ViTSP: A Vision Language Models Guided Framework for Large-Scale Traveling Salesman Problems

Anonymous authors
Paper under double-blind review

Abstract

Solving Traveling Salesman Problem (TSP) is NP-hard yet fundamental for wide real-world applications. Classical exact methods face challenges in scaling, and heuristic methods often require domain-specific parameter calibration. While learning-based approaches have shown promise, they suffer from poor generalization and limited scalability due to fixed training data. This work proposes *ViTSP*, a novel framework that leverages pre-trained vision language models (VLMs) to visually guide the solution process for large-scale TSPs. The VLMs function to identify promising small-scale subproblems from a visualized TSP instance, which are then efficiently optimized using an off-the-shelf solver to improve the global solution. *ViTSP* bypasses the dedicated model training at the user end while maintaining effectiveness across diverse instances. Experiments on real-world TSP instances ranging from 1k to 88k nodes demonstrate that *ViTSP* consistently achieves solutions with average optimality gaps below 0.2%, outperforming existing learning-based methods. Under the same runtime budget, it surpasses the best-performing heuristic solver, LKH-3, by reducing its gaps by 12% to 100%, particularly on very-large-scale instances with more than 10k nodes. Our framework offers a new perspective in hybridizing pre-trained generative models and operations research solvers in solving combinatorial optimization problems, with practical implications for integration into more complex logistics systems. The code is available at `https://anonymous.4open.science/r/ViTSP_codes-6683`.

1 Introduction

The Traveling Salesman Problem (TSP) is a fundamental combinatorial optimization (CO) problem with broad real-world applications, including transportation, logistics, and chip design (Applegate, 2006; Yin et al., 2023). Efficiently solving TSPs not only yields economical and societal benefits across those domains but also informs the development of solution strategies for other CO problems. The operations research (OR) community has developed numerous exact and heuristic algorithms to address this NP-hard problem (Davendra, 2010). However, exact methods often **struggle to produce high-quality solutions as the problem size increases**. Heuristic algorithms offer faster approximate solutions, yet their effectiveness **depends on domain-specific knowledge and careful calibration of instance-specific parameters**.

Advances of machine learning (ML) have led to various learning-based approaches for solving TSPs (referred to as *neural solvers*), including end-to-end models for solution construction (Vaswani, 2017; Jin et al., 2023; Sun & Yang, 2023; Li et al., 2025) and learned neural strategies for local improvement (Zong et al., 2022; Cheng et al., 2023; Ye et al., 2023; 2024b; Zheng et al., 2025). These methods are shown to shorten the computation time and maintain good solutions for in-distribution, small-scale instances (nodes < 1,000) (Wu et al., 2024). **However, they suffer from poor generalization and limited scalability as soon as the real-world problem deviates from the training data**.

The surge of pre-trained large language models (LLMs) and vision language models (VLMs) has raised interest in their potential for tackling optimization problems. Their efforts mainly focused on end-to-end construction of text-based solutions (Yang et al., 2023; Elhenawy et al., 2024) or on heuristic designs (Ye et al., 2024a; Liu et al., 2024) that rely solely on textual information of TSP instances. While these studies open new perspectives on using generative models to rethink optimization, **their approaches fall short of demonstrating reliable performance on large-scale practical TSPs** (Khan & Hamad, 2024)

In this study, we reconsider how recent advances of pre-trained generative models can effectively complement established OR techniques for solving varying large-scale TSP instances to facilitate broader application domains. We leverage pre-trained VLMs to provide adaptive decomposition heuristics that integrate directly into the optimization routine. Effective decomposition must account not only for spatial locality but also for combinatorial neighborhoods that help escape local optima. VLMs are well-suited for this task, as they can interpret instance-specific spatial structures by treating TSP instances as 2D images, enabling more informed selection of subproblems. Importantly, unlike ML approaches that require domain-specific training and graph embeddings, VLMs offer generic reasoning capabilities that avoid costly data collection or retraining. Furthermore, the subproblems, being smaller than their original TSP, can be reliably solved by exact solvers, avoiding performance degradation often experienced in learned neural solvers (Joshi et al., 2022; Wu et al., 2024).

We propose *ViTSP*, a Vision-guided framework for solving large-scale TSP. In *ViTSP*, VLMs guide the optimization process by identifying meaningful subproblems from visualized TSP instances, while an off-the-shelf solver continuously refines those subproblems. *ViTSP* orchestrates the two modules asynchronously to accommodate input/output (I/O) intensive VLMs and CPU-intensive solvers. On unseen TSPLIB (Reinelt, 1991) instances ranging from 1k to 88k, *ViTSP* finds the global optimum in 11 out of 33 instances and outperforms baseline learning-based methods, whose performance degrades significantly compared to their reported in-distribution results. Compared to the best-performing heuristic solver, LKH-3, *ViTSP* converges to superior solutions under the same time budget and reduces optimality gaps by 12% to 100%. Our key contributions in this study are summarized below:

1. We propose a vision-guided solution framework *ViTSP* that has strong performance to adapt to TSP instances with varying compositions.
2. Our approach leverages pre-trained VLMs to visually derive decomposition heuristics while bypassing the costly and time-consuming training and data curation for edge deployment.
3. We conduct experiments on (very-)large-scale instances to validate the effectiveness of *ViTSP*. The ablation studies further underpinned ViTSP's ability to perform principled guidance. To the best of our knowledge, our work presents one of the most comprehensive evaluations of real-world TSPLIB instances with $N > 1000$, whereas few prior works reported sufficient results at this scale.

2 Related works

Existing approaches to solving large-scale TSP can be categorized into three primary schemes: (1) OR approaches, (2) learning-based approaches, and (3) LLM/VLM-based approaches. We briefly review these works in this section, and we supplement the detailed discussion in the Appendix B.

2.1 OR approaches

Exact algorithms typically require explicit mathematical formulations and search for exact solutions via branch and bound procedures (Laporte, 1992; Wolsey, 2020). Off-the-shelf exact solvers, such as Concorde, Gurobi, and OR-Tools, have the potential to reach global optimality. Among them, Concorde remains the state-of-the-art (SOTA), using specialized rules to speed up the search process. However, the computation of exact solvers becomes intractable as the problem size increases.

Heuristic algorithms, such as farthest insertion (Rosenkrantz et al., 1974), genetic algorithm (Holland, 1992), and Lin-Kernighan-Helsgaun-3 (LKH-3) (Helsgaun, 2017), iteratively refine solutions based on hand-crafted rules. LKH-3 is regarded as the SOTA in solving TSPs. However, LKH-3 relies on tunable parameters, such as the number of total runs and candidate edges. Without domain knowledge and instance-specific calibration, achieving strong performance is often non-trivial. According to Adenso-Díaz & Laguna (2006), only about 10% of the effort in developing and testing heuristics or metaheuristics goes into designing it, with the remaining 90% spent in parameter tuning.

2.2 Learning-based approaches

Learning-based approaches for solving CO problems have gained wide attention since the surge of deep learning. These works commonly employ graph neural networks to embed TSP instances. The networks are trained using either supervised learning, which requires high-quality solutions from exact or heuristic methods as labels, or reinforcement learning, which relies on extensive trial-and-error (Fu et al., 2021). Existing works mainly deploy trained networks under two paradigms.

End-to-end construction. This paradigm seeks to learn a policy to directly construct a solution, using either autoregressive or non-autoregressive (heatmap-based) schemes. The autoregressive scheme trains attention-based neural networks (Vaswani, 2017; Kwon et al., 2020; Jin et al., 2023). The network sequentially constructs solutions by outputting one node at a time, with previous outputs incorporated into the network to guide the generation of subsequent nodes (Deudon et al., 2018; Kool et al., 2019). In contrast, the non-autoregressive approaches, such as Qiu et al. (2022); Sun & Yang (2023); Li et al. (2023; 2025), estimate the likelihood of connecting each edge between nodes and construct the solution in one shot.

Local improvement. This paradigm iteratively updates solutions using learned policies in two ways. First, it repeatedly selects partial problems or decomposes the whole problem into separate subproblems, and reconstructs them using a separate neural solver or an OR solver (Li et al., 2021; Fu et al., 2021; Zong et al., 2022; Cheng et al., 2023; Pan et al., 2023; Ye et al., 2024b; Zheng et al., 2025). Second, it learns to predict stepwise searching to assist existing OR algorithms (Xin et al., 2021; Hudson et al., 2022; Zheng et al., 2022; Wu et al., 2022; Ye et al., 2023; Ma et al., 2023).

Despite promising in-distribution performance presented, specialized learning-based approaches often fall short in handling out-of-distribution (OOD) instances since their neural networks were trained on fixed datasets (Li & Zhang, 2025). Therefore, they often fail to compete with the reliability of established OR solvers. Such limitations hinder their applicability at a practical scale. In fact, few studies have evaluated OOD performance on open-source TSPLIB instances with more than 5,000 nodes (Reinelt, 1991), limiting our understanding of their robustness in real-world scenarios.

2.3 LLMs/VLMs-based approaches

The surge of pre-trained LLMs/VLMs has drawn wide attention for optimization problems, including TSP. Yang et al. (2023) treats node coordinates as text input and prompts LLMs to output solutions, but the resulting solutions exhibit large optimality gaps even on small instances (i.e., $N = 50$). Another approach in Elhenawy et al. (2024) uses TSP images and relies on the VLMs to read node indices to construct tours. This design has limited scalability, as densely distributed nodes make it difficult for the VLMs to correctly recognize node indices. Liu et al. (2024) prompts LLMs for automatic heuristics design and translates them into code. However, their text-only framework overlooks instance-specific spatial structure. Consequently, their standalone strategy exhibits large variance in optimality gaps when applied to varying TSPs, limiting their reliability for practical use.

3 Methods

We reposition VLMs from unreliable end-to-end solvers to practical complements that can be integrated with established OR tools in building scalable optimization routines. In contrast to graph-based neural solvers, which demand extensive training and still struggle to generalize, we leverage the generic multi-modal reasoning of VLMs to process TSP as an image, enabling them to interpret spatial structures and provide adaptive decompositions without task-specific training.

Building on this motivation, we propose the *ViTSP* framework (Figure 1), which integrates VLM guidance into the optimization pipeline through three key modules: solution initialization, visual selection, and subproblem optimization. Starting from an initial solution for a given TSP instance (Sec 3.2 in Figure 1), VLMs identify box coordinates that delineate promising subproblems for further refinement based on the visualized TSP solution (Sec 3.3 in Figure 1). The exact solver iteratively optimizes returned subproblems to improve global solutions (Sec 3.4 in Figure 1). Iteratively solving subproblems allows certain subproblems to be optimized combinatorially under varying neighborhoods to escape local optima.

Since visual selection and subproblem optimization have distinct computational overheads, *ViTSP* coordinates their outputs asynchronously via a shared global solution, trajectory history, and subproblem queue to minimize the idle time in the subproblem optimization (Sec 3.5 in Figure 1).

The key advantages that ensure the effectiveness and scalability of our approach are threefold:

1. We leverage the pre-trained models to provide a decomposition-like heuristic rather than an error-prone end-to-end solution construction. As strong generalists for user-specified tasks, these models eliminate the need for *ad hoc* (re-)training during real-world deployment.

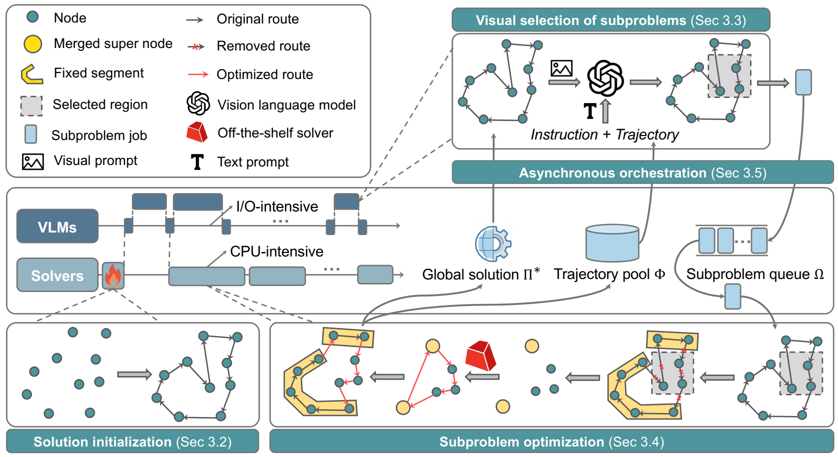

Figure 1: The vision-guided framework (*ViTSP*) for large-scale TSP, where pre-trained VLMs and off-the-shelf solvers are asynchronously coordinated to identify and optimize subproblems, respectively.

2. Visually guiding the selection of box regions is scalable, as they rely on spatial coordinates that remain consistent even as the TSP instance grows in size.
3. We reformulate the identified subproblems as standard TSPs. This allows us to harness the robust exact solvers, guaranteeing high-quality improvement to the global solution.

3.1 Preliminary: Traveling Salesman Problem (TSP)

We briefly introduce the TSP in this section and provide detailed notations and descriptions in the Appendix A. A TSP is characterized by a list of nodes and the corresponding coordinate sets or the distance matrix. The goal of TSP is to find an optimal tour Π^* that departs from an initial node, visits each node exactly once, and returns to the starting node, which minimizes the total distance traveled $L(\Pi^*)$. Notably, when the distance between two nodes is identical in both directions, the problem is known as symmetric TSP (**STSP**). In contrast, asymmetric TSP (**ATSP**) allows for different distances between certain node pairs in opposite directions.

3.2 Solution initialization

The *ViTSP* is warm-started using heuristic solvers. Critically, to effectively handle OOD instances, *ViTSP* avoids extensive parameter tuning and instead uses the default settings of the solver. This eliminates the dependency on prior domain knowledge that would otherwise hinder *ViTSP*'s adaptability to varying instances.

3.3 Visual selection of subproblems by VLMs

In the visual selection module $F_{\text{selector}}(\cdot)$, we prompt VLMs to select box regions and then formulate them as subproblems. We provide an overview of multimodal prompts and expected outputs, and defer the detailed example prompts in Appendix C. The pseudocode for the visual selection process is detailed in Algorithm 1 in Appendix D.

Visual prompts. Given a TSP instance, we plot its nodes and their current connections on an image based on their 2D coordinates and the global solution Π. This image is input to VLMs as a visual prompt. An example of such visual input is illustrated in Figure 4 of Appendix C.

Textual prompts. We specify three types of information as textual inputs to an VLM: (1) *meta-instructions* I, detailing the subproblem selection task description and the expected output format; (2) *selection trajectories* $\Phi = \{\phi_1, \phi_2, ...\}$, served as memory to address the stateless nature of API-based VLM calls. Each trajectory entry ϕ_i includes a selected subproblem, the number of nodes within this subproblem, the solution gain through optimization, and the solver's runtime. The selection trajectories from earlier steps reveal instance-specific structures as the solving progresses, which informs VLMs to make better subsequent selections (Yang et al., 2023; Laskin et al., 2023; Monea et al., 2024; Moeini et al., 2025); (3) *Pending subproblems* $\Omega = \{\omega_1, \omega_2, ...\}$, indicating identified yet unsolved subproblems that remain in the queue. This avoids duplicated subproblems selected by VLMs.

Image-level output. The VLM is prompted to generate a quadruple $C = (x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}})$ as a textual response. This quadruple represents the coordinates of a box region at the image level. As generative models, VLMs can be flexibly tailored to generate Q coordinate sets $C_1, C_2, \ldots, C_Q$ per response (where $Q \geq 2$). We will leverage these multiple-subproblem outputs during module orchestration as described in Section 3.5.

Forming a subproblem. Given the current global solution Π, connections to the covered nodes within a given box region are removed, leading to a list of free nodes $W = \{w_1, w_2, ..., w_{|W|}\}$. The remaining connected nodes outside of the box form segments $K = \{(\pi_1^1, \ldots, \pi_{c_1}^1), \ldots, (\pi_1^{|K|}, \ldots, \pi_{c_{|K|}}^{|K|})\}$, where $(\pi_1^k, \ldots, \pi_{c_k}^k)$ denotes the k-th segment containing $|c_k|$ connected nodes; and π_1^k and $\pi_{c_k}^k$ denote the starting and ending nodes in the k-th segment, respectively. As a result, the visual selection module produces a subproblem $\omega = (W, K)$.

Zoom-in reselection. *ViTSP* employs a zoom-in reselection to ensure scalability on very large-scale TSPs. Because such instances often exhibit highly dense node distributions, making connections in the initial visualization less discernible. To address this, a second round of selection is performed on the initially identified subregion bounded by C. If the number of nodes $|W|$ covered by the subregion exceeds a predefined threshold α, the VLM zooms into it to examine the finer-grained pattern and identify a new quadruple C'.

3.4 Subproblem optimization

Rather than training a dedicated neural solver to optimize selected subproblems separately, as in prior works that adopt the local improvement paradigm (Cheng et al., 2023; Pan et al., 2023; Zheng et al., 2025), we transform the formulated subproblem $\omega = (W, K)$ into a standard symmetric TSP (STSP). As existing exact solvers are primarily designed for STSP, this reformulation allows us to leverage solvers to obtain a globally feasible solution with guaranteed quality.

3.4.1 Reformulating subproblems

During subproblem optimization, free nodes are reconnected either to other free nodes or to existing segments. Similarly, connections within each segment are preserved from the solution Π, while the links between the segment's endpoints and the rest of the segments or free nodes are refined. By aggregating each segment $(\pi_1^k, \ldots, \pi_{c_k}^k)$ into a super node s_k, we construct a new list of nodes of size $|K| + |W|$, denoted as $\{s_1, ..., s_{|K|}, w_1, ..., w_{|W|}\}$. This updated node list leads to a partially asymmetric TSP (ATSP), characterized by an asymmetric block distance matrix:

$$D_{ATSP} = \left[\begin{array}{c|c} D_{|K|\times|K|} & D_{|K|\times|W|} \\ \hline D_{|W|\times|K|} & D_{|W|\times|W|} \end{array}\right]_{(|W|+|K|)\times(|W|+|K|)}$$

where $D_{|W|\times|W|}$ contains symmetric distances d_{w_i,w_j} between free nodes. The submatrices $D_{|W|\times|K|}$, $D_{|K|\times|W|}$, and $D_{|K|\times|K|}$ are the root of asymmetry. $D_{|W|\times|K|}$ contains distances d_{w_i,π_1^k} from free nodes to the starting nodes of fixed segments, whereas $D_{|K|\times|W|}$ represents distances $d_{\pi_{c_k}^k,w_i}$ from the ending nodes of fixed segments to free nodes; $D_{|K|\times|K|}$ indicates the distances $d_{\pi_{c_k}^k,\pi_1^k}$ from the ending nodes of fixed segments to the starting nodes of other segments.

We further transform this partially ATSP into a standard STSP to make it compatible with the solver. Following the approaches in Jonker & Volgenant (1983); Cirasella et al. (2001), the transformation introduces a dummy node s'_k for each node s_k in the ATSP, expanding the node set to $\{s_1, ..., s_{|K|}, s'_1, ..., s'_{|K|}, w_1, ..., w_{|M|}\}$. The resulting STSP is characterized by a symmetric block distance matrix:

$$D_{STSP} = \begin{bmatrix} \infty & D_{|W|\times|K|}^T & \hat{D}_{|K|\times|K|}^T \\ D_{|W|\times|K|} & D_{|W|\times|W|} & D_{|K|\times|W|}^T \\ \hat{D}_{|K|\times|K|} & D_{|K|\times|W|} & \infty \end{bmatrix}_{(|W|+2|K|)\times(|W|+2|K|)}$$

where the diagonal of $\hat{D}_{|K|\times|K|}$ is set to be a small enough value compared to the original $D_{|K|\times|K|}$, which encourages the super nodes k and their corresponding dummy nodes $k + |W|$ to be adjacently connected during the optimization.

3.4.2 Solving and recovering the solution for the original TSP

The solver optimizes an STSP using its D_{STSP} and produces an optimal solution $\mathit{\Pi}_{STSP}^*$. The output solution $\mathit{\Pi}_{STSP}^*$ is then recovered into the corresponding ATSP solution $\mathit{\Pi}_{ATSP}^*$ by directly removing all dummy nodes in the solution Π^*_{STSP}. Furthermore, each super node $s_k \in \Pi^*_{ATSP}$ is unfolded into its original segment $(\pi^k_1, \ldots, \pi^k_{c_k})$. This recovery process results in an updated solution for the original TSP conditioned on the identified subproblem ω: $\Pi^* = F_{\text{solver}}(D_{STSP}, T_{\text{max}} \mid \omega)$, where T_{max} is the runtime limit set for the exact solvers. The time limit T_{max} forces the solver to stop improving lower bounds and return the best incumbent solutions. This prevents the solver from getting stuck on certain subproblems for an excessively long time. We use the hill-climbing rule in accepting this new solution if it reaches a lower objective value than the current solution.

3.5 Asynchronous orchestration

The visual selection module $F_{\text{selector}}(\cdot)$ is I/O intensive, dominated by waiting for responses from the VLM server, whereas the exact solver module $F_{\text{solver}}(\cdot)$ is CPU-intensive. Due to their distinct computational profiles, sequential execution easily leaves solvers idle while waiting for VLM's sections. To address this, *ViTSP* executes the optimization and selection modules asynchronously on multi-core CPU systems, assigning them to separate cores and coordinating through three shared components: global solution Π, trajectory pool Φ, and subproblem queue Ω. These components provide the necessary contextual information required for module execution.

To further improve efficiency, *ViTSP* deploys multiple VLMs and solvers. On the selection side, we employ both fast-thinking and reasoning VLMs, leveraging pre-trained models with complementary strengths (Shen et al., 2023; Snell et al., 2024; Kumar et al., 2025). Each single VLM is elicited to generate Q coordinate sets $\{C_1, C_2, ..., C_Q\}$ per prompt, where $Q \geq 2$.

On the optimization side, multiple identical solvers retrieve and optimize subproblems from the shared queue in parallel, ensuring that newly generated subproblems are not left unprocessed. To mitigate conflicts in updating global solutions, *ViTSP* assigns P slave solvers to optimize and screen the retrieved subproblems, while a single master is permitted to update Π. Slave solvers discard subproblems without improvements, while those yielding net gains are forwarded to the master solver for refining Π. The process continues iteratively until no improvement is observed in K consecutive steps. Detailed pseudocode is provided in Algorithm 2 in Appendix E.

4 Experiments and results analysis

4.1 Experimental setups

Evaluation datasets. To comprehensively assess the performance of *ViTSP*, this work used TSP instances from TSPLIB (Reinelt, 1991) as primary evaluation datasets. TSPLIB offers a wide range of real-world instances for TSP, covering diverse distributions and scales. Moreover, Reinelt (2007) provides proven optimality for instances, enabling the measurement of optimality gaps even at very large scales. We chose TSP instances with $N \geq 1,000$ from the dataset to represent (very-)large-scale problems, where exact solvers begin to struggle. This results in 33 TSPLIB instances. These instances follow a naming format of `[keywords][number of nodes]`, such as `pla85900`. Instances with the same keywords are from the same application domain. Notably, since our framework does not require additional training or fine-tuning during implementation, we do not curate any training dataset.

None of the TSPLIB instances has been exposed to the baseline learning-based algorithms during their training phase. This ensures a fair evaluation of generalizability and scalability across all baselines. To the best of our knowledge, our work provides one of the most comprehensive evaluations on this real-world benchmark dataset, offering a thorough assessment of the proposed *ViTSP*.

Evaluation metrics. We used two metrics to measure the performance of algorithms: (1) **Optimality gaps** (%); and (2) **Runtime** (in seconds). We used the reported proven optimal objective values L^* (total distance traveled) for TSPLIB instances in Reinelt (2007) as reference and the gap is calculated as: $\frac{L_{\text{Model}} - L^*}{L^*} \times 100\%$, where L_{Model} is the objective value produced by a baseline model.

***ViTSP* setups.** In the initialization module, we used LKH-3 with its default parameter settings to warm start the *ViTSP*. In the visual selection module, we employed GPT-4.1 (fast thinking VLM) and o4-mini (reasoning VLM) as the selectors in this study. We set the number of subproblems generated per prompt $Q = 2$. In the subproblem optimization module, Concorde, the SOTA exact solver, was utilized as the subproblem solver. *ViTSP* terminates when no improvement is observed in $K = 5$ consecutive steps, and the duration from initialization to termination is recorded as runtime. For more detailed parameter configurations, please refer to Section F.5 in Appendix F.

Baselines. We compared our *ViTSP* against both classical OR approaches and learning-based approaches. Specifically, we applied the following ten baselines: (1) **Concorde**; (2) **LKH-3 (Default)**; (3) **LKH-3 (more `RUNS`)** (4) **FI**; (5) **AM**; (6) **DIFUSCO**; (7) **INViT**; (8) **DeepACO**; (9) **SO**; (10) **UDC**; (11) **EoH**. Their description and implementation details are provided in the Appendix F.

The selected learning-based approaches include both end-to-end solution construction methods and local improvement techniques that match the decomposition heuristics used in this study. They either provided open-source code and pre-trained checkpoints or reported results on TSPLIB instances (e.g., SO). The selected EoH produced applicable results on large-scale instances, whereas other LLM-based approaches, such as Yang et al. (2023); Elhenawy et al. (2024), failed to generate valid solutions even on small-scale cases and were therefore not included as baselines in this study.

We align the runtime across baselines to ensure fair comparison. In addition to using the default parameter values of LKH-3, we introduce LKH-3 (more `RUNS`), where the `RUNS` value is increased to match LKH-3's runtime with that of *ViTSP* on each instance. Similarly, Concorde, DeepACO, UDC, and EoH are run with the same or slightly longer time limit as *ViTSP*. For FI, runtime is deterministic with respect to instance size. End-to-end methods (AM, DIFUSCO, and INViT) also have deterministic runtimes, as they perform only a single feedforward inference.

Hardware. We used an AMD EPYC 7443 24-Core CPU and an Nvidia L40 GPU with 48GB memory to implement our work and baseline algorithms. In *ViTSP*, the VLMs were accessed on demand online. Their usage did not rely on local GPU resources but was confined by the I/O rate.

4.2 Main results

Performance comparison results are summarized in Table 1, reporting runtime (in seconds) and optimality gaps with the lowest gaps highlighted. Since *ViTSP* is warm-started from LKH-3 (Default), we further illustrate the reduction of optimality gaps over time between *ViTSP* and LKH-3 on selected instances of different scales in Figure 2. This comparison highlights *ViTSP*'s ability to improve solutions beyond what LKH-3 can achieve given additional time. We present the complete results for optimality gap reduction over time across all TSPLIB instances between *ViTSP* and LKH-3 in Appendix G. The selected box regions yielding gap reductions by VLMs are illustrated in Appendix H.

Overall, *ViTSP* achieves an average gap of 0.19% across all instances while LKH-3 (more `RUNS`) has an average gap of 0.31%. Surprisingly, *ViTSP* attains the global optimum in 11 out of 33 instances [1] with $1000 \leq N < 2,500$, despite being an approximate approach. In 6 of these 11 cases [2], it reaches optimality faster than Concorde, which in some cases has not yet reached the optimal solution within the same time limit.

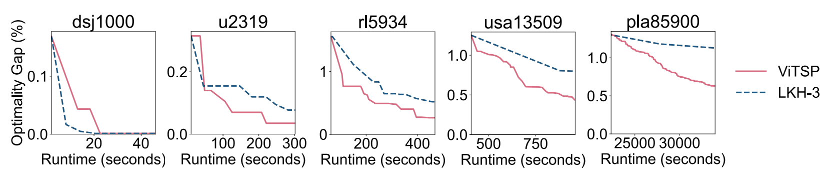

Figure 2: Optimality gaps over time on selected instances between ViTSP and LKH-3 (more `RUNS`).

ViTSP outperforms LKH-3 in 20 out of 33 instances under the same time budget in reducing gaps by 12.05% to 100.00%[3], as highlighted in yellow in Table 1. Figure 2 further illustrates how both methods reduce optimality gaps over time. As the SOTA heuristic, LKH-3 remains highly efficient for instances with $1,000 < N < 3,000$. When given a small amount of additional runtime beyond its default settings, LKH-3 reduces optimality gaps more rapidly than *ViTSP* in the early phase. However, its improvement quickly plateaus, whereas *ViTSP* continues to improve and eventually surpasses or matches (both reaching a 0% gap) LKH-3. As the problem size increases beyond $4,000$, *ViTSP* significantly speeds up the reduction of optimality gaps compared to LKH-3.

Learning-based methods struggle to generalize their learned policies to OOD instances, leading to inferior performance compared to *ViTSP*. Large gaps remain across the baselines, which impair

[1]These 11 instances are `dsj1000`, `pr1002`, `u1060`, `vm1084`, `rl1323`, `nrw1379`,`u1432`, and `fl1577`, `d1655`, `rl1889`, and `pr2392`.

[2]These 6 instances are `u1060`, `vm1084`, `rl1323`, `u1432`, `fl1577`, and `rl1889`.

[3]A 100% reduction indicates *ViTSP* has reached the global optimum whereas LKH-3 has not.

Table 1: Performance comparison on 33 TSPLIB instances.

		dsj1000	pr1002	u1060	vm1084	pcb1173	d1291	rl1304	rl1323	nrw1379	fl1400	u1432
Concorde	Time(s)	46.2	8.3	106.7	57.5	50.2	252.5	25.8	122.3	45.7	115.4	111.8
	Gap	0.00%	**0.00%**	0.00%	0.00%	**0.00%**	0.65%	**0.00%**	0.03%	**0.00%**	0.24%	0.03%
LKH-3 (Default)	Time(s)	1.9	2.0	2.0	2.1	2.2	2.5	2.7	2.7	3.4	4.5	3.8
	Gap	0.17%	0.47%	0.53%	0.12%	1.04%	0.61%	0.55%	0.21%	0.61%	0.19%	0.54%
LKH-3 (RUNS)	Time(s)	9.8	95.1	100.4	22.9	93.0	248.8	45.9	64.8	162.4	69.0	95.6
	Gap	**0.00%**	0.00%	0.01%	**0.00%**	0.01%	**0.00%**	0.00%	0.05%	0.01%	**0.18%**	0.03%
FI	Time(s)	0.4	0.4	0.5	0.4	0.5	0.7	0.6	0.7	0.8	0.8	0.8
	Gap	11.23%	10.30%	12.45%	9.51%	15.27%	21.50%	22.99%	20.78%	11.29%	4.17%	12.71%
AM (G)	Time(s)	0.8	0.7	0.8	0.8	0.9	1.0	1.0	1.1	1.1	1.1	1.1
	Gap	41.43%	40.57%	56.78%	44.05%	41.71%	48.82%	38.22%	42.85%	37.74%	63.15%	37.27%
DIFUSCO (S + 2-opt)	Time(s)	23.6	23.1	23.9	25.8	29.3	31.9	33.1	34.7	36.1	33.1	37.4
	Gap	7.83%	9.04%	7.52%	6.18%	9.26%	9.70%	9.22%	8.23%	9.70%	4.48%	8.84%
INViT	Time(s)	7.6	6.5	6.7	6.5	8.1	8.7	8.3	8.4	10.0	8.7	10.1
	Gap	8.65%	10.55%	9.74%	6.61%	6.85%	8.92%	8.97%	8.33%	6.57%	13.86%	5.09%
DeepACO	Time(s)	165.1	164.1	167.0	171.9	185.5	199.7	200.6	200.8	209.8	215.7	213.4
	Gap	21.51%	20.96%	37.97%	34.59%	20.40%	24.85%	35.00%	29.60%	18.92%	45.59%	14.75%
SO†	Time(s)	N/A	N/A	35.0	35.0	38.0	N/A	42.0	42.0	44.0	45.0	N/A
	Gap			2.21%	2.20%	2.87%		6.76%	4.21%	1.63%	1.96%	
UDC	Time(s)	47.0	91.5	65.2	53.2	140.6	242.8	42.1	123.2	148.6	70.2	89.2
	Gap	15.36%	18.75%	32.81%	29.28%	20.70%	25.30%	28.29%	18.01%	18.94%	33.57%	17.33%
EoH	Time(s)	48.11	93.2	62.7	53.8	137.7	245.3	41.9	123.5	146.2	69.8	87.4
	Gap	448.08%	56.86%	596.53%	6.47%	104.29%	18.31%	89.73%	50.14%	94.52%	26.38%	35.39%
ViTSP	Time(s)	46.1	90.7	62.4	51.5	136.3	242.7	41.8	122.2	145.7	69.2	87.3
	Gap	0.00%	0.00%	**0.00%**	0.00%	0.02%	0.05%	0.14%	**0.00%**	0.00%	0.18%	**0.00%**

		fl1577	d1655	vm1748	u1817	rl1889	d2103	u2152	u2319	pr2392	pcb3038	fl3795
Concorde	Time(s)	262.7	24.4	189.4	199.8	112.3	100.0	314.7	443.9	13.1	390.6	600.0
	Gap	1.52%	**0.00%**	0.05%	0.34%	0.09%	1.47%	0.25%	0.11%	**0.00%**	0.11%	0.44%
LKH-3 (Default)	Time(s)	4.0	4.5	5.3	5.1	6.1	6.1	7.4	10.7	11.4	15.3	25.1
	Gap	0.25%	0.80%	0.55%	1.11%	0.57%	0.54%	0.95%	0.32%	1.08%	1.20%	1.73%
LKH-3 (RUNS)	Time(s)	16.3	150.0	180.0	202.3	118.8	94.5	331.5	316.4	282.1	407.6	527.4
	Gap	**0.00%**	0.00%	0.12%	0.15%	0.06%	**0.00%**	0.09%	0.08%	0.03%	0.12%	0.67%
FI	Time(s)	1.0	1.2	1.2	1.3	1.4	1.7	1.8	2.1	2.4	4.0	5.6
	Gap	17.61%	15.41%	11.90%	18.10%	17.63%	23.57%	20.65%	6.54%	13.71%	14.92%	17.57%
AM (G)	Time(s)	1.3	1.4	1.5	1.6	1.7	1.9	1.9	2.2	2.1	3.1	4.3
	Gap	51.92%	61.15%	49.56%	56.51%	49.57%	55.48%	66.28%	29.92%	62.35%	62.33%	84.55%
DIFUSCO (S + 2-opt)	Time(s)	44.3	46.2	53.8	55.9	61.4	77.8	79.1	94.8	111.2	214.1	327.3
	Gap	7.66%	10.33%	8.93%	13.47%	8.53%	12.65%	13.94%	5.72%	10.74%	10.95%	6.98%
INViT	Time(s)	10.7	12.7	11.8	13.4	13.1	15.9	17.1	18.8	20.7	30.0	36.4
	Gap	7.65%	11.54%	8.26%	8.32%	10.03%	7.74%	7.11%	0.93%	8.12%	7.85%	15.13%
DeepACO	Time(s)	236.8	241.7	250.8	262.9	276.3	296.3	299.0	321.1	333.6	418.0	506.7
	Gap	42.74%	25.46%	34.78%	25.78%	42.15%	19.32%	24.52%	11.12%	30.43%	24.71%	103.56%
SO†	Time(s)	51.0	N/A	53.0	N/A	58.0	71.0	66.0	N/A	N/A	93.0	121.0
	Gap	4.42%		3.61%		4.63%	8.32%	4.91%			2.67%	5.49%
UDC	Time(s)	93.2	146.8	178.3	198.5	108.9	95.4	312.5	265.1	248.9	364.5	OOM
	Gap	24.94%	19.55%	33.24%	24.43%	36.28%	15.79%	27.32%	21.52%	28.25%	26.98%	
EoH	Time(s)	92.5	147.2	179.6	198.7	108.3	95.3	311.6	265.7	249.2	365.5	304.1
	Gap	181.75%	26.82%	3.47%	18.07%	515.05%	279.06%	57.30%	50.31%	11.71%	33.82%	59.46%
ViTSP	Time(s)	92.2	146.8	178.1	197.3	107.7	94.5	311.5	264.6	248.5	364.3	303.6
	Gap	0.00%	0.00%	**0.04%**	**0.13%**	**0.00%**	0.45%	**0.07%**	**0.04%**	0.00%	**0.10%**	**0.42%**

		fnl4461	rl5915	rl5934	pla7397	rl11849	usa13509	brd14051	d15112	d18512	pla33810	pla85900
Concorde	Time(s)	305.0	550.0	266.0	671.3	1000.0	960.0	1742.0	2500.0	2941.0	Failed	Failed
	Gap	0.30%	**0.67%**	0.89%	0.48%	0.85%	0.48%	0.49%	0.44%	0.57%		
LKH-3 (Default)	Time(s)	39.8	63.3	64.0	92.0	311.8	382.9	417.5	524.3	728.2	2079.7	22344.0
	Gap	0.96%	1.96%	1.56%	0.83%	1.75%	1.25%	1.18%	1.22%	1.29%	1.43%	1.31%
LKH-3 (RUNS)	Time(s)	306.5	549.3	548.1	785.5	1014.4	979.3	1908.5	2974.5	3176.2	8237.9	33966.5
	Gap	0.45%	0.73%	0.47%	0.29%	1.06%	0.81%	0.82%	0.84%	0.94%	1.00%	1.13%
FI	Time(s)	8.85	14.96	15.42	22.26	60.50	82.51	89.38	104.11	155.37	462.16	3294.52
	Gap	11.30%	22.15%	20.91%	13.24%	19.39%	12.52%	11.64%	11.67%	11.77%	16.84%	14.46%
AM (G)	Time(s)	5.18	7.97	8.23	11.06	23.51	29.63	31.62	35.46	50.77	153.60	959.18
	Gap	70.93%	79.80%	86.11%	107.17%	104.74%	142.13%	111.99%	105.51%	118.44%	137.11%	175.24%
DIFUSCO (S + 2-opt)	Time(s)	586.0	1321.0	1317.3	1946.9	5097.6	6598.8	7098.5	8226.8	12163.2	OOM	OOM
	Gap	11.03%	11.53%	11.01%	9.32%	52.49%	26.47%	53.80%	61.81%	80.49%		
INViT	Time(s)	55.3	51.4	51.5	73.6	154.0	220.0	232.4	234.3	373.9	1010.4	5910.8
	Gap	6.58%	9.43%	10.84%	7.66%	10.19%	11.94%	9.21%	8.04%	8.38%	7.34%	6.34%
DeepACO	Time(s)	594.7	766.9	763.7	1061.7	2513.4	3424.8	3818.8	4560.3	7665.8	33819.5	OOM
	Gap	37.31%	73.03%	77.79%	76.26%	94.37%	130.03%	102.88%	84.05%	100.42%	160.79%	
SO†	Time(s)	139.0	194.0	196.0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Gap	2.43%	7.99%	6.14%								
UDC	Time(s)	304.1	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
	Gap	12.80%										
EoH	Time(s)	305.1	539.2	468.2	739.2	725.6	961.0	1744.5	2501.7	Failed	Failed	Failed
	Gap	3.10%	15.15%	26.68%	23.72%	114.86%	60.94%	70.64%	11.41%			
ViTSP	Time(s)	303.9	538.0	467.1	738.8	725.4	960.1	1742.2	2499.6	2941.1	6004.6	33022.1
	Gap	**0.05%**	1.21%	**0.26%**	**0.04%**	**0.73%**	**0.43%**	**0.34%**	**0.30%**	**0.30%**	**0.31%**	**0.65%**

† Extracted from original papers.

their utilization at a practical scale. For example, despite their claimed speedup, AM and DeepACO even underperform the simple heuristic algorithm FI, demonstrating their brittleness when no model reconfiguration or time-consuming retraining is performed. Furthermore, due to their high GPU memory requirements, all learning-based algorithms except AM and INViT fail to scale to `pla85900` and encounter OOM errors. Notably, UDC suffers from OOM when the instance size exceeds 5,000, failing to produce feasible decomposition heuristics for very large-scale TSPs. For LLM-based approaches, EoH shows limited effectiveness in designing heuristics for varying TSP instances, exhibiting high variance in optimality gaps across the 33 TSPLIB instances. These observations highlight the advantage of *ViTSP*, which leverages generative models to visually guide high-quality decomposition heuristics while reducing reliance on local GPU resources and (re-)training.

Our experiments suggest that certain TSP structures can make optimization more or less difficult. Although `pr2392` is twice the size of `prc1173`, Concorde uses only 26% of the time required for `prc1173` to reach optimality for `pr2392`. Also, both *ViTSP* and LKH-3 struggle to find high-quality solutions on `fl1400`, and *ViTSP* shows difficulty in reducing gaps on `d2103` and `rl5915`.

4.3 Ablation Studies

To verify that VLMs can conduct principled selection as visual selectors, rather than fruitless permutation, we devise two heuristic selection policies adopted from Li et al. (2021); Cheng et al. (2023): (1) **Random sequence selector**: uniformly and randomly selecting a segment of a given length from the tour. The segment length is set to match the average number of nodes selected per step by *ViTSP*. (2) **Random box selector**: uniformly and randomly selecting rectangular subproblems of random sizes. It chooses $Q = 2$ subproblems at each step as *ViTSP*. We replace the default VLM selector modules with these alternatives in *ViTSP* and execute the same experiments on a given instance, where all scenarios start with solutions initialized using LKH-3. Due to the page limit, we report the optimality gaps over time using different selector policies on selected instances in Figure 3. The full results are illustrated in Appendix I.

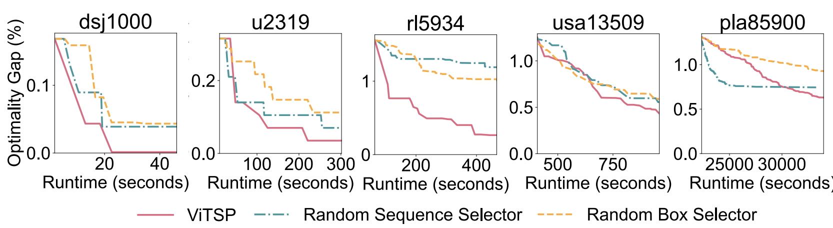

Figure 3: Ablation studies of different selection policies on selected instances.

VLMs are capable of performing meaningful, non-random subproblem selection as prompted in the framework *ViTSP*. As shown in Figure 3, *ViTSP* consistently reduces optimality gaps over time and outpaces both random sequence and random box methods. This highlights the effectiveness of visually leveraging instance-specific structures to guide subproblem selection and generalize to (very-) large-scale unseen TSP instances.

While the two random selection strategies demonstrate some ability to reduce gaps, they consistently converge to local optima. Interestingly, for `pla85900`, the random sequence selection outperforms *ViTSP* in the early stages by reducing the gap more rapidly. This suggests the potential value of alternative operations beyond the box-region selection used in this study.

5 Conclusions

In this study, we proposed a vision-guided framework to effectively solve TSPs with varying scales and distributions. *ViTSP* hybridizes the strength of pre-trained VLMs and existing OR techniques by selecting instance-specific subproblems visually and then delegating them to an off-the-shelf solver. Our proposed *ViTSP* bypasses *ad hoc* training while exhibiting effectiveness and scalability, achieving lower average optimality gaps than LKH-3 and baseline learning-based approaches. Because TSP serves as the base case for a wide family of routing problems, the promising results from ViTSP suggest opportunities to expand our framework to other routing problems. While this study validates the effectiveness of visual guidance, interpreting *how* the decomposition is determined lies beyond our current scope and remains an important future direction. Further limitations are discussed in the Appendix J. More broadly, this work highlights the potential of generative AI to support CO at practical scales, particularly in settings where abundant training data is unavailable.

Ethics Statement

I acknowledge that I and all co-authors of this work have read and commit to adhering to the ICLR Code of Ethics. To the best of our knowledge, this work does not involve potential violations of the ICLR Code of Ethics.

Reproducibility Statement

We have made efforts to ensure the reproducibility of this work. The source code is available at https://anonymous.4open.science/r/ViTSP_codes-6683 . Details of the experimental setup are provided in Section 4.1 of the main text and in Sections C and F.5 of the Appendix to further support reproducibility. Additionally, the TSPLIB dataset used in this study is publicly available, which ensures the reproducibility of the results.

The use of large language models (LLMs)

The authors confirm that LLMs were used only for grammar checking and text polishing. They were not involved in research ideation. Their role in writing was limited, such that they are not considered contributors.

References

Belarmino Adenso-Díaz and Manuel Laguna. Fine-Tuning of Algorithms Using Fractional Experimental Designs and Local Search. *Operations Research*, 54(1):99–114, February 2006. ISSN 0030-364X, 1526-5463. doi: 10.1287/opre.1050.0243. URL https://pubsonline.informs.org/doi/10.1287/opre.1050.0243.

David L. Applegate. *The traveling salesman problem: a computational study*, volume 17. Princeton university press, 2006. URL https://books.google.com/books?hl=en&lr=&id=vhsJbqomDuIC&oi=fnd&pg=PP11&dq=The+traveling+salesman+problem+:+a+computational+study&ots=YLCJVzMU78&sig=-6F9EbUUIJl46yILKTkg6CxfIHg.

Federico Berto, Chuanbo Hua, Junyoung Park, Laurin Luttmann, Yining Ma, Fanchen Bu, Jiarui Wang, Haoran Ye, Minsu Kim, Sanghyeok Choi, Nayeli Gast Zepeda, André Hottung, Jianan Zhou, Jieyi Bi, Yu Hu, Fei Liu, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Davide Angioni, Wouter Kool, Zhiguang Cao, Qingfu Zhang, Joungho Kim, Jie Zhang, Kijung Shin, Cathy Wu, Sungsoo Ahn, Guojie Song, Changhyun Kwon, Kevin Tierney, Lin Xie, and Jinkyoo Park. RL4CO: an Extensive Reinforcement Learning for Combinatorial Optimization Benchmark, June 2024. URL http://arxiv.org/abs/2306.17100. arXiv:2306.17100 [cs].

Hanni Cheng, Haosi Zheng, Ya Cong, Weihao Jiang, and Shiliang Pu. Select and optimize: Learning to solve large-scale tsp instances. In *International Conference on Artificial Intelligence and Statistics*, pp. 1219–1231. PMLR, 2023. URL https://proceedings.mlr.press/v206/cheng23a.html.

Jill Cirasella, David S. Johnson, Lyle A. McGeoch, and Weixiong Zhang. The Asymmetric Traveling Salesman Problem: Algorithms, Instance Generators, and Tests. In Adam L. Buchsbaum and Jack Snoeyink (eds.), *Algorithm Engineering and Experimentation*, pp. 32–59, Berlin, Heidelberg, 2001. Springer. ISBN 978-3-540-44808-2. doi: 10.1007/3-540-44808-X_3.

Donald Davendra (ed.). *Traveling Salesman Problem, Theory and Applications*. InTech, December 2010. ISBN 978-953-307-426-9. doi: 10.5772/547.

Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. Learning Heuristics for the TSP by Policy Gradient. In Willem-Jan Van Hoeve (ed.), *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, volume 10848, pp. 170–181. Springer International Publishing, Cham, 2018. ISBN 978-3-319-93030-5 978-3-319-93031-2. doi: 10.1007/978-3-319-93031-2_12. URL https://link.springer.com/10.1007/978-3-319-93031-2_12. Series Title: Lecture Notes in Computer Science.

Mohammed Elhenawy, Ahmad Abutahoun, Taqwa I. Alhadidi, Ahmed Jaber, Huthaifa I. Ashqar, Shadi Jaradat, Ahmed Abdelhay, Sebastien Glaser, and Andry Rakotonirainy. Visual Reasoning and Multi-Agent Approach in Multimodal Large Language Models (MLLMs): Solving TSP and mTSP Combinatorial Challenges, June 2024. URL http://arxiv.org/abs/2407.00092. arXiv:2407.00092.

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 *ICML'24*, pp. 12973–12992, Vienna, Austria, July 2024. JMLR.org.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a Small Pre-trained Model to Arbitrarily Large TSP Instances. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8): 7474–7482, May 2021. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v35i8.16916. URL https://ojs.aaai.org/index.php/AAAI/article/view/16916.

Keld Helsgaun. Short description of the parameters to LKH-2.0, March 2016. URL http://webhotel4.ruc.dk/~keld/research/LKH/LKH-2.0/DOC/LKH-2.0_PARAMETERS.pdf.

Keld Helsgaun. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 12:966–980, 2017. URL http://akira.ruc.dk/~keld/research/LKH/LKH-3_REPORT.pdf.

John H. Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992. URL https://www.jstor.org/stable/24939139?casa_token=yaAGwUMbktkAAAAA:yMqqHin9c83M7-AEwddJKRzRE0AmTV8bIaWP\WhNFhzz4WxU1NXQGl0AUUSs_ _\Af4Pqb3T3Kd3zw96SCD1fM5EJYdh-\cSCyTQ19HoCP9uKu3NE3G6PrQ. Publisher: JSTOR.

Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph Neural Network Guided Local Search for the Traveling Salesperson Problem, April 2022. URL http://arxiv.org/abs/2110.05291. arXiv:2110.05291 [cs].

Yan Jin, Yuandong Ding, Xuanhao Pan, Kun He, Li Zhao, Tao Qin, Lei Song, and Jiang Bian. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 8132–8140, 2023. URL https://ojs.aaai.org/index.php/AAAI/article/view/25982. Issue: 7.

Roy Jonker and Ton Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161–163, November 1983. ISSN 0167-6377. doi: 10.1016/0167-6377(83)90048-2. URL https://www.sciencedirect.com/science/article/pii/0167637783900482.

Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the Travelling Salesperson Problem Requires Rethinking Generalization, May 2022. URL http://arxiv.org/abs/2006.07054. arXiv:2006.07054.

Muhammad Asif Khan and Layth Hamad. On the Capability of LLMs in Combinatorial Optimization, November 2024. URL https://www.techrxiv.org/users/397659/articles/1236430-on-the-capability-of-llms-in-combinatorial-optimization?commit=93c3f663127d5993fda95e409ae9bb389c640556.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, Learn to Solve Routing Problems!, February 2019. URL http://arxiv.org/abs/1803.08475. arXiv:1803.08475 [cs, stat].

Komal Kumar, Tajamul Ashraf, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, Phillip H. S. Torr, Salman Khan, and Fahad Shahbaz Khan. LLM Post-Training: A Deep Dive into Reasoning Large Language Models, February 2025. URL http://arxiv.org/abs/2502.21321. arXiv:2502.21321 [cs].

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. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/f231f2107df69eab0a3862d50018a9b2\-Abstract.html.

Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, June 1992. ISSN 03772217. doi: 10.1016/0377-2217(92)90138-Y. URL https://linkinghub.elsevier.com/retrieve/pii/037722179290138Y.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, Maxime Gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. IN-CONTEXT REINFORCEMENT LEARNING WITH ALGORITHM DISTILLATION. 2023.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. In *Advances in Neural Information Processing Systems*, volume 34, pp. 26198–26211. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/dc9fa5f217a1e57b8a6adeb065560b38\-Abstract.html.

Xiayang Li and Shihua Zhang. Learning-Based TSP-Solvers Tend to Be Overly Greedy, February 2025. URL http://arxiv.org/abs/2502.00767. arXiv:2502.00767 [cs] version: 1.

Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:50020–50040, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/9c93b3cd3bc60c0fe7b0c2d74a2da966-Abstract-Conference.html.

Yang Li, Jinpei Guo, Runzhong Wang, Hongyuan Zha, and Junchi Yan. Fast T2T: Optimization Consistency Speeds Up Diffusion-Based Training-to-Testing Solving for Combinatorial Optimization, February 2025. URL http://arxiv.org/abs/2502.02941. arXiv:2502.02941 [cs].

Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model. In *Proceedings of the 41st International Conference on Machine Learning*, July 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

Yining Ma, Zhiguang Cao, and Yeow Meng Chee. Learning to Search Feasible and Infeasible Regions of Routing Problems with Flexible Neural k-Opt. In *Advances in Neural Information Processing Systems*, volume 36, pp. 49555–49578, December 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/9bae70d354793a95fa18751888cea07d-\Abstract-Conference.html.

Amir Moeini, Jiuqi Wang, Jacob Beck, Ethan Blaser, Shimon Whiteson, Rohan Chandra, and Shangtong Zhang. A Survey of In-Context Reinforcement Learning, February 2025. URL http://arxiv.org/abs/2502.07978. arXiv:2502.07978 [cs].

Giovanni Monea, Antoine Bosselut, Kianté Brantley, and Yoav Artzi. LLMs Are In-Context Reinforcement Learners, October 2024. URL http://arxiv.org/abs/2410.05362. arXiv:2410.05362.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8):9345–9353, June 2023. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v37i8.26120. URL https://ojs.aaai.org/index.php/AAAI/article/view/26120.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems, October 2022. URL http://arxiv.org/abs/2210.04123. arXiv:2210.04123.

Gerhard Reinelt. TSPLIB—A Traveling Salesman Problem Library. *ORSA Journal on Computing*, 3(4):376–384, November 1991. ISSN 0899-1499, 2326-3245. doi: 10.1287/ijoc.3.4.376. URL https://pubsonline.informs.org/doi/10.1287/ijoc.3.4.376.

Gerhard Reinelt. Optimal solutions for symmetric tsps, May 2007. URL http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html.

Daniel J. Rosenkrantz, Richard Edwin Stearns, and Philip M. Lewis. Approximate algorithms for the traveling salesperson problem. In *15th Annual Symposium on Switching and Automata Theory (swat 1974)*, pp. 33–42. IEEE, 1974. URL https://ieeexplore.ieee.org/abstract/document/4569756/?casa_token=qTFLUAv6htkAAAAA:R8rgcatI4H46B-\eegx9JQ5jwikkOIrXkBEOW4hXLU5\LAUOFECEVhK4Vx24AdIfKaLhAAM38O.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in HuggingFace, April 2023. URL http://arxiv.org/abs/2303.17580. arXiv:2303.17580 [cs].

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters, August 2024. URL http://arxiv.org/abs/2408.03314. arXiv:2408.03314 [cs].

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization, December 2023. URL http://arxiv.org/abs/2302.08224. arXiv:2302.08224.

A. Vaswani. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. URL https://user.phil.hhu.de/~cwurm/wp-content/uploads/2020/01/7181-attention-is-all-you-need.pdf.

Laurence A. Wolsey. *Integer programming*. John Wiley & Sons, 2020. URL https://books.google.com/books?hl=en&lr=&id=knH8DwAAQBAJ&oi=fnd&pg=PP1&dq=integer+programming&ots=wlzvktKHk9&sig=Xt-9Zc39eRkb6ifGuSpmTcSKDwU.

Xuan Wu, Di Wang, Lijie Wen, Yubin Xiao, Chunguo Wu, Yuesong Wu, Chaoyu Yu, Douglas L. Maskell, and You Zhou. Neural Combinatorial Optimization Algorithms for Solving Vehicle Routing Problems: A Comprehensive Survey with Perspectives, October 2024. URL http://arxiv.org/abs/2406.00415. arXiv:2406.00415 [cs].

Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. Learning Improvement Heuristics for Solving Routing Problems. *IEEE Transactions on Neural Networks and Learning Systems*, 33(9):5057–5069, September 2022. ISSN 2162-2388. doi: 10.1109/TNNLS.2021.3068828. URL https://ieeexplore.ieee.org/document/9393606/?arnumber=9393606. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

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, pp. 7472–7483. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/3d863b367aa379f71c7afc0c9cdca41d-Abstract.html.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large Language Models as Optimizers, September 2023. URL http://arxiv.org/abs/2309.03409. arXiv:2309.03409 [cs].

Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization, November 2023. URL http://arxiv.org/abs/2309.14032. arXiv:2309.14032.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution.

In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 43571–43608. Curran Associates, Inc., 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/4ced59d480e07d290b6f29fc8798f195-Paper-Conference.pdf.

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, July 2024b. URL http://arxiv.org/abs/2312.08224. arXiv:2312.08224 [cs].

Zhuoli Yin, Zhaoyu Kou, and Hua Cai. A Deep Reinforcement Learning Model for Large-Scale Dynamic Bike Share Rebalancing with Spatial-Temporal Context. In *The 12th International Workshop on Urban Computing*, 2023. URL http://urban-computing.com/urbcomp2023/file/UrbComp2023_paper_7.pdf.

Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, and Chu-Min Li. Reinforced Lin-Kernighan-Helsgaun Algorithms for the Traveling Salesman Problems, July 2022. URL http://arxiv.org/abs/2207.03876. arXiv:2207.03876 [cs].

Zhi Zheng, Changliang Zhou, Tong Xialiang, Mingxuan Yuan, and Zhenkun Wang. UDC: A Unified Neural Divide-and-Conquer Framework for Large-Scale Combinatorial Optimization Problems, January 2025. URL http://arxiv.org/abs/2407.00312. arXiv:2407.00312 [cs].

Zefang Zong, Hansen Wang, Jingwei Wang, Meng Zheng, and Yong Li. RBG: Hierarchically Solving Large-Scale Routing Problems in Logistic Systems via Reinforcement Learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4648–4658, Washington DC USA, August 2022. ACM. ISBN 978-1-4503-9385-0. doi: 10.1145/3534678.3539037. URL https://dl.acm.org/doi/10.1145/3534678.3539037.

A TRAVELING SALESMAN PROBLEM (TSP)

A TSP is characterized by a list of nodes $i \in \{1, 2, ..., N\}$, and the corresponding coordinate sets $\{(x_i, y_i) \mid i = 1, 2, \ldots, N\}$ or the 2D Euclidean distance matrix D_N. The 2D Euclidean distance between a node pair is calculated as $d(i, j) = [\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}]$ and we round the distance to the nearest integer in this study.

The goal of TSP is to find an optimal tour that departs from an initial node, visits each node exactly once, and returns to the starting node. With loss of generality, the solution Π in this study is represented as a cyclic sequence of nodes $\Pi = [\pi_1, \pi_2, ..., \pi_N] + [\pi_1]$, where π_n represents the *n-th* node in this sequence, and π_1 denotes both the starting and ending node of the tour to form a complete cycle. The objective is to minimize the total distance traveled $L(\Pi) = \sum_{i=1}^{N-1} d(\pi_i, \pi_{i+1}) + d(\pi_N, \pi_1)$.

When the distance between two nodes is identical in both directions, i.e., $d(i, j) = d(j, i)$, the problem is known as symmetric TSP (STSP). In contrast, asymmetric TSP (ATSP) allows for different distances between certain node pairs in opposite directions, i.e., $d(i, j) \neq d(j, i)$.

B ADDITIONAL RELATED WORKS

B.1 OR APPROACHES

Many commercial solvers, such as Gurobi, OR-Tools, and CPLEX, are designed for generic optimization purposes. They search for exact optimal solutions using techniques like branch and cut, but these solvers struggle with large-scale optimization problems. Besides these commercial solvers, Concorde is believed to be the SOTA exact solver designed for TSP to obtain optimal solutions. Essentially, it also employs the LKH algorithm–the SOTA heuristic solver–and branch-and-cut techniques to find exact solutions. Concorde has been shown to solve TSPs with more than 80k nodes, but still at the expense of years of computation.

B.2 LEARNING-BASED APPROACHES

End-to-end construction. The autoregressive approach is characterized by an attention-based network architecture, such as the Transformer (Vaswani, 2017), or its variants, such as Pointerformer (Jin et al., 2023). AM in Kool et al. (2019) uses the REINFORCE algorithm to sequentially predict the node with the highest probability, while POMO in Kwon et al. (2020) produces multiple solutions in decoding steps to improve the model performance. In contrast, the non-autoregressive approaches estimate the likelihood of connecting each edge between nodes to produce a heatmap. For example, DIMES in Qiu et al. (2022) proposed learning a continuous space to parameterize the solution distribution using an anisotropic graph neural network. DIFUSCO (Sun & Yang, 2023), T2T (Li et al., 2023), and Fast T2T (Li et al., 2025) developed a graph-based diffusion model to generate the solution, which denoises random noise and the problem instance to gradually produce a feasible solution. However, DIFUSCO employs the 2-opt method to further refine the output, which brings a significant performance gain. Its standalone performance to generalize to new instances is questionable. Fang et al. (2024) introduced Invariant Nested View Transformer (INViT) to identify partial nodes with similar distributions as the trained ones to hierarchically handle partial problems. However, since these solvers always generate approximate solutions with an inevitable optimality gap, the overall solution quality can deteriorate significantly in out-of-distribution instances.

Local improvement. Li et al. (2021) trains a backbone to select a promising subproblem and then delegates it to an off-the-shelf solver for further improvement. Zong et al. (2022) developed Rewriting-by-Generating (RBG) to iteratively refine the solution partitioning and infer new local solutions by a trained generator. Similarly, Select-and-Optimize (SO) in Cheng et al. (2023) trained a policy to select promising sequences within the complete tour and used a trained solver to improve the selected sequence iteratively. Intuitively, the subproblem solution quality fully depends on the solver. RBG and SO trained small-scale solvers following the methods in end-to-end construction. Fu et al. (2021) developed a graph convolutional residual neural network with attention mechanisms (AttGCRN) to optimize split subgraphs, and it fuses optimized partial solutions as the complete solution. Similarly, H-TSP (Pan et al., 2023), GLOP (Ye et al., 2024b), and UDC (Zheng et al., 2025) decompose a TSP into open TSPs (instead of standard symmetric TSP) and optimize them using dedicated trained solvers.

C Prompts to VLMs for subproblem visual selection

The meta-text prompt instructing VLMs to select promising TSP subproblems is devised as follows. ***Italicized text in bold*** denotes placeholders for problem-specific inputs, and Q represents the number of sub-regions to be selected per query. Δ indicates the margin used to allow selection near the edge of the instance. It is set to 10% of the spatial boundary of the given TSP instance. We do not set parameters or rules for exploring, to investigate VLMs' capability to learn from the context to determine the best selection by itself. Due to the inherent differing execution pace between modules in asynchronous orchestration, VLMs can generate selections on the same global solutions before solvers finish current subproblem jobs. To mitigate duplicate selections from VLMs during asynchronous processes, the real-time subproblem queue Ω is included as part of the input prompts to VLMs.

> You are tasked with improving an existing solution to a Traveling Salesman Problem (TSP) by selecting a sub-region where the routes can be significantly optimized. Carefully consider the locations of the nodes (in red) and connected routes (in black) in the initial solution on a map. The boundary of the map is x_min= ***{x_min - Δ}***, x_max= ***{x_max + Δ}***, y_min= ***{y_min - Δ}***, y_max= ***{y_max + Δ}***.
>
> Please return ***{Q}*** sub-rectangle(s) that you believe would most reduce total travel distance from further optimization by a downstream TSP solver. Analyze the problem to do meaningful selections. Remember, if you don't see significant improvement, try selecting larger areas that cover more nodes based on your analysis of the prior selection trajectory.
> Keep your output very brief as in the following template. Don't tell me you cannot view or analyze the map. I don't want an excuse:
> $\langle$coordinates$\rangle$ x_min=1,000 , x_max=2,000 , y_min=1,000 , y_max=2,000 $\langle/$coordinates$\rangle$
>
> Avoid selecting the same regions as follows, which are pending optimization: ***{pending regions}***
> Below are some previous selection trajectories. Please avoid selecting the same subrectangle: ***{prior selection trajectory}***

where each entry in ***{pending regions}*** is retrieved from up-to-date subproblem queue Ω. Each entry in the ***prior selection trajectory*** is based on the trajectory pool Φ and formatted as follows:

> ***{coordinates}***, number of nodes within the subrectangle= ***{number of nodes}***, travel distance reduction= ***{delta objective improvement}***, computational time for this subproblem= ***{solver runtime in second}***

The visual prompt is an instance-specific image, visualizing the position of nodes and the current tour based on Π^*. The image is gridded based on $\left\lceil \sqrt{\frac{y_{\max}-y_{\min}}{100}} \right\rceil$ to adaptively provide coordinate reference for VLM.

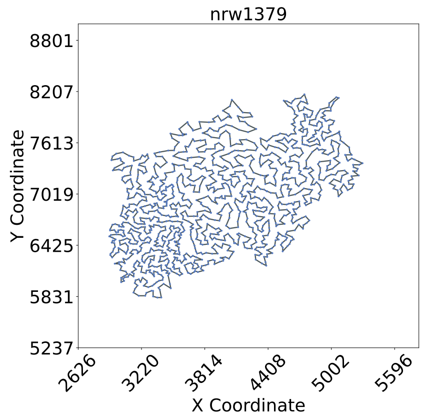

Figure 4: An example of the visual prompt to VLMs. In this example, `nrw1379` is used. The tour is initialized by LKH-3.

D Pseudo-codes for Visual Selection Module

Algorithm 1 Visual Selection Module

Input: Current global solution Π^*, Selection trajectory Φ, Meta-instruction I, Pending subproblem queue Ω, Number of subproblems per prompt Q, Maximum number of covered nodes α, VLM visual selection F_{selector}.
$(C_1, \ldots, C_Q) \leftarrow F_{\text{selector}}(\Pi^*, \Phi, I, \Omega, Q)$
for $q = 1$ **to** Q **do**
  Compute the number of covered nodes M
  while $M > \alpha$ **do**
    $C_q \leftarrow F_{\text{selector}}(\Pi^*(C_q), \Phi, I, \Omega, 1)$ ▷ Zoom-in based on current solution within C_q
    Update M
  end while
  $\omega_q = (W_q, K_q) \leftarrow \textsc{FormSubproblems}(\Pi^*, C_q)$
end for
if $\left|\bigcup_{q=1}^{Q} W_q\right| \leq \alpha$ **or** $\exists\, i \neq j$ such that $W_i \cap W_j \neq \emptyset$ **then** ▷ Too small or overlapping subproblems
  $\omega \leftarrow \omega_1 \cup \cdots \cup \omega_Q$ ▷ Merge subproblems
  Enqueue ω into Ω
else
  for $q = 1$ **to** Q **do**
    Enqueue ω_q into Ω
  end for
end if
Return Ω ▷ Updated subproblem queue

E PSEUDO-CODES FOR ASYNCHRONOUS ORCHESTRATION

Algorithm 2 Asynchronous Orchestration of *ViTSP*

Input: Distance matrix D_N, meta-instruction I, subproblems per prompt Q, node cap α, initializer $F_{\text{initializer}}$, VLM family $(\text{VLM}_1, \text{VLM}_2, \ldots)$, solver F_{solver}, max no-improvement steps K, number of parallel slave solvers P.
$\Pi^* \leftarrow F_{\text{initializer}}(D_N)$ ▷ Sec. 3.2
$\Phi \leftarrow \emptyset, \Omega \leftarrow \emptyset, \Omega' \leftarrow \emptyset$ ▷ Selection trajectory, subproblem queue, screened subproblem queue
[Parallel: Visual Selection Loop] ▷ Sec. 3.3
$F_{\text{selector}}(\cdot) \leftarrow \text{ROULETTE}(\text{VLM}_1, \text{VLM}_2, \ldots)$
$\Omega \leftarrow \text{VISUALSELECTION}(\Pi^*, \Phi, I, \Omega, Q, \alpha, F_{\text{selector}})$ ▷ Alg. 1
[Parallel: P Slave Solvers Loop] ▷ Sec. 3.5
Dequeue ω from Ω
$D_{STSP} \leftarrow \text{REFORMULATESUBPROBLEMS}(\omega)$
$\Pi \leftarrow F_{\text{solver}}(D_{STSP})$
if $L(\Pi^*) > L(\Pi)$ **then** Enqueue ω into Ω' ▷ Retain promising subproblem
[Parallel: One Master Solver Loop] ▷ Sec. 3.5
while Counter $\leq K$ **do**
    if $\Omega' \neq \emptyset$ **then** Dequeue ω from Ω'
    else Dequeue ω from Ω
    $D_{STSP} \leftarrow \text{REFORMULATESUBPROBLEMS}(\omega)$
    $\Pi \leftarrow F_{\text{solver}}(D_{STSP})$
    if $L(\Pi^*) > L(\Pi)$ **then** $\Pi^* \leftarrow \Pi$ ▷ Only master updates global solution
    else Counter $\leftarrow$ Counter $+ 1$
end while
Terminate all parallel processes ▷ Stop *ViTSP*
Return Π^* ▷ Final solution

F IMPLEMENTATION DETAILS OF VITSP AND BASELINE ALGORITHMS

F.1 CLASSICAL OR APPROACHES

Concorde We used *PyConcorde*, a Python wrapper for the Concorde solver, to solve the TSP instances. The runtime limit for Concorde is set to match the time *ViTSP* takes to converge on each specific instance.

LKH-3 The implementation version used in this study is *LKH-3.0.13*. For **LKH-3 (Default)**, we used the default parameters as specified in Helsgaun (2016), including `RUNS`=10, `MAX_TRIALS`=number of nodes, and `MOVE_TYPE`=5 (i.e., 5-opt). For **LKH-3 (more `RUNS`)**, we incrementally increase the value of `RUNS` by 50 until the actual runtime matches that of *ViTSP*. In the event that the gap is 0%, we report the runtime to reach this optimality.

Farthest insertion (FI) No parameters required for this algorithm.

F.2 END-TO-END CONSTRUCTION APPROACHES

Attention Model (AM) The algorithm was implemented based on RL4CO package (Berto et al., 2024). We utilized the open-source *tsp100* checkpoint as the backbone for the AM (Kool et al., 2019). Attempts to train a new model for larger instances, like *tsp1000*, failed due to an out-of-memory issue. We adopted the *greedy* decoding strategy since our experiments show it demonstrated superior performance compared to the *sampling* strategy. We denoted this algorithm as *AM(G)*.

DIFUSCO(Sun & Yang, 2023) We used the published pre-trained checkpoint *tsp10000-categorical* as the backbone for DIFUSCO in this study. All other parameters followed the defaults provided in the open-source code. Specifically, a sampling-based strategy was implemented with 10 diffusion steps and 16 samples. Additionally, 2-opt operations with 5,000 steps were applied. The diffusion type was categorical. We refer to this algorithm as *DIFUSCO(S+2-opt)*.

Invariant Nested View Transformer (INViT) (Fang et al., 2024) We used the open-sourced checkpoint for TSP as the backbone in assessing INViT's performance in this study and followed the default settings.

F.3 LOCAL IMPROVEMENT APPROACHES

DeepACO (Ye et al., 2023) We used an open-sourced checkpoint *tsp500*. Following the specifications in the paper, we set the number of ants to be 500.

Select-and-Optimize (SO) (Cheng et al., 2023) No public codes and checkpoints are available. We extracted the results reported in the original paper.

Unified Neural Divide-and-Conquer Framework (UDC) (Zheng et al., 2025) We used their pre-trained checkpoints for partitioning (dividing) and sub-TSP solver (conquering) to evaluate UDC's performance in TSP. We followed the default parameters reported in the paper. We set the runtime to align with *ViTSP*'s runtime on each instance.

F.4 LLM-BASED APPROACH

EoH (Liu et al., 2024) We used the open-source `LLM4AD` platform to run EoH with its default parameters. For LLM, we used GPT-4.1, the same as in *ViTSP*. Additionally, iterations are terminated once the runtime limit, matched to that of *ViTSP*, is reached.

F.5 VITSP

We used LKH-3 (Default) as the solution initializer. For VLMs, we set the number of subproblems generated per prompt $Q = 2$. To reconcile the solving time spent on a single subproblem, we set the upper bound of the number of selected nodes to $\alpha = 1000$ for instances $N < 10,000$ and $\alpha = 2000$ for instances $N > 10,000$ and we imposed the time limit on Concorde for solving each subproblem to be $T_{\max} = 10$ seconds. The number of slave solvers is set to be $P = 8$ in the asynchronous orchestration. For the VLM, a maximum of 100 tokens is set to ensure brief and speedy output. *ViTSP* is set to terminate if there are five consecutive solving steps without any improvement in the global objective value.

G Complete plots of optimality gap reduction over time between ViTSP and LKH-3 (more RUNS)

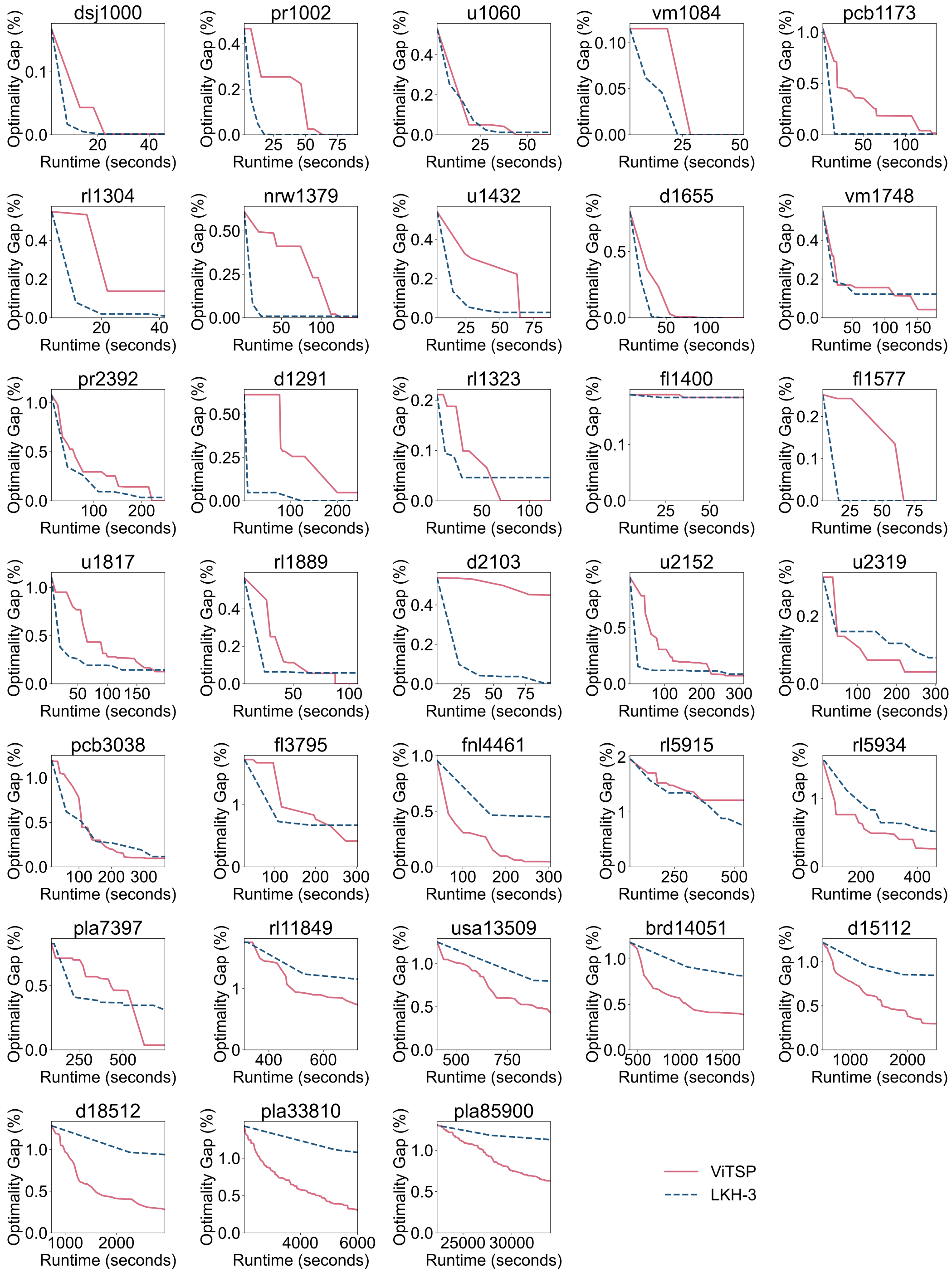

Figure 5: Optimality gap reduction over time between *ViTSP* and LKH-3 (more `RUNS`).

H IDENTIFIED SUBPROBLEMS BY VLMS THAT CONTRIBUTE TO OPTIMALITY GAP REDUCTION

In Figure 6, we plot the identified subproblems by VLM. For clarity, the selected box regions without contributing to optimality gap reductions are omitted.

Besides correcting crisscrossed edges, which is the most visually apparent suboptimality, *ViTSP* can also achieve improvements through:

(1) Refining dense regions where small gains can accumulate by transitioning from a locally approximate solution (initialized by LKH-3) to a locally optimal one, as each subproblem is re-optimized using an exact solver (Concorde).

(2) Combinatorially selecting neighborhoods whose joint optimization can escape the local optimum and lead to global improvements, recall that we use $Q = 2$ to allow two subproblem selections simultaneously at each step.

As shown in Figure 6, the subregions identified by the VLM often correspond to these two patterns. Dense areas may contain sub-paths that are locally optimal but globally improvable when viewed in context. Additionally, as we detailed in Section 3.3, we designed zoom-in reselection so VLMs always have chances to inspect detailed edge connections in a dense area.

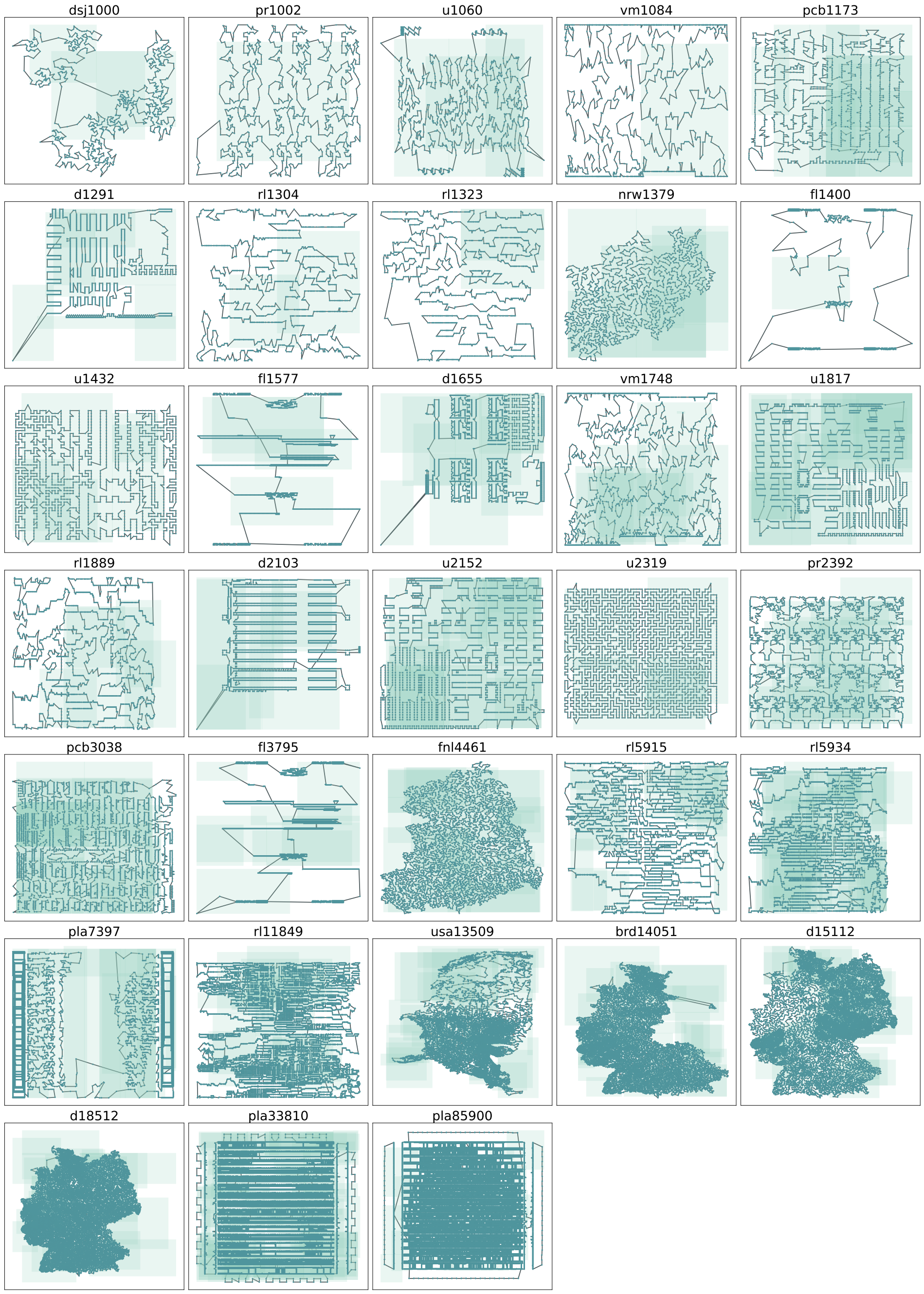

Figure 6: Visualization of selected box regions by VLMs on TSP instances.

I Complete plots for ablation studies of different selection policies

In Figure 7, we illustrate the optimality gap reduction curves among *ViTSP* and two random selectors across 33 TSPLIB instances to demonstrate the effectiveness of VLM in selecting meaningful subproblems.

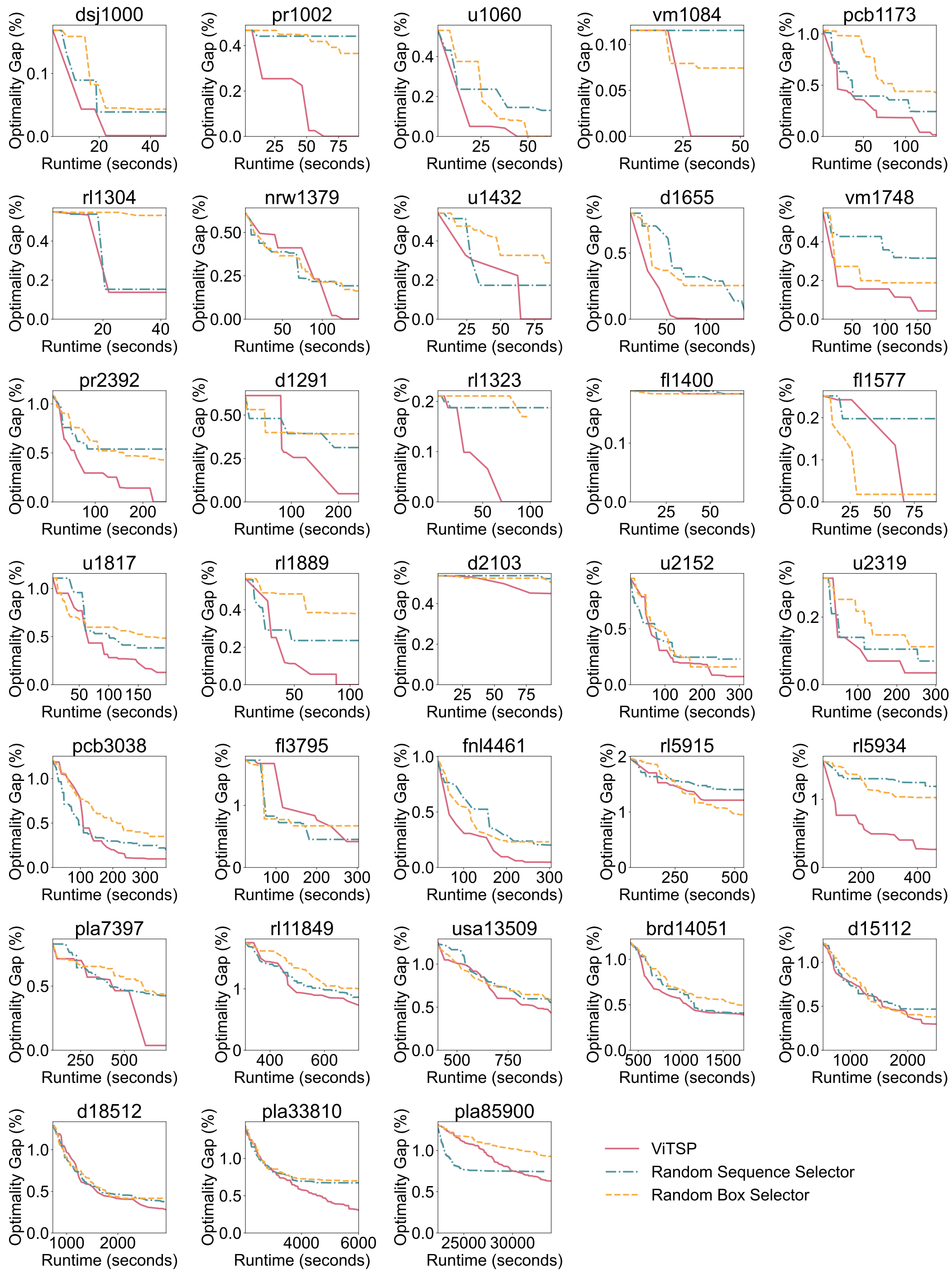

Figure 7: Optimality gap reduction over time among three selection policies.

J Limitations and Broader Impacts

We discuss two limitations and the direction of future work in this section. First, the box-based subproblem selection guided by VLMs in this study represents just one type of metaheuristic operation for combinatorial optimization. As ablation studies show, selecting a sequence of nodes may also be a helpful metaheuristic operation. Exploring additional operations designed by VLMs could further unlock the potential of hybridizing machine learning and operations research. Second, although parallel computing is employed, this study does not explicitly optimize the coordination between the selector and solver modules. In particular, there is an unexplored trade-off between solving a single large subproblem with longer runtime versus solving multiple smaller subproblems within the same time. We leave this investigation for future work.

Broadly, the TSP is more than an academic challenge—it is a foundational problem with broad applications across industries, including transportation, logistics, chip design, and DNA sequencing. This work creates new opportunities for the machine learning community to develop high-quality solutions for large-scale TSP and related problems using general-purpose large models. As VLMs become increasingly accessible and capable of advanced reasoning, they offer scalable solutions deployable on both cloud and edge devices, paving the way for practical and impactful applications.

K The use of large language models (LLMs)

The authors confirm that LLMs were used only for grammar checking and text polishing. They were not involved in research ideation. Their role in writing was limited, such that they are not considered contributors.