deterministic algorithms, such as mixed integer programming algorithms, provide strong optimality guarantees but suffer from prohibitive computational costs as problem scales increase. Heuristics and metaheuristics improve search scalability but often rely on iterative evolution and evaluation (Yang et al., 2021; Wang & Tang, 2021). These limitations significantly hinder their applicability to large-scale real-world scenarios, where efficient decision making is increasingly required.

Recently, learning-based methods have emerged as promising alternatives to handcrafted routing solvers. Neural combinatorial optimization (NCO) uses neural networks to learn solution construction or improvement strategies through various training paradigms. Among these, reinforcement learning (RL) trains routing policies through reward-driven interactions with problem environments (Kwon et al., 2020; Kim et al., 2021). Although NCO methods enable efficient inference on unseen instances (Bi et al., 2022; Luo et al., 2023), scaling them to large routing problems remains challenging. Recent studies address this challenge through enhanced neural solvers, hierarchical decomposition, search space reduction, and neural-guided optimization (Ye et al., 2024; Li et al., 2021; Qiu et al., 2022). Despite substantial progress, these approaches primarily improve the solving process itself, often introducing increasingly sophisticated architectures, training procedures, or search mechanisms. This motivates a complementary perspective: improving where optimization begins.

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.

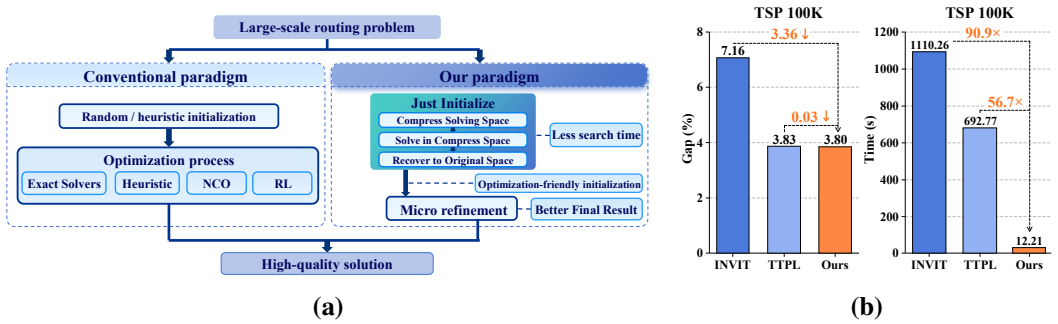

Figure 1: Rethinking large-scale routing optimization. (a) Conventional paradigm and our paradigm. (b) Performance comparison with SOTA methods on 100K-node TSP instances.

Rather than designing more sophisticated optimization procedures, we ask whether the computational burden can be reduced by improving where optimization starts. A good initialization should therefore be judged not only by its immediate solution quality, but also by how efficiently it can be refined into a high-quality solution. A solution with a better initial objective value may require substantial further search, while one with a worse initial value may reach a better result through limited refinement. This perspective shifts the goal of initialization toward identifying starting points with strong refinement potential. Since such starting points are primarily characterized by their global structure rather than fully resolved local decisions, compression provides a natural way to reduce the search space while retaining the information most relevant to subsequent refinement.

Previous studies have explored compression and coarsening to reduce problem size before recovering solutions in the original space (Orloff & Caprera, 1976; Nałęcz-Charkiewicz et al., 2025). When recovery is expected to produce a high-quality solution directly, the compressed representation must preserve both global structure and local details, limiting how aggressively the problem can be compressed.

Therefore, we propose Just Initialize, a training-free and solver-agnostic initialization component for large-scale routing optimization. Rather than directly searching for a final solution in the original space, Just Initialize compresses a large routing instance into a compact surrogate space, where promising global structures can be identified efficiently. The resulting compact solution is then recovered as an optimization-friendly starting point in the original space, leaving fine-grained decisions to downstream refinement. In this way, Just Initialize focuses on discovering where optimization should begin rather than resolving the entire solution from scratch.

Our contributions are summarized as follows:

- We introduce **Just Initialize**, a training-free and solver-independent initialization component that discovers optimization-friendly starting points for large-scale routing problems by separating solution discovery from solution optimization.
- We develop a geometry-guided instance compression procedure to construct compact routing representations, and propose an initialization-oriented objective to evaluate compact solutions according to their downstream optimization potential.
- Extensive experiments on TSPs, CVRPs, VRPTWs, and PCTSPs demonstrate that Just Initialize achieves competitive or superior solution quality compared with state-of-the-art methods, while substantially reducing computational cost and scaling to instances from 1K to 100K nodes.

2 Related Works

2.1 Traditional Optimization Methods

Traditional routing problems are commonly addressed by exact algorithms, heuristics, and metaheuristics. Exact methods provide strong optimality guarantees but become increasingly expen-

sive as problem size grows (Gamst et al., 2024; Aerts-Veenstra et al., 2024). For large-scale instances, heuristic and metaheuristic solvers instead rely on carefully designed neighborhoods and iterative search, including Lin–Kernighan–Helsgaun (Helsgaun, 2000), adaptive large neighborhood search (Ropke & Pisinger, 2006), hybrid genetic search (Vidal, 2022; Simensen et al., 2022), and specialized local-search procedures (Cook et al., 2024). Recent methods further demonstrate that carefully engineered search can scale to extremely large routing instances (Accorsi & Vigo, 2024). Despite their strong performance, obtaining high-quality solutions typically still requires substantial search as problem size increases.

2.2 Neural Combinatorial Optimization

Neural combinatorial optimization (NCO) learns solution construction or improvement strategies directly from problem instances (Bengio et al., 2021). Reinforcement learning is widely used for training such policies, with representative methods including Attention Model (Kool et al., 2019), POMO (Kwon et al., 2020), collaborative policies (Kim et al., 2021), knowledge distillation (Bi et al., 2022), and symmetry-aware learning (Kim et al., 2022). These methods reduce dependence on manually designed search rules and enable efficient inference after training.

Recent work has increasingly focused on scaling NCO to large routing instances. Existing approaches improve model architectures and training strategies (Luo et al., 2023; Fang et al., 2024; Luo et al., 2025a; Chen et al., 2025; Luo et al., 2025b), decompose large problems into smaller subproblems (Li et al., 2021; Pan et al., 2023; Ye et al., 2024; Zheng et al., 2024), learn alternative solution representations (Qiu et al., 2022; Sun & Yang, 2023), or combine neural predictions with classical optimization procedures (Xin et al., 2021; Kool et al., 2022; Ye et al., 2023). These methods improve scalability from different perspectives, but primarily focus on how solutions are constructed, represented, or optimized during the search process.

2.3 Problem Reduction and Compression

Problem reduction provides another way to improve routing scalability by reducing the number of decisions exposed to the solver. Classical approaches include cluster-first and route-first strategies (Gillett & Miller, 1974; Fisher & Jaikumar, 1981; Beasley, 1983; Prins et al., 2014), explicit problem reduction (Orloff & Caprera, 1976), restricted neighborhood search (Toth & Vigo, 2003), and subproblem-based optimization (Queiroga et al., 2021). More recent studies extend this idea through large-scale decomposition (Kerscher & Minner, 2025) and graph coarsening, where a smaller routing representation is constructed and subsequently expanded to the original problem (Nałęcz-Charkiewicz et al., 2025).

Different from existing reduction methods that aim to preserve sufficient information for solution recovery, we use compression to identify optimization-friendly starting points, while leaving detailed routing decisions to subsequent optimization.

3 Preliminaries

3.1 Routing Problems and Solution Spaces

Let $\mathcal{I}$ be the space of routing instances. Each instance $I = (G, X, K)$ defines a feasible solution space $\Omega(I)$ and an objective function C_I. To provide a unified representation across different routing variants, we represent a multi-route solution as a single visiting sequence with route boundaries. Specifically, a solution with routes $(r_1, \ldots, r_m)$ is written as

$$\pi = (\sigma, \mathbf{b}), \qquad \sigma = r_1 \oplus \cdots \oplus r_m, \qquad b_j = \sum_{\ell=1}^{j} |r_\ell|, \quad j = 1, \ldots, m, \tag{1}$$

where σ denotes the visiting sequence and $\mathbf{b}$ specifies the breakpoints that partition the sequence into individual routes. Given $\mathbf{b}$, each route can be recovered from the corresponding segment of σ. This representation is equivalent to the original multi-route formulation, while allowing different routing problems to share a common solution representation. For example, a solution to either TSP

or PCTSP contains a single segment, whereas CVRP and VRPTW impose additional constraints on route segments and visited nodes.

The optimization objective is defined as

$$\pi^* \in \arg\min_{\pi \in \Omega(I)} C_I(\pi). \tag{2}$$

For later refinement, we define a local neighborhood over the original solution space. A neighboring solution is generated by modifying either the visiting sequence or the route structure while preserving feasibility:

$$N_I(\pi) = \left\{ (T_{i,j}(\sigma), \mathbf{b}) : T_{i,j} \in \{\mathrm{swap}_{i,j}, \mathrm{rev}_{i:j}\}, i < j \right\} \cap \Omega(I), \tag{3}$$

where $\mathrm{swap}_{i,j}$ exchanges two positions in the visiting sequence and $\mathrm{rev}_{i:j}$ reverses the segment between positions i and j. This neighborhood defines the local modifications used by downstream refinement procedures.

3.2 Compact Representations and Solution Space Mapping

Given an original routing instance I, we first construct a compact instance by grouping original nodes into a small number of compact units.

$$\tilde{I} = \Phi_c(I). \tag{4}$$

The compact instance $\tilde{I}$ induces a lower-dimensional solution space $\Omega(\tilde{I})$, where each compact unit represents a group of original nodes. This correspondence defines a projection from the original solution space to the compact solution space:

$$\tilde{\pi} = \Phi_s(\pi; M), \qquad \pi \in \Omega(I), \quad \tilde{\pi} \in \Omega(\tilde{I}). \tag{5}$$

Here, M records the correspondence between original nodes and compact units established during instance compression. The projection $\Phi_s(\cdot)$ does not directly compress routing decisions. Instead, it preserves the coarse structural information induced by instance compression while omitting fine-grained decisions. Therefore, the projection from the original solution space to the compact solution space is generally many-to-one, where multiple original solutions may correspond to the same compact solution:

$$\Phi_s^{-1}(\tilde{\pi}; M) = \{\pi \in \Omega(I) | \Phi_s(\pi; M) = \tilde{\pi}\}. \tag{6}$$

3.3 Initialization Quality Measurement

The quality of an initialization is determined by how effectively it enables subsequent optimization, rather than solely by its immediate objective value. A feasible solution $\pi_0 \in \Omega(I)$ is regarded as an initialization when it is provided as the starting point for downstream optimization. Let $\mathcal{R}_I(\pi_0; B)$ denote the solution obtained by applying a downstream solver initialized at π_0 with computational budget B. The budget can be measured in runtime, search iterations, or objective evaluations. We first evaluate an initialization by the expected objective after optimization:

$$J_B(\pi_0; I, \mathcal{R}) = \mathbb{E}\left[C_I(\mathcal{R}_I(\pi_0; B))\right], \tag{7}$$

where the expectation accounts for randomness in stochastic solvers and can be omitted for deterministic procedures. Because C_I is a cost to be minimized, a lower J_B indicates a more effective initialization, and initializations are compared by minimizing J_B under a fixed solver and budget.

4 Methodology

4.1 Overview of Just Initialize

Building on the formulation in Section 3, Just Initialize constructs an optimization-friendly initialization through three stages: instance compression, compact-space optimization, and solution recovery.

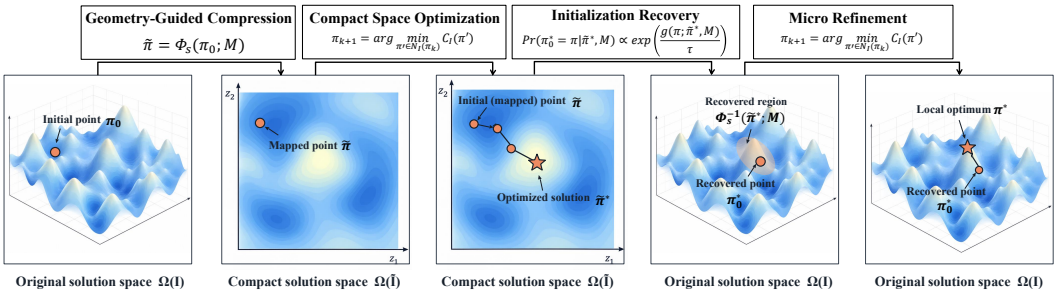

Figure 2: Overview of Just Initialize. The framework compresses the original solution space, optimizes in a compact space, recovers an optimization-friendly initialization, and performs lightweight refinement in the original space.

The central idea is to separate solution discovery from solution optimization. Instead of directly searching in the original high-dimensional solution space, Just Initialize first compresses the problem into a compact space, where coarse routing structures can be efficiently explored. The obtained compact solution is then recovered into the original space as an optimization-friendly initialization and further improved through downstream refinement. The following sections describe each component in detail.

4.2 Geometry-Guided Instance Compression for Solution Space Reduction

The first stage constructs a compact representation of the original routing instance. Different from conventional compression methods that aim to recover a high-quality solution directly, our goal is to identify optimization-friendly regions while leaving fine-grained decisions to subsequent refinement.

Given an original instance I, we construct a compact instance by grouping nodes into compact units: $\tilde{I} = \Phi_c(I)$. The compact instance induces a reduced solution space $\Omega(\tilde{I})$, where solutions are defined over compact units instead of individual nodes. Accordingly, the original solution space is projected through

$$\tilde{\pi} = \Phi_s(\pi; M), \quad \pi \in \Omega(I), \tilde{\pi} \in \Omega(\tilde{I}). \tag{8}$$

Since multiple original solutions may share the same compact structure, Φ_s is generally many-to-one. The compact representation preserves coarse geometric structures and boundary information required for recovery, while omitting unresolved local decisions. The detailed construction procedure is summarized in Algorithm 1.

Algorithm 1: Geometry-Guided Instance Compression for Solution Space Reduction

Input: Routing instance $I = (G, X, K)$; target unit size b

Output: Compact instance $\tilde{I}$, correspondence information M, and solution projection Φ_s.

$F \leftarrow \textsc{ExtractGeometry}(G, X)$
$N \leftarrow \textsc{BuildNeighborhood}(V, F)$
$C \leftarrow \textsc{ConstructCompactUnits}(V, N, F, b, K)$
foreach $C_i \in C$ **do**
  $z_i \leftarrow \textsc{SummarizeGeometry}(C_i, F)$;
  $\mathcal{T}_i \leftarrow \textsc{ExtractBoundaryStates}(C_i, F, K)$;
  $M_i \leftarrow (C_i, z_i, \mathcal{T}_i, F|_{C_i})$;
$\tilde{C} \leftarrow \textsc{EstimateConnections}(C, \{z_i, \mathcal{T}_i\}, I)$
$\tilde{K} \leftarrow \textsc{AggregateConstraints}(C, K)$
$\tilde{I} \leftarrow (C, \{z_i, \mathcal{T}_i\}, \tilde{C}, \tilde{K})$
$\Phi_s \leftarrow \textsc{ConstructSolutionProjection}(C, M)$
return $(\tilde{I}, M, \Phi_s)$

4.3 SOLVING IN THE COMPACT SPACE

After compression, the routing problem is optimized in the compact solution space $\Omega(\tilde{I})$, where each solution represents the ordering of compact units and their corresponding local states. Following the initialization quality criterion defined in Section 3.3, we evaluate a compact solution according to the downstream performance of its recovered initialization.

Since J_B is defined as an expectation over downstream optimization processes, its exact value is generally unavailable. We therefore estimate it using Monte Carlo sampling with a small number of refinement trials:

$$\hat{J}_B(\tilde{\pi}) = \frac{1}{S}\sum_{s=1}^{S} C_I(R_I^{(s)}(\pi_0; B)), \tag{9}$$

where S denotes the number of sampled refinement trajectories and $R_I^{(s)}$ represents the s-th refinement process.

The compact-space search then aims to identify solutions with better initialization potential. Starting from an initial compact solution, we iteratively update the solution by minimizing $\hat{J}_B$, the estimated initialization quality. Similar to the neighborhood operators defined in the original solution space, the compact-space neighborhood $N(\tilde{\pi})$ is constructed by applying local modifications to compact-unit ordering while preserving compact feasibility:

$$\tilde{\pi}_{k+1} = \arg\min_{\tilde{\pi}' \in N(\tilde{\pi}_k)} \hat{J}_B(\tilde{\pi}'). \tag{10}$$

The final compact solution $\tilde{\pi}^*$ captures a global routing structure with strong refinement potential and is subsequently recovered to generate an optimization-friendly initialization in the original solution space.

4.4 RECOVERING OPTIMIZATION-FRIENDLY INITIALIZATIONS

Given the final compact solution $\tilde{\pi}^*$, recovery selects an original-space initialization from the feasible region $\Phi_s^{-1}(\tilde{\pi}^*; M)$ induced by the solution-space mapping defined in Section 3.2. The structural information M extracted during compression provides guidance for evaluating the compatibility between candidate solutions and the compact representation. Specifically, we define a structural consistency score $g(\pi; \tilde{\pi}^*, M)$ to measure how well an original-space solution preserves the compact-unit ordering, geometric structure, and boundary configurations specified by $\tilde{\pi}^*$ and M. The detailed formulation of $g(\pi; \tilde{\pi}^*, M)$ is provided in Appendix B. The initialization is selected according to:

$$\Pr(\pi_0^* = \pi \mid \tilde{\pi}^*, M) = \frac{\exp(g(\pi; \tilde{\pi}^*, M)/\tau)}{\sum_{\pi' \in \Phi_s^{-1}(\tilde{\pi}^*; M)} \exp(g(\pi'; \tilde{\pi}^*, M)/\tau)}, \quad \pi \in \Phi_s^{-1}(\tilde{\pi}^*; M). \tag{11}$$

Here, τ controls the trade-off between selecting highly compatible candidates and maintaining diversity. The resulting optimization friendly initialization π_0^* is then passed to downstream refinement in the original solution space.

4.5 MICRO REFINEMENT

The π_0^* serves as an optimization-friendly initialization rather than a final solution. We therefore perform a lightweight local refinement in the original solution space using the neighborhood operators defined in Section 3.1.

At each iteration, we select the best improving neighbor:

$$\pi_{k+1} = \arg\min_{\pi' \in N_I(\pi_k)} C_I(\pi'), \quad C_I(\pi_{k+1}) < C_I(\pi_k).$$

If no improving neighbor exists, the procedure terminates. Since the initialization already captures the global routing structure, this lightweight refinement only needs to correct remaining local decisions without rebuilding the solution.

Table 1: Comparison results on large-scale TSP and CVRP instances. Obj. and Gap denote the solution objective and relative gap, respectively. Bold marks the lowest Gap and Time at each scale before rounding; gray highlights our method. OOM: results that exceed GPU memory limits. OOT: the method fails to finish within the predefined time limit.

Method	TSP5K			TSP10K			TSP20K			TSP50K			TSP100K		
	Obj.↓	Gap	Time	Obj.↓	Gap	Time	Obj.↓	Gap	Time	Obj.↓	Gap	Time	Obj.↓	Gap	Time
LKH-3 (Reference)	50.88	–	–	71.76	–	–	101.39	–	–	159.98	–	–	225.95	–	–
POMO (no aug.)	88.33	73.62	50.6 s	133.64	86.23	6.5 min		OOM			OOM			OOM	
POMO (×8 aug.)	87.62	72.23	7.7 min		OOM			OOM			OOM			OOM	
LEHD (greedy)	59.50	16.94	1.2 min	90.26	25.77	7.9 min		OOM			OOM			OOM	
INViT	54.22	6.57	46.1 s	76.76	6.97	1.5 min	108.57	7.08	3.1 min	171.42	7.15	8.3 min	242.13	7.16	18.5 min
SIL	57.23	12.49	1.4 min	93.50	30.29	10.5 min	179.04	76.59	1.6 h		OOM			OOM	
L2C-Insert ($I = 1000$)	51.73	**1.68**	57.1 s	74.30	3.54	58.4 s	105.15	3.71	1.0 min	166.13	3.84	1.2 min	237.29	5.02	1.6 min
DIFUSCO	52.48	3.15	26.0 s	73.97	**3.07**	1.3 min	105.35	3.91	4.5 min	166.03	3.78	29.7 min		OOM	
GLOP	54.12	6.38	1.5 s	76.59	6.73	1.8 s	108.25	6.76	3.3 s	170.91	6.83	17.0 s	241.53	6.90	1.2 min
UDC	52.82	3.83	3.5 s	74.69	4.08	3.8 s	105.66	4.21	5.2 s	166.87	4.30	18.7 s	235.84	4.38	1.2 min
H-TSP	55.07	8.24	5.1 s	77.85	8.48	9.3 s	110.18	8.67	18.8 s	173.89	8.69	47.3 s	245.50	8.65	1.7 min
TTPL	52.81	3.79	33.6 s	74.75	4.17	1.1 min	105.51	4.06	2.3 min	166.31	3.95	5.8 min	234.61	3.83	11.5 min
JI+VND	53.35	4.87	15.3 s	75.24	4.85	28.9 s	106.63	5.16	31.5 s	168.14	5.10	1.7 min	238.92	5.74	2.7 min
JI+Refine (Ours)	52.56	3.30	**0.24 s**	74.20	3.40	**0.53 s**	105.00	**3.56**	**1.47 s**	165.77	**3.62**	**4.83 s**	234.20	**3.65**	**12.21 s**

Method	CVRP5K			CVRP10K			CVRP20K			CVRP50K			CVRP100K		
	Obj.↓	Gap	Time	Obj.↓	Gap	Time	Obj.↓	Gap	Time	Obj.↓	Gap	Time	Obj.↓	Gap	Time
LKH-3 (Reference)	86.00	–	–	97.97	–	–	125.10	–	–	236.46	–	–	409.25	–	–
OR-Tools	108.10	25.69	2.0 min	131.90	34.64	2.0 min		OOM			OOM			OOM	
LEHD	104.51	21.52	1.4 min		OOM			OOM			OOM			OOM	
INViT	104.53	21.55	1.8 min	130.29	32.99	3.8 min	183.17	46.42	11.4 min	355.78	50.46	57.4 min	604.34	47.67	3.6 h
SIL	103.24	20.05	2.6 min	127.60	30.24	5.4 min	191.14	52.79	11.4 min		OOM			OOM	
L2C-Insert	114.46	33.08	59.7 s	148.95	52.04	2.0 min	216.71	73.24	4.0 min	409.53	73.19	10.0 min	680.13	66.19	20.1 min
GLOP	122.28	42.19	3.1 s	129.20	31.88	10.2 s	174.14	39.20	27.3 s		OOM			OOM	
UDC	155.31	80.59	14.9 s	361.79	269.29	17.2 s	534.78	327.48	27.5 s		OOM			OOM	
TTPL	101.81	18.38	44.6 s	127.61	30.25	1.5 min	179.09	43.16	3.0 min	345.87	46.27	7.5 min	606.71	48.25	15.1 min
JI+Refine (Ours)	99.32	**15.56**	**0.43 s**	117.31	**19.80**	**0.9 s**	153.62	**22.77**	**2.2 s**	284.74	**20.51**	**7.9 s**	474.09	**16.05**	**18.9 s**

5 EXPERIMENT

5.1 EXPERIMENT SETTINGS

Datasets. We evaluate Just Initialize on four representative routing problems: TSP, CVRP, VRPTW, and PCTSP, covering large-scale instances from 1K to 100K nodes. For TSP and CVRP, we follow previous large-scale synthetic benchmarks and use 80 instances per problem, including 16 instances at each scale of 5K, 10K, 20K, 50K, and 100K nodes. For VRPTW and PCTSP, we use 56 instances for each problem following established generation protocols.

Baselines. We compare against representative methods from classical optimization, neural constructive optimization, heatmap-based approaches, decomposition methods, and neural-enhanced metaheuristics. Empirical reference solutions are generated independently using LKH-3 for TSPs and CVRPs, HGS-PyVRP for VRPTWs, and LKH-3-PCTSP for PCTSPs. These references are used only for computing relative gaps.

Metrics. We report objective value, relative gap, and time cost. For Just Initialize, runtime includes compression, compact-space optimization, solution recovery, and downstream refinement. We additionally evaluate initialization effectiveness by comparing downstream optimization from Just Initialize with alternative starting points.

Implementation Details. Experiments are conducted on an Intel Xeon Platinum 8358P 64-core CPU, eight NVIDIA GeForce RTX 4090 GPUs, and 256 GB RAM. Each configuration is run three times, and the results are averaged; objective values and gaps are reported only for feasible solutions.

5.2 COMPARATIVE STUDY

The results are summarized in Tables 1 and 2. Across four routing benchmarks, **JI+Refine** consistently achieves competitive or superior solution quality while substantially reducing computational cost. On large-scale TSP and CVRP instances, JI+Refine scales to 100K nodes and maintains stable solution gaps, while many existing approaches suffer from rapidly increasing runtime or fail to complete due to computational limitations. Notably, JI+VND also achieves competitive performance compared with many NCO-based methods, demonstrating that the effectiveness of JI mainly comes from providing better optimization-friendly initializations rather than relying on a specific re-

finement strategy. For VRPTW and PCTSP, JI+Refine achieves competitive solution quality across different scales while reducing the solving time from minutes or hours to seconds.

Overall, JI+Refine achieves an average speedup of approximately 70× compared with existing approaches while preserving comparable solution quality. On 10K-node instances, JI+Refine completes optimization within seconds, and even 100K-node instances can be solved within tens of seconds. These results demonstrate that optimization-friendly initialization, combined with effective refinement, provides an efficient way to reduce the computational burden of large-scale routing optimization.

Beyond the large-scale synthetic benchmarks, we further evaluate JI+Refine under diverse instance distributions, including non-uniform TSP settings with clustered, explosion, and implosion patterns. We also test JI+Refine on widely used TSPLIB and CVRPLIB benchmarks to examine its generalization ability beyond synthetic settings. Across these additional benchmarks, JI+Refine maintains competitive solution quality with consistent computational efficiency, demonstrating robustness across different spatial distributions and benchmark sources. Detailed results are provided in Appendix E.

Table 2: Comparison results on large-scale VRPTW and PCTSP instances.

Method	VRPTW2K			VRPTW5K			VRPTW10K			VRPTW20K		
	Obj. ↓	Gap	Time	Obj. ↓	Gap	Time	Obj. ↓	Gap	Time	Obj. ↓	Gap	Time
HGS-PyVRP (Reference)	335.93	-	-	823.85	-	-	1419.13	-	-	2479.68	-	-
OR-Tools	352.83	5.03	30.0 min	891.49	8.21	2.0 min	1498.60	5.60	10.0 min	2584.07	4.21	30.0 min
VROOM	342.08	**1.83**	6.0 min	831.43	**0.92**	14.7 min	-	OOM	-	-	OOM	-
ALNS	381.58	13.59	3.8 h	901.13	9.38	3.6 h	1520.17	7.12	3.1 h	2615.07	5.46	3.3 h
JI+Refine (Ours)	362.91	8.25	**2.9 s**	863.80	5.16	**9.6 s**	1478.16	**4.43**	**9.3 s**	2574.90	**3.96**	**15.8 s**
Method	PCTSP2K			PCTSP5K			PCTSP10K			PCTSP20K		
	Obj. ↓	Gap	Time	Obj. ↓	Gap	Time	Obj. ↓	Gap	Time	Obj. ↓	Gap	Time
LKH-3 (Reference)	27.28	-	-	44.39	-	-	66.65	-	-	94.99	-	-
GLOP-G		OOT		49.55	11.63	1.5 s		OOT			OOT	
GLOP-S		OOT		49.39	11.26	6.9 s		OOT			OOT	
Attention Model	58.55	114.62	1.6 s	109.38	146.41	4.0 s	174.96	162.51	7.4 s		OOM	
DeepACO	40.68	49.12	1.3 min	78.98	77.92	3.4 min		OOM			OOM	
OR-Tools PCTSP	29.98	**9.88**	7.5 min	59.64	34.35	7.5 min	85.31	27.98	7.5 min	125.02	31.62	12.8 min
JI+Refine (Ours)	30.99	13.62	**0.5 s**	48.37	**9.01**	**1.3 s**	70.02	**5.05**	**2.9 s**	101.30	**6.65**	**6.4 s**

5.3 Effectiveness as a Solver Initialization

We evaluate whether JI can improve downstream optimization by providing more effective initial solutions. Specifically, we compare each solver in its original setting with the corresponding solver augmented by JI initialization. All reported runtimes include the complete JI pipeline. Table 3 reports the final relative gaps and total solving times.

Table 3: Search performance with different initializations and refinement procedures. Bold values indicate the better result within each solver pair before rounding.

	1K		5K		10K		20K		50K		100K	
Solver	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
2-opt	11.22	0.03 s	11.74	0.3 s	11.70	0.5 s	11.64	1.5 s	11.70	9.9 s	11.67	42.9 s
JI+2-opt	**8.63**	**0.02 s**	**9.24**	**0.2 s**	**9.24**	**0.1 s**	**9.21**	**0.2 s**	**9.21**	**0.8 s**	**9.10**	**3.3 s**
VND	5.22	4.3 s	5.57	35.5 s	5.51	42.5 s	5.78	42.4 s	6.64	1.4 min	8.17	1.2 min
JI+VND	**4.44**	**2.7 s**	**4.77**	**24.3 s**	**4.85**	**28.9 s**	**5.05**	**35.4 s**	**5.93**	**34.9 s**	**6.08**	**1.1 min**
ILS	6.99	1.4 s	8.76	41.3 s	9.18	40.1 s	10.03	41.8 s	10.74	1.4 min	11.21	1.4 min
JI+ILS	**6.59**	**0.9 s**	**8.49**	**16.9 s**	**8.75**	**12.9 s**	**8.90**	**31.8 s**	**9.04**	**1.3 min**	**9.04**	**1.3 min**
GLS	10.38	2.8 s	11.22	43.7 s	11.44	43.4 s	11.53	40.3 s	11.68	45 s	11.66	1.2 min
JI+GLS	**8.12**	**1.5 s**	**9.00**	**14.3 s**	**9.07**	**16.6 s**	**9.13**	**34.6 s**	**9.19**	**35.3 s**	**9.10**	**54.8 s**
JI+Refine	**2.56**	**0.04 s**	**3.48**	**0.2 s**	**3.51**	**0.5 s**	**3.68**	**1.4 s**	**3.74**	**4.8 s**	**3.80**	**12.2 s**

The results show that JI consistently improves both solution quality and computational efficiency across all refinement procedures and problem scales. Compared with the original solvers, JI-enhanced solvers achieve lower final gaps with shorter overall runtimes, demonstrating that JI pro-

vides effective initializations for downstream optimization. This verifies that improving initialization can reduce the burden of subsequent optimization.

We further evaluate our dedicated refinement strategy under the same JI-generated initialization. Compared with existing refinement procedures using identical initializations, the proposed refinement achieves better solution quality, showing that effective initialization and specialized refinement provide complementary benefits.

Additional initialization analyses, including matched-gap and greedy-start comparisons as well as search-budget evaluations, are provided in Appendix E.5–E.6.

5.4 Ablation Study

We perform ablation studies on uniform TSP instances by removing each component of JI individually, including compression, compressed-space solving, and recovery. The gap increase relative to the full model is reported in parentheses. As shown in Table 4, removing compression or recovery consistently increases the final gap across different scales, with over 1% degradation in large-scale instances, while removing the solving stage causes smaller changes. These results verify that compression and recovery are essential for generating effective initializations, and the complete JI pipeline enables optimization to start from more favorable regions of the solution space.

Table 4: JI ablation on uniform TSP instances. Numbers in parentheses denote the gap increase relative to the full model. All results are averaged over three seeds.

	Full		w/o compress		w/o solve		w/o restore	
Scale	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)
1K	2.84	0.03	3.59 (+0.75)	0.04	3.39 (+0.55)	0.05	3.90 (+1.06)	0.03
5K	3.62	0.16	4.26 (+0.64)	0.18	3.68 (+0.06)	0.16	4.44 (+0.82)	0.17
10K	3.49	0.36	4.83 (+1.34)	0.47	3.60 (+0.11)	0.35	4.66 (+1.17)	0.41
20K	3.70	0.89	4.88 (+1.18)	1.20	3.74 (+0.04)	0.91	4.87 (+1.17)	1.11
50K	3.67	4.23	4.82 (+1.15)	7.23	4.25 (+0.58)	4.12	4.96 (+1.29)	5.30
100K	3.78	12.56	5.00 (+1.22)	24.91	4.34 (+0.56)	12.75	5.08 (+1.30)	16.89

6 Conclusion

In this work, we revisit large-scale routing optimization from the perspective of where optimization begins. Instead of continuously increasing the complexity of optimization procedures, we show that a more effective starting point can fundamentally reduce the difficulty of downstream search. Based on this insight, we propose **Just Initialize**, a training-free and solver-independent initialization component that separates solution discovery from solution optimization. By exploring coarse global structures in a compact space and recovering them as optimization-friendly starting points, Just Initialize enables existing solvers to focus on refining promising regions rather than searching the entire solution space.

Extensive experiments on TSP, CVRP, VRPTW, and PCTSP demonstrate that Just Initialize scales to instances from 1K to 100K nodes while maintaining competitive solution quality with substantially reduced computational cost. These results reveal that initialization can serve as an effective mechanism for navigating large combinatorial search spaces, providing a complementary direction to improving optimization algorithms themselves.

AI USE STATEMENT

In this work, we used generative AI tools to aid or polish writing, retrieve and discover related work, and draft sections of the paper.

We have not used generative AI tools for the following other required-disclosure tasks that were relevant to our research but performed entirely by the authors: helping develop theoretical models or conceptual frameworks, proposing or refining hypotheses, and designing or providing feedback on research methodology or experiments.

The remaining required-disclosure tasks are not applicable to this work because our study did not involve them: assisting in the writing of proofs, providing critical ingredients for proving mathematical claims, and formulating mathematical claims.

Additionally, we used generative AI tools for the following recommended-disclosure tasks: drafting parts of a research paper, editing a research paper to improve readability, formatting references, and creating or editing software code (for utility scripts and visualisation functions).

We have reviewed all AI-assisted work. Specifically:

- All AI-generated code was tested on multiple runs, verified for correctness against expected outputs, and peer-reviewed by at least two authors.
- The cleaned dataset was manually inspected for consistency and missing-value handling after AI-assisted preprocessing.
- AI-drafted text was substantially revised to match our scientific voice and checked for factual accuracy against our experimental results.

We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI. The authors alone developed the core research ideas, designed the experiments, and interpreted the final results, without AI involvement.

ETHICS STATEMENT

This paper studies training-free initialization methods for large-scale routing optimization. It does not involve human subjects, personal data, or privacy-sensitive information. All datasets used are publicly available and contain mathematical problems. We foresee no ethical concerns and declare no conflicts of interest.

REPRODUCIBILITY STATEMENT

We describe Just Initialize in Section 4, with algorithmic pseudocode and parameter settings in Appendix B. Dataset construction procedures, generation seeds, and additional solver settings are provided in Appendix D. Experimental hardware, evaluation metrics, and runtime measurement conventions are described in Section 5.1.

REFERENCES

Luca Accorsi and Daniele Vigo. Routing one million customers in a handful of minutes. *Computers & Operations Research*, 164:106562, 2024. doi: 10.1016/j.cor.2024.106562.

Marjolein Aerts-Veenstra, Marilène Cherkesly, and Timo Gschwind. A unified branch-price-and-cut algorithm for multicompartment pickup and delivery problems. *Transportation Science*, 58(5): 1121–1142, 2024. doi: 10.1287/trsc.2023.0252.

John E. Beasley. Route first—cluster second methods for vehicle routing. *Omega*, 11(4):403–408, 1983. doi: 10.1016/0305-0483(83)90033-6.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021. doi: 10.1016/j.ejor.2020.07.063.

Jieyi Bi, Yining Ma, Jiahai Wang, Zhiguang Cao, Jinbiao Chen, Yuan Sun, and Yeow Meng Chee. Learning generalizable models for vehicle routing problems via knowledge distillation. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Yuanyao Chen, Rongsheng Chen, Fu Luo, and Zhenkun Wang. Improving generalization of neural combinatorial optimization for vehicle routing problems via test-time projection learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Concorde TSP Solver. Concorde tsp solver. `https://math.uwaterloo.ca/tsp/concorde/`.

William Cook, Stephan Held, and Keld Helsgaun. Constrained local search for last-mile routing. *Transportation Science*, 58(1):12–26, 2024. doi: 10.1287/trsc.2022.1185.

Raafat Elshaer and Hadeer Awad. A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers & Industrial Engineering*, 140:106242, 2020. doi: 10.1016/j.cie.2019.106242.

Han Fang, Zhihao Song, Paul Weng, and Yutong Ban. INViT: A generalizable routing problem solver with invariant nested view transformer. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12973–12992. PMLR, 2024.

Marshall L. Fisher and Ramchandran Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981. doi: 10.1002/net.3230110205.

Mette Gamst, Richard Martin Lusby, and Stefan Ropke. Exact and heuristic methods for the split delivery vehicle routing problem. *Transportation Science*, 58(4):741–760, 2024. doi: 10.1287/trsc.2022.0353.

Billy E. Gillett and Leland R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349, 1974. doi: 10.1287/opre.22.2.340.

Bruce Golden, Xingyin Wang, and Edward Wasil. *The Evolution of the Vehicle Routing Problem: A Survey of VRP Research and Practice from 2005 to 2022*. Synthesis Lectures on Operations Research and Applications. Springer Nature Switzerland, 2023. doi: 10.1007/978-3-031-18716-2.

Google. OR-Tools: Routing. `https://developers.google.com/optimization/routing`.

Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. doi: 10.1016/S0377-2217(99)00284-2.

Keld Helsgaun. An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. Technical report, Roskilde University, 2017.

Christoph Kerscher and Stefan Minner. Decompose-route-improve framework for solving large-scale vehicle routing problems with time windows. *Transportation Research Part E: Logistics and Transportation Review*, 204:104409, 2025. doi: 10.1016/j.tre.2025.104409.

Minsu Kim, Jinkyoo Park, and Joungho Kim. Learning collaborative policies to solve NP-hard routing problems. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-NCO: Leveraging symmetricity for neural combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019.

Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. Deep policy dynamic programming for vehicle routing problems. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pp. 190–213. Springer, 2022.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. POMO: Policy optimization with multiple optima for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pp. 21188–21198, 2020.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Fu Luo, Xi Lin, Zhenkun Wang, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Self-improved learning for scalable neural combinatorial optimization. *arXiv preprint arXiv:2403.19561*, 2024.

Fu Luo, Xi Lin, Yaoxin Wu, Zhenkun Wang, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. Boosting neural combinatorial optimization for large-scale vehicle routing problems. In *International Conference on Learning Representations*, 2025a.

Fu Luo, Xi Lin, Mengyuan Zhong, Fei Liu, Zhenkun Wang, Jianyong Sun, and Qingfu Zhang. Learning to insert for constructive neural vehicle routing solver. In *Advances in Neural Information Processing Systems*, volume 38, 2025b.

Katarzyna Nałęcz-Charkiewicz, Arnav Das, Turbasu Chatterjee, Joshua Keene, Paweł Góra, and Carlos C. N. Kuhn. Graph coarsening approach to the vehicle routing problem: An approximation strategy. *IEEE Access*, 13:22459–22472, 2025. doi: 10.1109/ACCESS.2025.3534677.

Clifford Orloff and David Caprera. Reduction and solution of large scale vehicle routing problems. *Transportation Science*, 10(4):361–373, 1976. doi: 10.1287/trsc.10.4.361.

Xuanhao Pan, Yan Jin, Yuandong Ding, Mingxiao Feng, Li Zhao, Lei Song, and Jiang Bian. H-TSP: Hierarchically solving the large-scale traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 9345–9353, 2023. doi: 10.1609/aaai.v37i8.26120.

Christian Prins, Philippe Lacomme, and Caroline Prodhon. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies*, 40:179–200, 2014. doi: 10.1016/j.trc.2014.01.011.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Eduardo Queiroga, Ruslan Sadykov, and Eduardo Uchoa. A POPMUSIC matheuristic for the capacitated vehicle routing problem. *Computers & Operations Research*, 136:105475, 2021. doi: 10.1016/j.cor.2021.105475.

Gerhard Reinelt. TSPLIB—a traveling salesman problem library. *ORSA Journal on Computing*, 3 (4):376–384, 1991. doi: 10.1287/ijoc.3.4.376.

Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006. doi: 10.1287/trsc.1050.0135.

Martin Simensen, Geir Hasle, and Magnus Stålhane. Combining hybrid genetic search with ruin-and-recreate for solving the capacitated vehicle routing problem. *Journal of Heuristics*, 28:653–697, 2022. doi: 10.1007/s10732-022-09500-9.

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003. doi: 10.1287/ijoc.15.4.333.24890.

Thibaut Vidal. Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood. *Computers & Operations Research*, 140:105643, 2022. doi: 10.1016/j.cor.2021.105643.

VROOM Project. VROOM: Vehicle routing open-source optimization machine. `https://github.com/VROOM-Project/vroom`.

Qi Wang and Chunlei Tang. Deep reinforcement learning for transportation network combinatorial optimization: A survey. *Knowledge-Based Systems*, 233:107526, 2021. doi: 10.1016/j.knosys.2021.107526.

Liang Xin, Wen Song, Zhiguang Cao, and Jie Zhang. NeuroLKH: Combining deep learning model with lin–kernighan–helsgaun heuristic for solving the traveling salesman problem. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Weibo Yang, Liangjun Ke, David Z. W. Wang, and Jasmine Siu Lee Lam. A branch-price-and-cut algorithm for the vehicle routing problem with release and due dates. *Transportation Research Part E: Logistics and Transportation Review*, 145:102167, 2021. doi: 10.1016/j.tre.2020.102167.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. DeepACO: Neural-enhanced ant systems for combinatorial optimization. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. GLOP: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 20284–20292, 2024. doi: 10.1609/aaai.v38i18.30009.

Zhi Zheng, Changliang Zhou, Xialiang Tong, Mingxuan Yuan, and Zhenkun Wang. UDC: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

A Baselines and Benchmarks

A.1 Baselines

We compare Just Initialize with the methods in Tables 1 and 2. Each description below summarizes a method's main strength and the corresponding computational or modeling tradeoff.

Classical solvers and metaheuristics. LKH-3 (Helsgaun, 2017) combines Lin–Kernighan search with problem-specific extensions and repeated local search. Concorde (Concorde TSP Solver) provides exact TSP solutions and optimality certificates through branch-and-cut, at a computational cost that rises sharply with instance size. HGS (Vidal, 2022) combines population diversity with specialized CVRP local search, while crossover and route improvement add search overhead. OR-Tools and its PCTSP configuration (Google) offer flexible routing constraints and established search operators, with solution quality depending on the chosen search strategy and time limit. VROOM (VROOM Project) rapidly constructs and improves feasible vehicle routes with capacities and time windows, trading exact guarantees for heuristic search speed. ALNS (**?**) explores route changes through adaptive removal and repair operators with repeated reconstruction. PCTSP-ILS iteratively changes the visited-customer set and tour to explore prize–penalty tradeoffs, while repeated local improvement increases runtime.

Neural construction and diffusion methods. POMO (Kwon et al., 2020) exploits multiple equivalent starting points to produce diverse candidate routes, while eightfold augmentation multiplies inference work relative to the unaugmented configuration. LEHD (Luo et al., 2023) uses a heavy decoder to make context-aware construction decisions on large instances, while sequential decoding raises inference cost. INViT (Fang et al., 2024) uses invariant nested views to transfer routing decisions across scales and distributions, while processing multiple views adds computation. SIL (Luo et al., 2024) learns from solutions improved by local reconstruction and uses linear-complexity

attention for scale, while generating and retraining on pseudo-labels adds training effort. L2C-Insert (Luo et al., 2025b) can place a customer at any valid position in a partial route, while evaluating insertion positions costs more than simple append decisions. The Attention Model (Kool et al., 2019) learns route construction directly with attention, while autoregressive decoding remains sequential as the instance grows. DIFUSCO (Sun & Yang, 2023) uses diffusion-generated edge scores to guide global tour construction, while denoising and tour recovery add inference stages.

Decomposition and learned search. GLOP (Ye et al., 2024) combines global partitioning with local neural construction to scale to large routing instances, while the final tour depends on the quality of the partition. GLOP-G (Ye et al., 2024) uses greedy partition decoding to reduce sampling work, while a single partition offers less exploration than sampled decoding. UDC (Zheng et al., 2024) jointly learns division and solution of subproblems across routing tasks, while reunion must reconcile decisions made in separate parts. H-TSP (Pan et al., 2023) constructs large tours through hierarchical node selection and local routing, while early subset decisions influence later connections. L2D (Li et al., 2021) learns which subroutes to delegate to a local solver for focused improvement, while repeated delegated searches add runtime. TTPL (Chen et al., 2025) adapts a learned policy to test-instance features to address distribution shift, while the projection step adds test-time work. DeepACO (Ye et al., 2023) combines learned heuristic information with ant-colony exploration, while more ants and iterations increase the search budget.

A.2 BENCHMARKS

Let $N = \{1, \ldots, n\}$ be the set of cities or customers. For a node set V, let $A(V) = \{(i, j) : i, j \in V,\ i \neq j\}$ be its directed arc set and let d_{ij} be the instance-defined travel distance. The formulations below describe the feasible routes and objectives; Appendix D specifies how their attributes and reference values are generated.

For synthetic instances, coordinates and distance-related quantities use a 10^6 integer export; VRPTW time attributes and PCTSP prizes, penalties, and quota are scaled consistently, while demands and capacities retain their original units. Reported synthetic objectives are divided by 10^6. Public TSP instances use their source distance rules and unscaled objectives. For an instance objective C and its recorded reference C_{ref}, the reported gap is $100(C - C_{\mathrm{ref}})/C_{\mathrm{ref}}$ percent; dataset composition and reference procedures are detailed in Appendix D.

Definition A.1 (TSP). For the traveling salesman problem, set $V = N$ and let $x_{ij} = 1$ when the tour traverses arc (i, j). Every city has one predecessor and one successor, and the subtour constraints connect them into a single tour:

$$\min_{x} \quad \sum_{(i,j)\in A(V)} d_{ij} x_{ij} \tag{12}$$

$$\text{s.t.} \quad \sum_{j\in V\setminus\{i\}} x_{ij} = 1, \qquad \forall i \in V, \tag{13}$$

$$\sum_{j\in V\setminus\{i\}} x_{ji} = 1, \qquad \forall i \in V, \tag{14}$$

$$\sum_{\substack{i,j\in S \\ i\neq j}} x_{ij} \leq |S| - 1, \qquad \forall S \subset V,\ 2 \leq |S| \leq n-1, \tag{15}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in A(V). \tag{16}$$

The synthetic TSP benchmarks contain 96 uniform and 144 extended-distribution instances; the additional 42 public instances retain their original distance definitions.

Definition A.2 (CVRP). For the capacitated vehicle routing problem, let $V = \{0\} \cup N$, where 0 is the depot, $q_i > 0$ is customer demand, and Q is vehicle capacity. The binary variable x_{ij} selects arc (i, j), and the continuous variable u_i is the load delivered up to customer i on its route. The number

of depot-to-depot routes is chosen by the solution:

$$\min_{x,u} \quad \sum_{(i,j)\in A(V)} d_{ij}x_{ij} \tag{17}$$

$$\text{s.t.} \quad \sum_{j\in V\setminus\{i\}} x_{ij} = 1, \qquad \forall i \in N, \tag{18}$$

$$\sum_{j\in V\setminus\{i\}} x_{ji} = 1, \qquad \forall i \in N, \tag{19}$$

$$\sum_{j\in N} x_{0j} = \sum_{i\in N} x_{i0}, \tag{20}$$

$$u_j \geq u_i + q_j - Q(1 - x_{ij}), \qquad \forall i, j \in N,\ i \neq j, \tag{21}$$

$$q_i \leq u_i \leq Q, \qquad \forall i \in N, \tag{22}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in A(V). \tag{23}$$

Positive demands and the load constraints also rule out customer-only cycles. The main CVRP comparison uses 80 instances at five scales from 5K to 100K customers, each with one additional depot.

Definition A.3 (VRPTW). The vehicle routing problem with time windows retains customer demands q_i and capacity Q, and adds a service duration s_i and a service-start window $[a_i, b_i]$ at each customer. To represent route departures and returns, use a start depot 0 and an end-depot copy $n + 1$. The arc set A^{tw} contains arcs from 0 to customers, between distinct customers, and from customers to $n+1$. The end copy has the depot's location and window, so $d_{i,n+1} = d_{i0}$ and $\tau_{i,n+1} = \tau_{i0}$. Here t_i is the continuous service start, τ_{ij} is travel time, u_i is continuous cumulative load, and M_{ij} is a valid time-propagation bound. Waiting is allowed before service begins:

$$\min_{x,u,t} \quad \sum_{(i,j)\in A^{\mathrm{tw}}} d_{ij}x_{ij} \tag{24}$$

$$\text{s.t.} \quad \sum_{j:(i,j)\in A^{\mathrm{tw}}} x_{ij} = 1, \qquad \forall i \in N, \tag{25}$$

$$\sum_{j:(j,i)\in A^{\mathrm{tw}}} x_{ji} = 1, \qquad \forall i \in N, \tag{26}$$

$$\sum_{j\in N} x_{0j} = \sum_{i\in N} x_{i,n+1}, \tag{27}$$

$$u_j \geq u_i + q_j - Q(1 - x_{ij}), \qquad \forall i, j \in N,\ i \neq j, \tag{28}$$

$$q_i \leq u_i \leq Q, \qquad \forall i \in N, \tag{29}$$

$$t_j \geq t_i + s_i + \tau_{ij} - M_{ij}(1 - x_{ij}), \qquad \forall (i, j) \in A^{\mathrm{tw}}, \tag{30}$$

$$a_i \leq t_i \leq b_i, \qquad \forall i \in N, \tag{31}$$

$$t_0 = a_0, \quad a_0 \leq t_{n+1} \leq b_0, \tag{32}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall (i, j) \in A^{\mathrm{tw}}. \tag{33}$$

We take $s_0 = 0$, $a_{n+1} = a_0$, and $M_{ij} = \max\{0, b_i + s_i + \tau_{ij} - a_j\}$; the benchmark has 72 instances from 1K to 20K customers, each with one physical depot.

Definition A.4 (PCTSP). For the prize-collecting traveling salesman problem, let $V = \{0\} \cup N$, with depot 0, prize p_i, omission penalty λ_i, and required collected prize P. The binary variable y_i records whether customer i is visited, while x_{ij} selects tour arcs. A single depot tour minimizes travel plus

penalties for omitted customers:

$$\min_{x,y} \quad \sum_{(i,j)\in A(V)} d_{ij}x_{ij} + \sum_{i\in N} \lambda_i(1-y_i) \tag{34}$$

$$\text{s.t.} \quad \sum_{j\in V\setminus\{i\}} x_{ij} = y_i, \qquad \forall i \in N, \tag{35}$$

$$\sum_{j\in V\setminus\{i\}} x_{ji} = y_i, \qquad \forall i \in N, \tag{36}$$

$$\sum_{j\in N} x_{0j} = \sum_{i\in N} x_{i0} = 1, \tag{37}$$

$$\sum_{i\in S}\sum_{j\in V\setminus S} x_{ij} \geq y_k, \qquad \forall \emptyset \neq S \subseteq N,\ k \in S, \tag{38}$$

$$\sum_{i\in N} p_i y_i \geq P, \tag{39}$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A(V), \tag{40}$$

$$y_i \in \{0,1\}, \qquad \forall i \in N. \tag{41}$$

The PCTSP benchmark contains 72 instances from 1K to 20K optional customers, with $P = 1$ before integer export.

B Implementation Details of Just Initialize

B.1 Algorithmic Implementation

Algorithm 2 summarizes the implementation of Just Initialize. The procedure constructs an initial solution in the compact space, improves it under a restricted geometric neighborhood, recovers a structurally compatible solution in the original space, and performs a bounded local refinement. During compression, each compact unit additionally retains an ordered sequence of its original nodes, which is used to instantiate the unit during recovery.

The compact representation follows the same structure for all four problem classes, while the attributes retained by a unit reflect the corresponding constraints. TSP primarily uses the local node sequence and its boundary geometry. CVRP additionally retains the aggregated demand of each unit, PCTSP retains aggregated prize and penalty information, and VRPTW augments the unit representation with demand and temporal boundary information. Accordingly, the feasibility test in Algorithm 2 checks only the resources relevant to the current problem.

The compact objective F_P is likewise instantiated according to the routing problem. For TSP, it measures the closed-tour cost induced by the ordered compact fragments. For CVRP, it is the sum of depot-to-depot route costs, with capacity-infeasible routes excluded. VRPTW uses the same distance-based route objective while additionally enforcing capacity and time-window feasibility during concatenation. For PCTSP, the travel cost of the selected compact tour is combined with the penalties of unselected units, and plans that do not satisfy the prize quota are infeasible. The incremental score Δ_P used during construction follows the same information: required-node problems are primarily guided by incremental connection cost, while optional unit selection also accounts for the prize–penalty structure.

Compact-space search uses the same restricted geometric neighborhood in all cases. Exchange, relocation, and reversal modify the arrangement of compact units, while optional-node instances additionally allow the selected set to change. These variations affect the candidate set but not the subsequent search procedure.

Downstream evaluation is applied only to a small shortlist rather than to every compact neighbor. For deterministic refinement, this evaluation corresponds to a single trajectory of $\widehat{J}_{B,P}$. Recovery itself expands each compact route by concatenating the stored local sequences according to their selected orientations. The resulting original-space solution is then refined under the same problem-

Algorithm 2: Implementation of Just Initialize

Input: Instance I; compression scale b; candidate width K_c; search budget T_c; evaluation samples S; budgets B, B_μ; temperature τ
Output: Refined initialization π_{JI}

$(\tilde{I}, M, \Phi_s) \leftarrow \textsc{Compress}(I, b)$ using Algorithm 1;
$\mathcal{Q} \leftarrow \textsc{BuildSpatialIndex}(\{z_i : C_i \in \tilde{I}\})$;
$\tilde{\pi} \leftarrow \emptyset, \quad \mathcal{U} \leftarrow \textsc{RequiredCompactUnits}(\tilde{I})$;
while $\mathcal{U} \neq \emptyset$ **do**
  $\mathcal{C} \leftarrow \textsc{QueryCandidates}(\mathcal{Q}, \tilde{\pi}, \mathcal{U}, K_c)$;
  $\mathcal{A} \leftarrow \{(C_i, t) : C_i \in \mathcal{C},\ t \in \mathcal{T}_i,\ \textsc{FeasibleAppend}(\tilde{\pi}, C_i, t, \tilde{K})\}$;
  if $\mathcal{A} = \emptyset$ **then**
    $\tilde{\pi} \leftarrow \textsc{StartNewRoute}(\tilde{\pi})$;
  else
    $(C^\star, t^\star) \leftarrow \arg\min_{(C_i,t)\in\mathcal{A}} \textsc{ConnectionCost}(\tilde{\pi}, C_i, t, \tilde{C})$;
    $\tilde{\pi} \leftarrow \tilde{\pi} \oplus (C^\star, t^\star), \quad \mathcal{U} \leftarrow \mathcal{U} \setminus \{C^\star\}$;
for $k \leftarrow 1$ **to** T_c **do**
  $C_i \leftarrow \textsc{SelectCompactUnit}(\tilde{\pi}), \quad \mathcal{C}_i \leftarrow \textsc{QueryCandidates}(\mathcal{Q}, C_i, K_c)$;
  $\widetilde{\mathcal{N}} \leftarrow \textsc{GenerateCompactNeighbors}(\tilde{\pi}, C_i, \mathcal{C}_i) \cap \Omega(\tilde{I})$;
  foreach $\tilde{\pi}' \in \widetilde{\mathcal{N}} \cup \{\tilde{\pi}\}$ **do**
    $\widehat{J}_B(\tilde{\pi}') \leftarrow 0$;
    for $s \leftarrow 1$ **to** S **do**
      $\mathcal{P}_s \leftarrow \textsc{RecoveryCandidates}(\tilde{\pi}', M)$;
      $\pi_0^{(s)} \sim \dfrac{\exp(g(\pi; \tilde{\pi}', M)/\tau)}{\sum_{\bar{\pi}\in\mathcal{P}_s} \exp(g(\bar{\pi}; \tilde{\pi}', M)/\tau)}$;
      $\pi^{(s)} \leftarrow \pi_0^{(s)}$;
      for $r \leftarrow 1$ **to** B **do**
        $\mathcal{N} \leftarrow \textsc{CandidateRestrict}(\mathcal{N}_I(\pi^{(s)})) \cap \Omega(I)$;
        $\pi' \leftarrow \arg\min_{\bar{\pi}\in\mathcal{N}} C_I(\bar{\pi})$;
        if $\mathcal{N} = \emptyset$ ***or*** $C_I(\pi') \geq C_I(\pi^{(s)})$ **then**
          break
        $\pi^{(s)} \leftarrow \pi'$;
      $\widehat{J}_B(\tilde{\pi}') \leftarrow \widehat{J}_B(\tilde{\pi}') + C_I(\pi^{(s)})/S$;
  $\tilde{\pi} \leftarrow \arg\min_{\bar{\pi}\in\widetilde{\mathcal{N}}\cup\{\tilde{\pi}\}} \widehat{J}_B(\bar{\pi})$;
$\mathcal{P} \leftarrow \textsc{RecoveryCandidates}(\tilde{\pi}, M), \quad \pi_0^\star \sim \dfrac{\exp(g(\pi; \tilde{\pi}, M)/\tau)}{\sum_{\bar{\pi}\in\mathcal{P}} \exp(g(\bar{\pi}; \tilde{\pi}, M)/\tau)}$;
$\pi \leftarrow \pi_0^\star$;
for $r \leftarrow 1$ **to** B_μ **do**
  $\mathcal{N}_\mu \leftarrow \textsc{CandidateRestrict}(\mathcal{N}_I(\pi)) \cap \Omega(I)$;
  $\pi' \leftarrow \arg\min_{\bar{\pi}\in\mathcal{N}_\mu} C_I(\bar{\pi})$;
  if $\mathcal{N}_\mu = \emptyset$ ***or*** $C_I(\pi') \geq C_I(\pi)$ **then**
    break
  $\pi \leftarrow \pi'$;
return π;

dependent objective and feasibility conditions until no improving candidate remains or the budget B_μ is exhausted.

Table 5: Shared implementation parameters of Just Initialize. m denotes the number of compact units.

Parameter	Value	Role
K_c	$\min\{64, m\}$	Candidate width for compact-solution construction
K_{local}	$\min\{24, m\}$	Candidate width for compact-space local search
K_{merge}	$\min\{32, m\}$	Candidate width for route merging, when applicable
S	1	Number of refinement trials used to estimate $\widehat{J}_B$
τ	Low temperature	Controls the concentration of recovery sampling
B_μ	Bounded	Maximum number of post-recovery refinement steps
s	0	Random seed for compact-space search

B.2 Parameter Settings

Table 5 summarizes the parameters shared across all problem classes. These settings are kept fixed throughout the experiments. The problem-dependent settings primarily involve compression granularity and compact-search budget. For TSP, compact units contain approximately 32 nodes, while PCTSP uses smaller units of 8 nodes. CVRP controls the granularity by accumulated demand, with a target load of $0.8Q$ per compact unit. For VRPTW, compression groups up to 40 local routes, with individual fragments limited to at most 16 nodes to preserve local temporal structure. The compact-search budget T_c is scaled with the compressed size m and adjusted to the structural complexity of each problem class. All problem-specific settings are fixed within each problem class and are not tuned for individual instances.

C Theoretical Analysis

We analyze Just Initialize under the computational model that Algorithm 2 implements. An instance contains n nodes, which are grouped into m compact units, so that a unit contains $b = n/m$ nodes on average; we assume $b > 1$. Each unit carries a constant-size summary of the nodes it contains: its geometry z_i, its boundary states $\mathcal{T}_i$, and, where the problem requires it, aggregated demand, prize, or temporal information. Both spaces are searched with candidate-restricted neighborhoods: a sweep in the original space examines at most K candidate neighbors at each of the n positions, and a sweep in the compact space examines at most K_c candidate units at each of the m units. The refinement operator is a descent procedure: it accepts a move only when the objective strictly decreases, and it terminates when no improving candidate remains or its budget is exhausted. The analysis below addresses three questions in turn, and each result is stated with the assumptions it requires and the design choice it supports.

C.1 Decision Count and Cost of the Compact Stage

Proposition 1 (Decision-space reduction). Under candidate-restricted search, one sweep examines at most

$$D_o = nK, \qquad D_c = mK_c \tag{42}$$

candidate decisions in the original and compact spaces, respectively. With $m = n/b$,

$$\frac{D_c}{D_o} = \frac{K_c}{bK}, \tag{43}$$

so that when the two candidate widths are of the same order ($K_c = \Theta(K)$), the number of decisions examined per sweep is reduced by the compression factor b.

Proof. Each sweep visits every position once and examines at most K (respectively K_c) candidates per position, which gives the two counts in Eq. (1). Substituting $m = n/b$ yields Eq. (43). □

Proposition 2 (Cost of a compact decision). Let the connection cost of appending a unit to a partial compact plan be evaluated from the stored summaries of the two units, as in Algorithm 2.

Then this evaluation runs in $O(1)$ time, independent of the number b of original nodes represented by a unit, and one construction sweep costs

$$T_{\text{construct}} = O(mK_c), \tag{44}$$

whereas evaluating the same number of placements against the full instance costs $O(nK)$.

Proof. The connection cost is a function of the two unit summaries, each of constant size, hence is evaluated in constant time. Multiplying by the number of examined candidates gives $O(mK_c)$ for the compact sweep. In the original space, each candidate evaluation reads both endpoint neighborhoods and thus costs $O(K)$ per position, giving $O(nK)$ per sweep. □

Combining Propositions 1 and 2, the compact stage removes a factor b from both the number of decisions and, by Eq. (43), the geometric work per sweep. The remaining cost of a run decomposes as

$$T_{\text{JI}} = T_{\text{compress}} + O(mK_c) + \sum_{j=1}^{L}\sum_{s=1}^{S}\Big(T_{\text{rec},j,s} + B\,T_{\text{move}}\Big) + T_{\text{final}}, \tag{45}$$

where L is the number of compact candidates whose initialization quality is estimated, S is the number of refinement trials per candidate, $T_{\text{move}} = O(nK)$ is the cost of one candidate-restricted improvement step in the original space, and $T_{\text{final}} \le B_\mu T_{\text{move}}$ is the final refinement. Because L, S, and B are fixed by the configuration, the cost of Just Initialize is a bounded multiple of a single original-space sweep. The quantity it controls is therefore the number of sweeps that downstream optimization must perform, which is exactly the criterion introduced in Section 3.3.

C.2 Validity of the Compact-Space Objective

Fix a compact candidate $\tilde{\pi}$ and let its recovered initialization be drawn as $\pi_0 \sim Q_M(\cdot \mid \tilde{\pi})$, where Q_M is the sampling rule of Section 4. Write

$$Y(\tilde{\pi}, \xi) = C_I\big(\mathcal{R}_I(\pi_0; B, \xi)\big), \qquad J_B(\tilde{\pi}) = \mathbb{E}\big[Y(\tilde{\pi}, \xi)\big], \tag{46}$$

where $\mathcal{R}_I(\pi_0; B, \xi)$ is the solution returned by the downstream solver started at π_0 with budget B, and ξ collects any randomness in refinement. This is the compact-space counterpart of the initialization quality functional of Section 3.3: a lower J_B means a more effective initialization.

Proposition 3 (Unbiased estimation and concentration). Let $S \ge 1$ independent trials produce $Y_1, \dots, Y_S$ and let $\widehat{J}_B(\tilde{\pi}) = \frac{1}{S}\sum_{s=1}^{S} Y_s$. If the trials are identically distributed with finite mean $J_B(\tilde{\pi})$ and variance $\sigma^2(\tilde{\pi})$, then

$$\mathbb{E}\big[\widehat{J}_B(\tilde{\pi})\big] = J_B(\tilde{\pi}), \qquad \mathrm{Var}\big(\widehat{J}_B(\tilde{\pi})\big) = \frac{\sigma^2(\tilde{\pi})}{S}. \tag{47}$$

If in addition $Y_s \in [0, C_{\max}]$ almost surely, then for every $\varepsilon > 0$

$$\Pr\Big(\big|\widehat{J}_B(\tilde{\pi}) - J_B(\tilde{\pi})\big| \ge \varepsilon\Big) \le 2\exp\left(-\frac{2S\varepsilon^2}{C_{\max}^2}\right). \tag{48}$$

Proof. Linearity of expectation gives the first identity, and independence removes the covariance terms, so $\mathrm{Var}(\widehat{J}_B) = \frac{1}{S^2}\sum_s \mathrm{Var}(Y_s) = \sigma^2/S$. The tail bound is Hoeffding's inequality applied to the independent, bounded variables Y_s. □

Corollary 1 (High-probability candidate selection). Let $\mathcal{C}$ be the finite set of compact candidates evaluated by the compact-space search and let $S \ge 1$. For any $\alpha \in (0, 1)$, set

$$\varepsilon = C_{\max}\sqrt{\frac{1}{2S}\ln\frac{2|\mathcal{C}|}{\alpha}}. \tag{49}$$

With probability at least $1 - \alpha$, $\max_{\tilde{\pi}\in\mathcal{C}} |\widehat{J}_B(\tilde{\pi}) - J_B(\tilde{\pi})| \le \varepsilon$, and the selected candidate $\widehat{\tilde{\pi}} \in \arg\min_{\tilde{\pi}\in\mathcal{C}} \widehat{J}_B(\tilde{\pi})$ satisfies

$$J_B\big(\widehat{\tilde{\pi}}\big) \le \min_{\tilde{\pi}\in\mathcal{C}} J_B(\tilde{\pi}) + 2\varepsilon. \tag{50}$$

Proof. A union bound over $\mathcal{C}$ applied to Eq. (5) gives the uniform deviation event. On that event, with $\tilde{\pi}^\star = \arg\min_{\tilde{\pi}\in\mathcal{C}} J_B(\tilde{\pi})$,

$$J_B(\widehat{\tilde{\pi}}) \le \widehat{J}_B(\widehat{\tilde{\pi}}) + \varepsilon \le \widehat{J}_B(\tilde{\pi}^\star) + \varepsilon \le J_B(\tilde{\pi}^\star) + 2\varepsilon, \tag{51}$$

where the second inequality uses the definition of $\widehat{\tilde{\pi}}$. □

Corollary 1 justifies the compact-space update of Eq. (8): the candidate returned by the search is, with high probability, within 2ε of the best initialization quality in the evaluated set, and the deviation shrinks as $1/\sqrt{S}$. When refinement is deterministic and recovery is concentrated on a single candidate — the low-temperature setting used in the experiments — Y is deterministic, $\sigma^2 = 0$, a single trial evaluates J_B exactly, and the bound holds with $\varepsilon = 0$.

C.3 Decomposition of the Residual Error

Fix a compact structure $z \in \Omega(\tilde{I})$ and let

$$\Omega_z = \{\pi \in \Omega(I) : \Phi_s(\pi; M) = z\}, \qquad C_z^* = \min_{\pi\in\Omega_z} C_I(\pi), \tag{52}$$

with $\Omega_z \neq \emptyset$, and let $C_I^* = \min_{\pi\in\Omega(I)} C_I(\pi)$. Both minima are attained because the feasible spaces are finite and the costs are finite.

Proposition 4 (Error decomposition). For any recovered initialization $\pi_r \in \Omega_z$,

$$C_I(\pi_r) - C_I^* = \underbrace{\big(C_z^* - C_I^*\big)}_{\text{structural error}} + \underbrace{\big(C_I(\pi_r) - C_z^*\big)}_{\text{recovery error}}, \tag{53}$$

and both terms are nonnegative.

Proof. Adding and subtracting C_z^* on the left gives the identity. Since $\Omega_z \subseteq \Omega(I)$, we have $C_z^* \ge C_I^*$, and since $\pi_r \in \Omega_z$, we have $C_I(\pi_r) \ge C_z^*$; both terms are therefore nonnegative. □

Corollary 2 (Residual error after refinement). Let π_f be the solution returned by the descent refinement started at π_r. Since every accepted move strictly decreases the objective, $C_I(\pi_f) \le C_I(\pi_r)$, and hence

$$C_I(\pi_f) - C_I^* \le \big(C_z^* - C_I^*\big) + \big(C_I(\pi_r) - C_z^*\big). \tag{54}$$

Proof. The descent operator is non-increasing by definition of the acceptance rule, so $C_I(\pi_f) \le C_I(\pi_r)$; substituting Eq. (10) gives the bound. □

The two terms in Eq. (10) separate the responsibilities of the two stages of Just Initialize. The structural error depends only on which structure was selected, and is therefore controlled by instance compression and by the compact-space search; no amount of improvement within Ω_z can reduce it. The recovery error depends only on how accurately the recovered initialization instantiates the selected structure, and is the term that local refinement absorbs. This is what licenses aggressive compression: the compressed representation must carry the global structure that determines C_z^*, while local detail is not required to be accurate at compression time, because Corollary 2 charges it to the recovery term. It is also consistent with the component ablation of Section 5.4, where replacing compression degrades the final gap more than removing the compact-space search: the former raises the structural term, the latter leaves it essentially unchanged.

Remark 1 (Immediate cost does not determine initialization quality). J_B is not a function of $C_I(\pi_0)$. Two initializations of equal objective value can lie in different basins of the descent operator, so that from one the operator reaches a strictly better local minimum within budget B than from the other. Comparing initializations by their immediate objective is therefore not sound, which is why Section 3.3 compares them by the expected refined objective at a fixed solver and budget, and why the comparisons of Section 5.3 and Appendix E.6 hold the refinement procedure fixed.

D DATASETS AND PARAMETERS

D.1 DATASETS

The TSP collection contains 282 instances in three groups. The uniform group has 16 instances at each of 1K, 5K, 10K, 20K, 50K, and 100K nodes; the main comparison reports the 80 instances from 5K to 100K. The extended group follows the clustered, explosion, and implosion generators used by INViT (Fang et al., 2024); each distribution has 16 instances at 1K, 10K, and 100K nodes, for 144 instances in total. The third group contains 42 public benchmark instances, including TSPLIB95 (Reinelt, 1991) cases such as `pr2392`, `rl11849`, and `pla85900`, together with instances from the National TSP, VLSI, TSP Art, and USA collections. We retain their source coordinates and distance definitions: 39 use `EUC_2D` and three use `CEIL_2D`. The synthetic TSP groups use `EUC_2D` after coordinate normalization and integer export.

The CVRP collection has 96 uniform instances, with 16 at each of 1K, 5K, 10K, 20K, 50K, and 100K customers; the main comparison reports the 80 instances from 5K to 100K. VRPTW and PCTSP each contain 72 uniform instances: 16 at each of 1K, 2K, 5K, and 10K customers and eight at 20K. These sizes count customers; each CVRP, VRPTW, and PCTSP instance also has one depot. The VRPTW generator first creates 16 candidates at 20K. The published set contains the eight lowest-reference-cost instances among the 11 candidates for which the reference search converged; the retained instances were renumbered from 01 to 08.

D.2 PARAMETERS

Table 6 collects the generation settings. Here n is the number of cities for TSP or customers for the other problems, and i is the original generation index within a scale. Each seed initializes one instance independently. The selected VRPTW 20K instances retain their original seeds after renumbering. Synthetic coordinates lie in the unit square and are exported on a 10^6 integer grid. For CVRP, VRPTW, and PCTSP, the canonical files retain the continuous coordinates and attributes; the corresponding solver files contain the integer export. The TSP exporter deterministically resolves coordinate collisions introduced by rounding.

Table 6: Instance generation settings for the full collections. $U[a, b]$ denotes a continuous uniform distribution and $U_{\mathbb{Z}}[a, b]$ an integer uniform distribution.

Family	Sizes × instances	Sampling and fixed settings	Generation seed
TSP uniform	1K, 5K, 10K, 20K, 50K, 100K ×16	Independent uniform two-dimensional coordinates	$s_U(n, i)$
TSP extended	1K, 10K, 100K ×16 per distribution	Clustered, explosion, or implosion coordinates	$s_E(n, i)$
CVRP	1K, 5K, 10K, 20K, 50K, 100K ×16	Depot/customers $\sim U[0, 1]^2$; demand $\sim U_{\mathbb{Z}}[1, 9]$	$s_R(n, i)$
VRPTW	1K, 2K, 5K, 10K ×16; 20K ×8	CVRP-style coordinates and demands; depot window $[0, 3]$; service time 0.2	$s_R(n, i)$
PCTSP	1K, 2K, 5K, 10K ×16; 20K ×8	Depot/customers $\sim U[0, 1]^2$; prize $\sim U[0, 4/n]$; penalty $\sim U[0, 1.2/\sqrt{n}]$; quota 1	$s_R(n, i)$

The seed rules are $s_U(n, i) = 420000 + n + i$ for uniform TSP, $s_E(n, i) = b + n + i$ for extended TSP, and $s_R(n, i) = 20260909 + 97n + 1000003i$ for CVRP, VRPTW, and PCTSP. In s_E, b is 520000 for clustered, 620000 for explosion, and 720000 for implosion.

For the extended TSP distributions, the clustered generator assigns nodes to three Gaussian clusters with center diversity 10. The explosion and implosion generators draw a center and a radius in $[0.1, 0.5]$, then transform points inside the radius; explosion uses an exponential draw with rate 10, while implosion uses a normal draw. All generated TSP point sets are normalized before integer export. In the full CVRP collection, capacities are 250, 500, 1000, and 2000 for 1K, 5K, 10K, and 20K customers, respectively, and remain 2000 at 50K and 100K. VRPTW uses capacities 250, 500, 500, 1000, and 2000 at its five published scales. Its customer-window center is sampled between

the travel time from the depot and the latest start allowing a direct return; the half-width is sampled uniformly from $[0.1, 1.0]$ and clipped to the depot window. Instances are resampled until every customer admits a feasible depot–customer–depot trip.

The dataset parameters above describe instance generation; the following fixed values govern the frozen solvers used for the Ours results. Reference-value searches use their own budgets. The TSP solver grows path fragments toward a target length of 32, using 64 candidate neighbors per node and examining eight neighbors when extending a fragment. Its fragment graph retains 16 portal neighbors per fragment. The compact solver uses beam width 6 and at most eight coarse 2-opt passes. On the original tour, refinement uses 64 candidate neighbors and 16 iterated-search perturbations, selecting perturbation positions among the 80 longest edges. The initial local search allows up to 16 passes of 3-opt, and the subsequent Lin–Kernighan search has depth 3.

The CVRP solver uses a Clarke–Wright parameter of $\lambda = 1.4$ and a fragment demand threshold of $0.8Q$, where Q is vehicle capacity. It uses 32 candidate neighbors per customer and 16 portal neighbors per fragment. Compact-route improvement allows up to 200 inter-route relocate moves. Route-level 2-opt examines 16 candidate neighbors and accepts at most ten moves per route during compression and restoration, followed by at most one move per route in the final refinement stage. The final 2-opt* search examines eight candidate neighbors and accepts at most four moves.

E Extended Experiments

E.1 Non-uniform TSP instances

Experiment Design. To evaluate the generalization ability of Just Initialize (JI), we further conduct experiments on non-uniform TSP instances with three challenging spatial distributions: clustered, explosion, and implosion. Each distribution contains 16 instances at 1K, 10K, and 100K nodes. Unlike learning-based approaches that rely on training data, JI is completely training-free and does not require distribution alignment between training and testing instances. This experiment evaluates whether JI can provide effective initializations under diverse instance distributions.

Table 7: TSP results on clustered distributions (16 instances per size). Bold marks the lowest gap and solve time at each size, excluding Concorde; gray highlights Ours.

Method	1K			10K			100K		
	Obj.↓	Gap (%)	Time	Obj.↓	Gap (%)	Time	Obj.↓	Gap (%)	Time
LKH-3 (2.0 h)	14.22	–	–	40.67	–	–	120.22	–	–
Concorde	14.22	0.00	1.7 min	40.86	0.48	3.0 min	121.35	0.94	3.0 min
POMO (no aug.)	22.32	56.96	0.6 s	109.77	169.91	6.5 min		OOM	
POMO (×8 aug.)	20.96	47.41	3.1 s		OOM			OOM	
LEHD (greedy)	15.51	9.07	4.5 s	54.95	35.11	8.0 min		OOM	
INViT	15.23	7.09	8.9 s	44.10	8.43	1.5 min	129.49	7.71	18.6 min
SIL	15.35	7.97	3.9 s	86.58	112.89	12.0 min		OOM	
L2C-Insert ($I = 1000$)	14.59	**2.59**	57.7 s	41.65	4.15	1.0 min	126.81	5.49	1.6 min
DIFUSCO	16.22	14.04	11.8 s	45.85	12.76	34.3 s		OOM	
GLOP	15.11	6.31	1.4 s	43.47	6.89	1.8 s	128.67	7.03	1.2 min
UDC	14.62	2.87	3.4 s	42.37	4.20	3.7 s	125.61	4.48	1.2 min
H-TSP	15.41	8.41	1.2 s	44.46	9.31	9.3 s	130.92	8.90	1.7 min
TTPL	14.80	4.09	6.9 s	42.04	3.74	1.1 min	124.22	3.33	11.5 min
DeepACO ($T = 10$)	14.73	3.58	3.8 min		OOM			OOM	
DeepACO ($T = 2$)	14.87	4.61	53.2 s		OOM			OOM	
Ours	14.63	2.89	**0.1 s**	42.14	**3.62**	**2.0 s**	124.57	**3.11**	**37.2 s**

Results. As shown in Tables 7, 8, and 9, JI consistently achieves competitive solution quality across all three distributions and different scales, while maintaining substantially lower computational costs. For 10K-node instances, JI obtains low gaps within only a few seconds, and for 100K-node instances, it remains efficient with runtimes within tens of seconds. These results demonstrate

Table 8: TSP results on explosion distributions (16 instances per size). Bold marks the lowest gap and solve time at each size, excluding Concorde; gray highlights Ours.

Method	1K			10K			100K		
	Obj.↓	Gap (%)	Time	Obj.↓	Gap (%)	Time	Obj.↓	Gap (%)	Time
LKH-3 (2.0 h)	17.01	–	–	40.26	–	–	108.17	–	–
Concorde	17.01	0.00	2.4 min	40.43	0.42	3.0 min	109.33	1.07	3.0 min
POMO (no aug.)	25.91	52.34	0.6 s	117.39	191.57	6.5 min		OOM	
POMO (×8 aug.)	24.75	45.48	3.1 s		OOM			OOM	
LEHD (greedy)	17.91	5.32	4.1 s	53.09	31.88	8.0 min		OOM	
INViT	18.30	7.57	8.9 s	44.36	10.18	1.5 min	117.23	8.38	18.5 min
SIL	17.57	3.30	3.9 s	99.48	147.09	12.0 min		OOM	
L2C-Insert ($I = 1000$)	17.24	**1.40**	57.2 s	41.32	4.23	1.0 min	114.60	5.94	1.6 min
DIFUSCO	18.98	11.60	3.2 s	45.69	13.49	34.3 s		OOM	
GLOP	17.95	5.53	1.4 s	43.01	6.84	1.8 s	115.75	7.01	1.2 min
UDC	17.47	2.64	3.4 s	41.96	4.22	3.8 s	113.11	4.57	1.2 min
H-TSP	18.56	9.11	1.3 s	44.19	9.75	9.7 s	119.52	10.49	1.7 min
TTPL	18.01	5.87	6.9 s	42.73	6.13	1.1 min	113.26	4.71	11.6 min
DeepACO ($T = 10$)	17.57	3.28	3.7 min		OOM			OOM	
DeepACO ($T = 2$)	17.68	3.96	50.9 s		OOM			OOM	
Ours	17.49	2.83	**0.1 s**	41.77	**3.75**	**1.9 s**	**112.17**	**3.76**	**36.7 s**

Table 9: TSP results on implosion distributions (16 instances per size). Bold marks the lowest gap and solve time at each size, excluding Concorde; gray highlights Ours.

Method	1K			10K			100K		
	Obj.↓	Gap (%)	Time	Obj.↓	Gap (%)	Time	Obj.↓	Gap (%)	Time
LKH-3 (2.0 h)	20.29	–	–	63.25	–	–	189.57	–	–
Concorde	20.29	0.00	1.7 min	63.53	0.44	3.0 min	191.64	1.10	3.0 min
POMO (no aug.)	28.85	42.20	0.6 s	119.39	88.76	6.5 min		OOM	
POMO (×8 aug.)	28.39	39.94	3.1 s		OOM			OOM	
LEHD (greedy)	21.05	3.75	4.2 s	81.98	29.61	8.0 min		OOM	
INViT	21.50	5.94	8.9 s	68.00	7.53	1.5 min	203.77	7.49	18.5 min
SIL	21.26	4.78	4.0 s	90.25	42.68	12.0 min		OOM	
L2C-Insert ($I = 1000$)	20.45	**0.78**	57.3 s	64.75	**2.39**	1.0 min	199.28	5.12	1.6 min
DIFUSCO	22.49	10.85	3.2 s	70.68	11.75	34.3 s		OOM	
GLOP	21.33	5.09	1.4 s	67.37	6.50	1.9 s	202.59	6.87	1.2 min
UDC	20.86	2.76	3.4 s	65.75	3.96	3.8 s	197.83	4.35	1.2 min
H-TSP	21.92	7.99	1.2 s	68.98	9.06	9.3 s	208.96	10.23	1.7 min
TTPL	21.08	3.88	6.9 s	65.99	4.33	1.1 min	196.36	3.84	11.6 min
DeepACO ($T = 10$)	20.91	3.02	3.9 min		OOM			OOM	
DeepACO ($T = 2$)	21.01	3.55	53.4 s		OOM			OOM	
Ours	20.88	2.88	**0.1 s**	65.48	3.52	**1.9 s**	196.66	**3.74**	**38.5 s**

that JI can be effectively combined with downstream refinement solvers without additional training or distribution-specific adaptation.

Analysis. The stable performance across clustered, explosion, and implosion distributions demonstrates the strong generalization capability of JI. Although learning-based approaches can exploit distribution-specific patterns through training, their effectiveness may depend on the similarity between training and testing distributions. In contrast, JI does not require such distribution alignment; it directly identifies promising regions from each instance through geometric compression and generates solver-compatible initializations. Therefore, JI provides a distribution-agnostic initialization mechanism, enabling efficient refinement across diverse routing scenarios.

E.2 TSPLIB95 INSTANCES

Experiment Design. To further evaluate the generalization ability of Just Initialize (JI) on real-world routing instances, we conduct experiments on 32 TSPLIB95 instances with different scales, ranging from 1K to 85K nodes. These instances contain diverse spatial structures and are widely used as standard benchmarks for evaluating large-scale TSP solvers. JI is directly applied without additional training or instance-specific adaptation, demonstrating its ability to generate effective initializations for unseen routing scenarios.

Results. Table 10 reports the relative optimality gaps on TSPLIB95 instances. JI achieves the lowest average gap among all compared methods, reducing the average gap to 3.54%. Moreover, JI obtains the best-reported gap on 19 out of 32 instances, while successfully solving all instances in the benchmark. Compared with existing methods, JI maintains consistently low gaps across different scales, including large instances with tens of thousands of nodes.

Analysis. The results demonstrate that JI can generalize beyond synthetic distributions to diverse real-world routing instances. Unlike learning-based methods that rely on training distributions to capture instance patterns, JI does not require such distribution alignment. Instead, JI extracts the underlying geometric structure of each instance through compression and provides a promising initialization for downstream optimization. This enables JI to remain effective across different instance scales and spatial characteristics, validating the robustness of the proposed initialization mechanism.

Table 10: Relative gaps (%) on TSPLIB95 instances. Bold marks the lowest reported gap in each row; gray highlights Ours. OOM marks out-of-memory on a 24 GB GPU.

Instance	Scale	LEHD	TTPL	H-TSP	UDC	GLOP	JI+Refine (Ours)
pr1002	1,002	4.43%	**0.73%**	7.19%	2.62%	5.21%	4.41%
u1060	1,060	10.01%	2.54%	10.84%	3.55%	4.82%	**2.08%**
vm1084	1,084	5.42%	**1.44%**	12.70%	3.95%	5.61%	3.29%
pcb1173	1,173	7.95%	**1.72%**	8.51%	5.23%	7.48%	4.36%
d1291	1,291	13.46%	6.22%	16.11%	8.32%	9.59%	**2.47%**
rl1304	1,304	8.14%	4.16%	14.67%	6.94%	10.00%	**2.82%**
rl1323	1,323	9.27%	3.89%	16.05%	5.28%	11.58%	**2.53%**
nrw1379	1,379	15.43%	**0.98%**	6.47%	2.20%	5.08%	3.73%
fl1400	1,400	18.10%	16.99%	19.64%	4.65%	5.50%	**3.48%**
u1432	1,432	7.93%	**2.15%**	8.95%	2.65%	5.83%	4.88%
fl1577	1,577	14.98%	11.01%	23.07%	7.52%	9.69%	**6.66%**
d1655	1,655	13.68%	6.46%	11.69%	4.23%	6.93%	**2.88%**
vm1748	1,748	10.11%	8.25%	10.78%	3.95%	5.51%	**2.77%**
u1817	1,817	8.98%	4.18%	11.94%	5.43%	10.52%	**3.24%**
rl1889	1,889	7.50%	4.91%	12.40%	7.57%	11.31%	**2.86%**
d2103	2,103	11.97%	8.01%	15.80%	8.80%	14.91%	**2.50%**
u2152	2,152	9.27%	**2.47%**	14.81%	5.85%	9.68%	4.10%
u2319	2,319	4.14%	**0.22%**	2.20%	1.04%	2.19%	2.26%
pr2392	2,392	12.31%	**2.85%**	18.37%	4.86%	7.14%	2.88%
pcb3038	3,038	13.37%	5.26%	8.45%	4.71%	7.98%	**3.88%**
fl3795	3,795	17.51%	34.75%	19.09%	5.96%	9.51%	**5.65%**
fnl4461	4,461	18.95%	**2.39%**	6.44%	3.14%	5.97%	3.17%
rl5915	5,915	24.18%	4.57%	16.11%	9.90%	13.44%	**2.78%**
rl5934	5,934	24.11%	8.31%	16.39%	8.33%	11.64%	**3.54%**
pla7397	7,397	40.94%	8.61%	10.74%	6.67%	7.73%	**3.25%**
rl11849	11,849	37.51%	5.27%	15.14%	7.85%	11.59%	**3.89%**
usa13509	13,509	70.17%	3.85%	10.11%	4.44%	7.16%	**3.83%**
brd14051	14,051	OOM	4.48%	9.46%	**3.52%**	5.78%	3.67%
d15112	15,112	OOM	**3.20%**	7.99%	3.51%	6.06%	3.41%
d18512	18,512	OOM	**2.28%**	6.88%	3.65%	6.48%	3.34%
pla33810	33,810	OOM	**4.10%**	10.23%	8.31%	10.23%	4.77%
pla85900	85,900	OOM	4.95%	9.91%	6.65%	9.19%	**3.88%**
Solved#		27/32	32/32	32/32	32/32	32/32	32/32
Best#		0/32	12/32	0/32	1/32	0/32	**19/32**
Avg. gap		16.29%	5.66%	12.16%	5.35%	8.17%	**3.54%**

E.3 CVRPLIB INSTANCES

Experiment Design. To further evaluate the applicability of Just Initialize (JI) beyond TSP, we conduct experiments on CVRPLIB instances with diverse scales and customer distributions. The benchmark contains 11 instances ranging from 1K to 30K customers, where additional capacity constraints are introduced compared with TSP. JI is directly applied without additional training or problem-specific adaptation, evaluating whether the proposed initialization mechanism can generalize to different routing formulations.

Results. Table 11 reports the relative gaps on CVRPLIB instances. JI successfully solves all 11 instances and achieves the lowest average gap among all compared methods, reaching an average gap of 7.77%. Moreover, JI obtains the best-reported gap on 8 out of 11 instances, demonstrating consistent performance across different problem scales. These results show that JI maintains competitive solution quality while remaining applicable to large-scale vehicle routing problems with up to 30K customers.

Analysis. The results demonstrate that JI can generalize from TSP to more complex vehicle routing scenarios with additional constraints. Unlike learning-based approaches that require problem-specific training to capture routing patterns, JI does not require such distribution alignment. Instead, JI extracts the geometric structure of each instance through compression and generates effective initializations for downstream optimization. This indicates that the proposed initialization mechanism is not limited to a specific routing formulation, but provides a general strategy for improving large-scale combinatorial optimization.

Table 11: Relative gaps (%) on CVRPLIB instances. Bold marks the lowest reported gap in each row; gray highlights Ours.

Instance	Scale	HGS	OR-Tools	LEHD	L2C	TTPL	UDC	GLOP	Ours
X-n1001-k43	1,000	**3.01%**	9.59%	7.55%	7.53%	6.01%	6.49%	88.26%	6.10%
Leuven1	3,000	5.42%	5.42%	16.36%	7.07%	6.48%	9.64%	105.31%	**5.35%**
Leuven2	4,000	**9.00%**	12.36%	33.12%	25.76%	24.06%	44.00%	45.29%	12.52%
Antwerp1	6,000	5.49%	7.25%	15.17%	6.23%	6.69%	9.59%	130.39%	**4.47%**
Antwerp2	7,000	11.27%	12.41%	21.36%	20.00%	13.54%	21.08%	63.61%	**9.56%**
Ghent1	10,000	6.74%	7.11%	27.64%	11.15%	10.64%	12.36%	146.15%	**3.96%**
Ghent2	11,000	**10.43%**	14.92%	39.85%	30.34%	16.08%	184.76%	45.79%	11.36%
Brussels1	15,000	9.54%	9.89%	OOM	15.41%	9.14%	17.13%	139.73%	**6.17%**
Brussels2	16,000	12.52%	16.28%	OOM	29.87%	15.03%	71.11%	55.88%	**10.80%**
Flanders1	20,000	6.43%	6.47%	OOM	18.07%	10.05%	13.91%	92.53%	**4.31%**
Flanders2	30,000	14.55%	OOT	OOM	45.81%	28.43%	309.02%	45.60%	**10.87%**
Solved#		11/11	10/11	7/11	11/11	11/11	11/11	11/11	11/11
Best#		3/11	0/11	0/11	0/11	0/11	0/11	0/11	**8/11**
Avg. gap		8.58%	10.17%	23.01%	19.75%	13.29%	63.55%	87.14%	**7.77%**

E.4 CASE STUDY

Experiment Design. To investigate how the compact-space solve stage affects the quality of the generated initialization, we perform a case study on a 100K-node uniform TSP instance. During the compact-space optimization process, we sample 24 intermediate solve states with different optimization budgets. Each state is independently recovered to the original space and refined using the same refinement procedure and budget. The resulting solutions are evaluated by their final GAP, allowing us to analyze whether later compact-space states provide better initialization quality.

Table 12: Intermediate solve states on the 100K uniform TSP instance. x denotes cumulative compact-space 2-opt evaluations (millions), and GAP is measured after recovery and refinement.

Point	1	2	3	4	5	6	7	8	9	10	11	12
x	0.00	5.13	23.47	24.95	27.88	44.78	48.84	51.61	51.76	71.21	74.72	76.97
GAP (%)	3.684	3.684	3.674	3.671	3.683	3.662	3.653	3.645	3.657	3.661	3.643	3.646
Point	13	14	15	16	17	18	19	20	21	22	23	24
x	91.20	93.77	101.18	102.97	103.52	104.91	107.66	111.03	123.94	146.85	151.44	162.71
GAP (%)	3.631	3.641	3.648	3.634	3.647	3.657	3.652	3.638	3.616	3.597	3.585	3.564

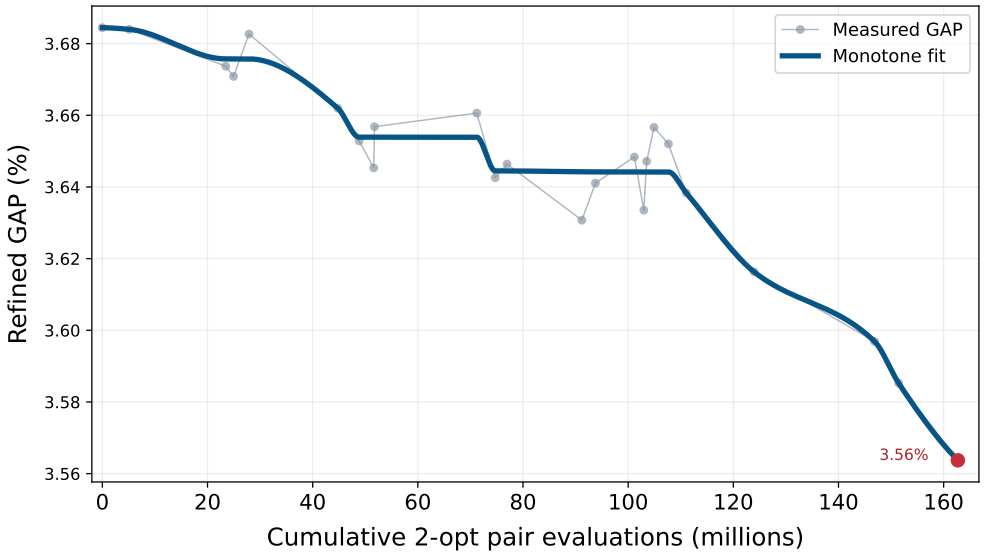

Figure 3: Solve-state convergence of JI on the 100K uniform TSP instance. Gray markers denote measured GAP values, and the curve shows the overall decreasing trend.

Results. As shown in Table 12 and Figure 3, the refined GAP consistently decreases as the compact-space optimization progresses. The GAP decreases from 3.684% at the initial checkpoint to 3.564% at the final checkpoint, despite small local fluctuations during intermediate stages. This demonstrates that additional optimization in the compact space leads to increasingly effective initializations for downstream refinement.

Analysis. The observed trajectory validates the motivation of JI: the compact-space solve stage is not directly optimizing the final solution, but identifying regions with stronger refinement potential in the original solution space. Although compact-space improvements do not guarantee monotonic decreases after recovery, later solve states generally produce better refined solutions under the same refinement budget. These results indicate that JI effectively guides the search toward more favorable regions before applying expensive fine-grained optimization.

E.5 Convergence from Matched Starting Gaps

Experiment Design. To evaluate whether JI provides better refinement potential beyond simply achieving a lower initial GAP, we conduct a matched initialization study. Three representative initialization strategies, including randomized nearest-neighbor (RNN), Hilbert space-filling-curve ordering (Hilbert), and minimum-spanning-tree traversal (MST), are selected with initial GAPs matched

to JI within 0.05 percentage points. All initializations undergo the same 64-neighbor 2-opt refinement procedure, and their final GAPs and refinement times are compared.

Table 13: Final GAP and refinement time under matched initializations.

	Final relative gap (%)				Search time (s)			
Scale	JI	RNN	Hilbert	MST	JI	RNN	Hilbert	MST
1K	**8.6338**	10.7757	12.3769	10.5459	0.016	0.015	0.024	0.022
5K	**9.2353**	11.3989	13.4103	11.0869	0.156	0.150	0.194	0.198
10K	**9.2395**	11.2762	13.4725	11.1091	0.092	0.092	0.180	0.169
20K	**9.2093**	11.3263	13.6263	11.1296	0.224	0.211	0.441	0.409
50K	**9.2136**	11.1755	13.5666	11.1678	0.792	0.836	1.843	1.394
100K	**9.1038**	11.0169	13.5540	11.1422	3.276	3.104	6.338	4.274

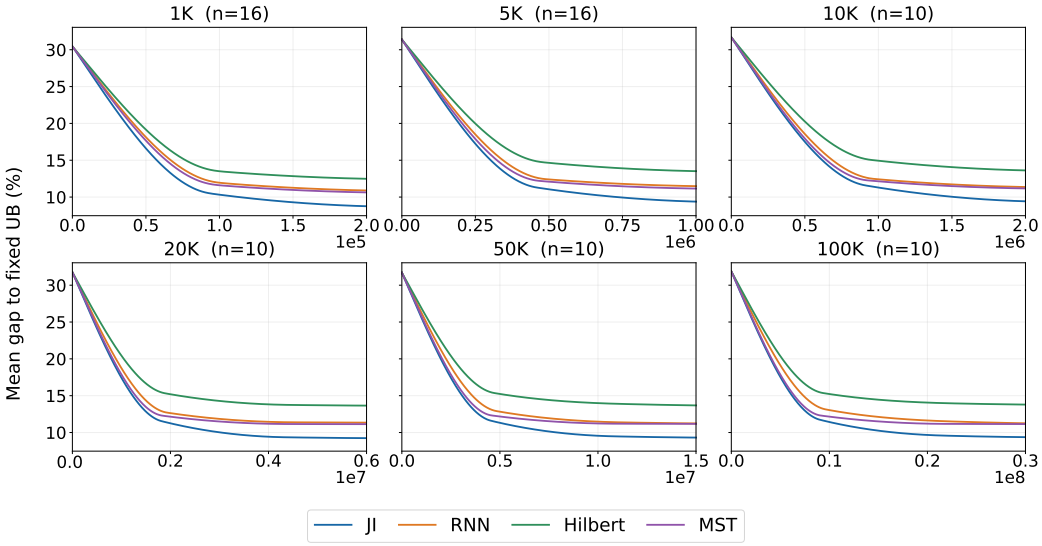

Figure 4: GAP convergence trajectories during 2-opt refinement under matched initializations across six TSP scales.

Results. Table 13 and Figure 4report the refinement performance under matched initial GAPs. JI consistently achieves lower final GAPs across all six scales while using comparable refinement procedures. The convergence trajectories further show that JI maintains a faster improvement trend during optimization and reaches better final solutions than alternative initializations.

Analysis. The matched-gap comparison demonstrates that the effectiveness of JI does not come from simply providing a better initial objective value. Instead, JI generates initial solutions located in regions with stronger downstream refinement potential. Under comparable starting quality and identical optimization procedures, JI enables local search to reach better solutions, supporting our hypothesis that initialization quality should be evaluated by refinement effectiveness rather than immediate solution quality.

E.6 Greedy Starts and Search Efficiency

Experiment Design. To evaluate whether JI improves optimization efficiency through better initialization, we compare JI with a nearest-neighbor greedy initialization on 96 uniform TSP instances across six scales. Both initializations are refined using the same deterministic 64-neighbor 2-opt procedure. We evaluate the final solution quality, refinement time, convergence trajectory, and the

number of candidate evaluations required to reach a target GAP of 10%. This controlled comparison isolates the effect of initialization quality from the optimization procedure.

Table 14: Final GAP and 2-opt refinement time using greedy and JI initializations across different TSP scales.

Scale	Greedy		JI	
	GAP(%)	Time(s)	GAP(%)	Time(s)
1K	8.4431	0.012	7.5405	0.012
5K	8.0722	0.072	7.7279	0.069
10K	8.2994	0.161	7.6362	0.145
20K	7.9563	0.373	7.6374	0.330
50K	7.9596	1.068	7.6603	0.785
100K	7.7531	2.826	7.6640	1.879

Table 15: Search budgets for reaching a 10% relative GAP under different initializations.

Scale	JI	Greedy	Saving(%)
1K	60,532	100,415	39.72
5K	308,912	434,586	28.92
10K	585,714	1,000,608	41.46
20K	1,201,814	1,837,954	34.61
50K	2,972,090	4,640,830	35.96
100K	5,940,941	8,894,603	33.21

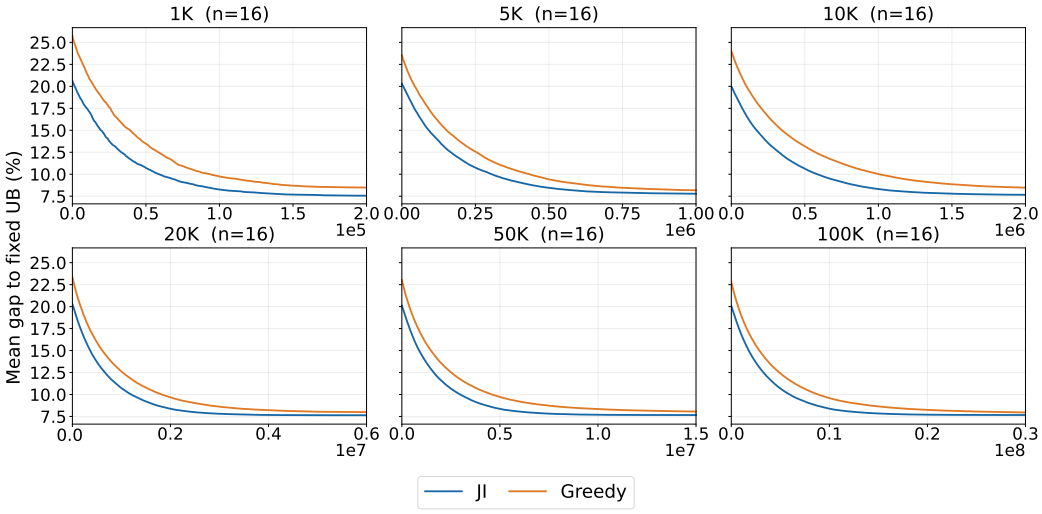

Figure 5: GAP convergence trajectories during 2-opt refinement with greedy and JI initializations across six TSP scales.

Results. Tables 14 and 15, together with Figure 5, show the refinement behavior of JI and greedy initialization. JI consistently achieves lower final GAPs across all scales while using comparable or lower refinement time. The convergence trajectories further demonstrate that JI improves more rapidly during early optimization and maintains a lower GAP throughout the search process. Moreover, JI reaches the 10% GAP target with fewer candidate evaluations, reducing the required search budget by 29–41% across different scales.

Analysis. These results indicate that the improvement of JI comes from providing more optimization-friendly initial solutions rather than modifying the refinement algorithm itself. Although greedy initialization produces feasible solutions with reasonable quality, it does not explicitly exploit the global structure of the solution space. In contrast, JI identifies more promising regions through compact-space optimization, allowing downstream refinement to spend fewer evaluations exploring unfavorable regions. This supports our hypothesis that improving initialization can reduce the computational burden of large-scale optimization by guiding search toward better solution regions.