UNLOCKING TRADITIONAL TSP HEURISTIC GUIDED-SAMPLING VIA NEURAL INSTANCE MODIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The Traveling Salesman Problem (TSP) is one of the most representative NP-hard problems. Traditional heuristic tour constructors are computationally efficient and highly practical, but their deterministic behavior limits exploration and often leads to local optima. In contrast, neural-based heuristic tour constructors alleviate this issue through guided-sampling and typically achieve superior solution quality, but at the cost of expensive training. To bridge this gap, we propose TSP-MDF, a novel instance modification framework that equips traditional deterministic heuristic tour constructors with guided-sampling capability. Specifically, TSP-MDF introduces a neural-based instance modifier that strategically shifts node coordinates to sample multiple modified instances, on which the base traditional heuristic tour constructor constructs tours that are mapped back to the original instance, allowing traditional tour constructors to explore higher-quality tours and escape local optima. At the same time, benefiting from our instance modification formulation, the neural-based instance modifier can be trained efficiently without any ground-truth supervision, ensuring the framework maintains practicality. Extensive experiments on large-scale TSP benchmarks and real-world benchmarks demonstrate that TSP-MDF can significantly improve the performance of traditional heuristics tour constructors and with an extremely short training time.

1 INTRODUCTION

The Traveling Salesman Problem (TSP) aims to find a shortest cycle that visits each node exactly once, given a set of nodes in Euclidean space. Existing approximation methods for TSP usually adopt a two-stage strategy: (1) a tour construction stage that employs a tour constructor to construct one or multiple candidate tours (Qiu et al., 2022; Sun & Yang, 2023; Li et al., 2024; Kwon et al., 2020; Joshi et al., 2020; Jin et al., 2023; Li et al., 2023; Hottung et al., 2021; Fu et al., 2021; Kim et al., 2021; Kool et al., 2018; Ye et al., 2023; Li et al., 2025; Goh et al., 2024; Drakulic et al., 2023; Luo et al., 2023; 2025b); (2) an optional tour refinement stage that iteratively improves the constructed candidate tours refined (Croes, 1958; Ye et al., 2024; Kim et al., 2021; Fu et al., 2021; Helsgaun, 2017; Ye et al., 2023; Luo et al., 2025a).

Regarding the tour construction stage, traditional heuristic tour constructors often construct a tour by iteratively inserting nodes into the partial tour, where the insertion order is determined by hand-crafted rules (e.g., Farthest/Nearest Insertion). These heuristics are efficient (e.g., $O(n^2)$), but are not flexible and often struggle in certain complex regions of the underlying instance and approximate a sub-optimal tour. Although the subsequent tour refinement stage can refine the constructed tour via local search (Croes, 1958; Ye et al., 2024; Kim et al., 2021; Helsgaun, 2017; Ye et al., 2023; Luo et al., 2025a), unfortunately, its effectiveness heavily depends on the quality of the initial tour. When the initial tour is of poor quality, local search may converge to another undesirable sub-optimal solution within the same local optimum, and may also require a large number of refinement steps, resulting in high computational cost compared to initial tour construction. **Thus, constructing high quality initial tours remains crucial for effective TSP approximation.**

A common approach to escape local optima is to sample multiple candidate initial tours during tour construction stage (Ye et al., 2024; Kim et al., 2021; Helsgaun, 2017), enabling broader exploration of the solution space. **However, traditional heuristic constructors (e.g., Farthest/Nearest Insertion) are inherently deterministic and typically produce only a single tour for a given instance.** Thus,

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.

multiple candidates are often obtained via random insertion orders in practice (Ye et al., 2024; Kim et al., 2021; Helsgaun, 2017), which leads to unstable tour quality and requires extensive sampling, resulting in inefficient exploration. To address these limitations, neural-based heuristic tour constructors have been widely studied (Qiu et al., 2022; Sun & Yang, 2023; Li et al., 2024; 2023; Hottung et al., 2021; Fu et al., 2021; Kwon et al., 2020; Joshi et al., 2020; Kim et al., 2021; Jin et al., 2023; Kool et al., 2018; Ye et al., 2023; Li et al., 2025; Goh et al., 2024; Drakulic et al., 2023; Luo et al., 2023; 2025b). They have achieved promising results by performing guided-sampling using the heuristic learned by the neural networks, and are often more flexible than traditional counterparts. Specifically, given a TSP instance, the neural network estimates distributions over promising regions of the solution space, enabling the sampling of multiple candidate tours with controlled quality. Such guided-sampling strategy improves exploration, increases the likelihood of escaping local optima and discovering higher-quality tours.

Unfortunately, existing neural-based constructors suffer from several practical issues: (1) they often **require ground-truth supervision** to achieve good solution quality, which is costly to obtain in practice, especially for large instances; (2) As neural-based heuristic tour constructors have to be trained entirely from scratch, they often learn ineffective and unstable heuristics that yield poor solutions in the early training stage. And since the high-quality solution space is extremely small and highly constrained, occupying only a tiny fraction of the overall solution space, learning effective heuristics becomes particularly challenging and often **requires an extremely long training time** to achieve satisfactory performance. These limitations collectively **increase the training cost and restrict the practicability of neural-based heuristic tour constructors**, particularly under limited computational budgets. In contrast, traditional heuristic constructors can directly produce reasonable initial tours without training or labeled data. This motivates a key question for tour construction: ***"Instead of learning a neural-based constructor from scratch, can we leverage a traditional heuristic tour constructor as a practical base constructor, and enhance it by equipping it with guided-sampling capability, while preserving practicability?"***

Our Contributions. To answer the above question, we propose TSP-MDF, a new formulation for TSP tour construction. Unlike prior work that re-designs heuristics, TSP-MDF introduces a simple and effective instance modification framework that enables traditional deterministic heuristic constructors to perform guided-sampling by approximating tours on multiple modified instances. In particular, **(1)** we introduce an additional pre-processing stage before the tour construction stage, where an unsupervised neural-based instance modifier is trained to strategically shift the node coordinates of the original TSP instance and sample multiple modified instances. Then, by simply applying the traditional heuristic tour constructor to these modified instances, multiple candidate tours can be obtained and mapped back to the original instance. In this way, **traditional deterministic heuristic tour constructors are now equipped with guided-sampling capability**; **(2)** Moreover, benefiting from our formulation, TSP-MDF also introduces an optional instance refinement stage, where modified instances can be further iteratively adjusted to guide the heuristic constructors toward better tours. To this end, TSP-MDF **provides great flexibility for traditional deterministic heuristic tour constructors to escape local optima and explore higher-quality tours by leveraging both parallel and sequential guided-sampling**; **(3)** Crucially, benefiting from our formulation, the space of high-quality instance modifications is far less constrained than that of high-quality tours, as multiple effective modifications may lead to the same improved tour. Meanwhile, a large portion of the unpromising solution space can be quickly pruned by restricting modifications to remain close to the original instance, substantially increasing the density of high-quality modifications in the search space. This significantly reduces the difficulty of finding effective modifications, **thus making the training our neural-based instance modifier practical and efficient**.

Extensive experiments on large-scale benchmarks show that TSP-MDF significantly improves the solution quality of traditional heuristic tour constructors with extremely short training time and without ground-truth supervision, demonstrating that our framework is both effective and practical.

2 Preliminary and Related Works

In this paper, we follow most of the previous worksto focus on the classic 2-D Euclidean TSP.

Traveling Salesman Problem (TSP). *Given a TSP instance $s = \{v_1^s, v_2^s, \ldots, v_n^s\}$ with a set of n nodes in the 2-dimensional Euclidean space, where each node i has coordinates $v_i^s = (v_{i,1}^s, v_{i,2}^s) \in$*

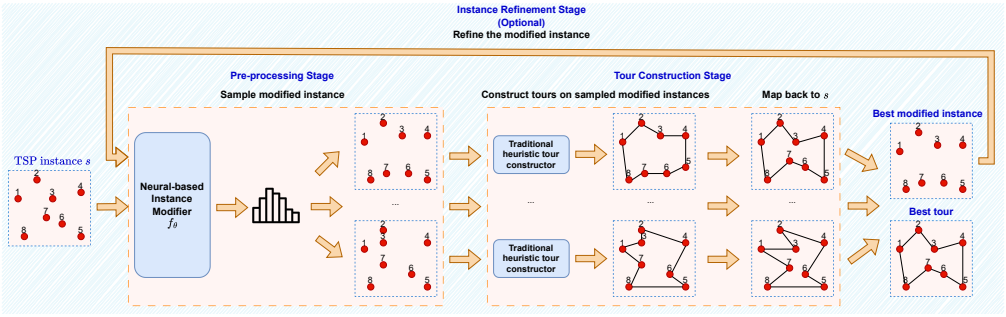


Figure 1: Overview of TSP-MDF. In the pre-processing stage, a neural-based instance modifier estimates a distribution, from which new instances with modified coordinates can be sampled. In the tour construction stage, traditional heuristic tour constructors are executed on modified instances to construct candidate tours and map them back to the original instance. In the optional instance refinement stage, the current best modified instance can be further modified to sample new tours.

$\mathbb{R}^2$, *find a tour* π *(i.e., a permutation of* $\{1, 2, \ldots, n\}$*) that minimizes the total tour length on* s*:*

$$c_s(\pi) = \sum_{i=1}^{n} \|v^s_{\pi_i} - v^s_{\pi_{i+1}}\|_2, \quad \textit{with } \pi_{n+1} = \pi_1 \tag{1}$$

where $\|v^s_{\pi_i} - v^s_{\pi_{i+1}}\|_2$ *is the Euclidean distance between* $v^s_{\pi_i}$ *and* $v^s_{\pi_{i+1}}$.

TSP Tour Construction. Classical heuristics, such as Farthest and Nearest Insertion, iteratively construct tours using distance-based rules. Although computationally efficient, they are deterministic and typically produce only one tour per instance, limiting solution-space exploration. Neural tour constructors (Qiu et al., 2022; Sun & Yang, 2023; Li et al., 2024; Kwon et al., 2020; Joshi et al., 2020; Jin et al., 2023; Li et al., 2023; Hottung et al., 2021; Fu et al., 2021; Kim et al., 2021; Kool et al., 2018; Ye et al., 2023; Li et al., 2025; Goh et al., 2024; Drakulic et al., 2023; Luo et al., 2023; 2025b) instead learn distributions over promising solutions, enabling multiple candidate tours to be sampled. Early approaches (Qiu et al., 2022; Kwon et al., 2020; Joshi et al., 2020; Jin et al., 2023; Hottung et al., 2021; Kim et al., 2021; Kool et al., 2018) primarily use reinforcement learning to construct tours sequentially, but often suffer from low sampling efficiency and limited scalability. More recent diffusion-based methods (Sun & Yang, 2023; Li et al., 2023; 2024) generate diverse, high-quality candidates and achieve promising performance on large-scale instances. However, they rely on supervised training with computationally expensive ground-truth solutions. Moreover, most neural constructors require costly training from scratch, limiting their practicality in real-world settings.

Notably, in this paper, we focus on the tour construction, due to the space limitation, an extended related work for the tour refinement can be found in Appendix A.

3 Proposed Approach: TSP-MDF

3.1 TSP-MDF Overview

In this section, we present an overview of TSP-MDF, which aims to address the question: ***"Can we enhance an existing traditional deterministic heuristic tour constructor by unlocking its guided-sampling capability to approximate multiple alternative better tours, while ensuring practicability?"***

To address the above question, we first tackle a sub-question: ***"How can we guide a traditional deterministic heuristic tour constructor to approximate multiple alternative tours that have the potential to outperform its original tour?"***. Inspired by prompt engineering in LLMs, where input prompts are carefully modified to guide the model toward better responses, our TSP-MDF introduces an additional pre-processing stage prior the initial tour construction stage as shown in Figure 1. Specifically, given an original TSP instance s, our pre-processing stage samples a set of modified instances S_1 by shifting the coordinates of nodes in s. By strategically modifying node coordinates to adjust the relative distances among nodes, complex regions in the original instance s can be simplified, enabling traditional heuristic tour constructors in the subsequent tour construction stage to more easily identify a better tour $\pi^{s_1} = \text{Heuristic}(s_1)$ on each modified instance $s_1 \in S_1$ that is otherwise

difficult to obtain in the original instance. Each new tour π^{s_1} can then be mapped back to the original instance s to yield an alternative tour of potentially shorter length $c_s(\pi^{s_1})$ on s.

Moreover, directly obtaining high-quality modifications can be challenging. Fortunately, our pre-processing stage is not limited to the original instance, it can also be optionally further applied to modify previously modified instances for an arbitrary number of iterations. Specifically, at each iteration t, given an instance s_t obtained after t modifications, we can refine s_t using the pre-processing stage to sample a set of new instances S_{t+1}. Thus, the overall pipeline of TSP-MDF can be viewed as an auto-regressive process starts from $s_0 = s$. Note that, in practice, to sample a set of modified instances S_{t+1} at each iteration t, we adopt a greedy strategy for simplicity and efficiency. Specifically, instead of modifying every $s_t \in S_t$, we sample S_{t+1} by only modifying the current best modified instance s_t^*, which yields the shortest tour length on s so far, such that:

$$s_t^* = \arg \min_{s' \in \{s_{t-1}^*\} \cup S_t} c_s(\pi^{s'}), \quad \text{with} \quad s_0^* = s \tag{2}$$

To this end, TSP-MDF enables both parallel and sequential guided-sampling for traditional heuristic constructors, and this brings to another key sub-question: ***"How can we strategically modify an instance and sample multiple modified instances, while ensuring practicability?"***. Similar to the neural-based heuristic tour constructors, we introduce a neural-based instance modifier f_θ that automatically learns to modify the instance and estimates a distribution over modified instances. Sampling from this distribution produces multiple candidates, each capable of guiding the traditional heuristic constructor toward improved tours. Importantly, our neural-based instance modifier offers a potential practical advantage over neural-based heuristic tour constructors in terms of learning difficulty. This stems from the fact that the space of high-quality tours is extremely small and highly constrained, only a few number of high-quality tours may exist for a given instance, making it challenging for neural tour constructors to accurately discover such solutions, thus often requiring substantial training time. In contrast, under our formulation, a high-quality tour corresponds to multiple distinct instance modifications, and the neural-based instance modifier only needs to discover any one of these effective modifications to guide the base tour constructor toward an improved tour, this substantially increases the tolerance to modeling errors and the probability of success, thereby potentially reducing the learning difficulty. Finally, our goal is to design an unsupervised neural-based instance modifier that can effectively modify instances, but without requiring costly training.

3.2 Basic Design of the Neural-based Instance Modifier f_θ

In this section, we present a basic design of the neural-based instance modifier f_θ, including its sampling strategy and unsupervised training strategy.

Modified instance sampling. In each modification iteration, given an original TSP instance s and a current best modified instance s_t^*, the neural-based instance modifier aims to modify s_t^* to sample a set of modified instances S_{t+1}, where each $s_{t+1} \in S_{t+1}$ contains a set of n nodes $s_{t+1} = \{v_1^{s_{t+1}}, \dots, v_n^{s_{t+1}}\}$, and each node i consists of two modified coordinates $v_i^{s_{t+1}} = (v_{i,1}^{s_{t+1}}, v_{i,2}^{s_{t+1}})$. A basic design is then to take both s and s_t^* as inputs, and learn to modify the instance by estimating a Gaussian distribution $f_\theta^{\Delta v_{i,k}^{s_{t+1}}}(\cdot|s_t^*, s)$ for each coordinate dimension k of each node i, from which a refined coordinate offset $\Delta v_{i,k}^{s_{t+1}}$ can be sampled, such that:

$$f_\theta^{\Delta v_{i,k}^{s_{t+1}}}(\cdot|s_t^*, s) = \mathcal{N}\big(\mu_\theta(\Delta v_{i,k}^{s_{t+1}}|s_t^*, s), \sigma_\theta(\Delta v_{i,k}^{s_{t+1}}|s_t^*, s)^2\big) \tag{3}$$

where f_θ predicts a mean $\mu_\theta(\Delta v_{i,k}^{s_{t+1}}|s_t^*, s)$ and a standard deviation $\sigma_\theta(\Delta v_{i,k}^{s_{t+1}}|s_t^*, s)$ for each coordinate dimension.

A modified instance s_{t+1} can then be sampled by sampling each new coordinate offset $\Delta v_{i,k}^{s_{t+1}}$ from its corresponding distribution and adding it to the corresponding original coordinate in s:

$$\begin{gathered} \Delta v_{i,k}^{s_{t+1}} \sim f_\theta^{\Delta v_{i,k}^{s_{t+1}}}(\cdot|s_t^*, s), \quad i = 1, \dots, n, \ k = 1, 2 \\ s_{t+1} = s + \Delta s_{t+1}, \quad \Delta s_{t+1} = \{(\Delta v_{1,1}^{s_{t+1}}, \Delta v_{1,2}^{s_{t+1}}), \dots, (\Delta v_{n,1}^{s_{t+1}}, \Delta v_{n,2}^{s_{t+1}})\} \end{gathered} \tag{4}$$

where Δs_{t+1} denotes the overall instance modification, and its overall probability (density) is:

$$\Delta s_{t+1} \sim f_\theta^{\Delta s_{t+1}}(\cdot|s_t^*, s), \quad \text{with} \quad f_\theta^{\Delta s_{t+1}}(\Delta s_{t+1}|s_t^*, s) = \prod_{i=1}^{n} \prod_{k=1}^{2} f_\theta^{\Delta v_{i,k}^{s_{t+1}}}(\Delta v_{i,k}^{s_{t+1}} \mid s_t^*, s) \tag{5}$$

Unsupervised training strategy. To train f_θ in an unsupervised manner, we first define its optimization objective. Given an original TSP instance s and a base traditional heuristic tour constructor, f_θ refines s auto-regressively for T iterations, aiming to enable the base tour constructor to produce tours shorter than its original tours on s. To enable T to be chosen flexibly during inference, we adopt a greedy objective that independently maximizes the tour length reduction at each iteration t:

$$J_t(\theta) = \mathbb{E}_{(s_t^*, s)\sim D_t, \Delta s_{t+1}\sim f_\theta^{\Delta s_{t+1}}(\cdot|s_t^*, s)} \left[c_s(\pi^{s_t^*}) - c_s(\pi^{s+\Delta s_{t+1}}) \right] \tag{6}$$

where D_t is the dataset for iteration t. This ensures that each modification iteration can effectively reduce the tour length on s, making f_θ effective under varying iteration budgets.

To optimize $J_t(\theta)$, we adopt the commonly used REINFORCE-based (Williams, 1992) gradient update rule:

$$\nabla_\theta J_t(\theta) = \mathbb{E}_{(s_t^*, s)\sim D_t, \Delta s_{t+1}\sim f_\theta^{\Delta s_{t+1}}(\cdot|s_t^*, s)} \Big[A(s_t^*, s+\Delta s_{t+1}) \cdot \nabla_\theta \log f_\theta^{\Delta s_{t+1}}(\Delta s_{t+1}|s_t^*, s) \Big] \tag{7}$$

where $A(s_t^*, s+\Delta s_{t+1}) = (c_s(\pi^{s_t^*}) - c_s(\pi^{s+\Delta s_{t+1}})) - b(s, t, t+1)$ and $b(s, t, t+1)$ is a baseline function used to reduce the variance of the gradients.

Intuitively, REINFORCE shapes the distribution by increasing the probability of sampling modified instances that lead to larger tour length reductions, while decreasing the probability of less effective ones. In this way, f_θ can be trained without ground-truth labels by auto-regressively sampling instance modifications from $f_\theta^{\Delta s_{t+1}}(\cdot|s_t^*, s)$ for each instance s in the training batch, and updating its parameters at each iteration based on the sampled instance modifications.

Ideally, f_θ can be guided toward sampling high quality modified instances as training progresses. However, the search space of possible instance modifications is extremely large, relying solely on this approach would still require extensive sampling to obtain high-quality modifications, resulting in impractically long training time. In the following sections, we introduce several simple designs to enable f_θ to be trained effectively and practically without incurring costly training.

3.3 Discretization of Coordinate Offset

In Section 3.2, the coordinate offset of each node is modeled as a continuous variable sampled from a Gaussian distribution, whose mean and standard deviation are predicted by f_θ. The mean determines the expected offset, while the standard deviation controls the shape of the distribution around the mean. Although this continuous formulation offers flexibility in candidate offset values, it suffers from several limitations: (1) it introduces a large continuous search space, requiring extensive sampling to discover high-quality modifications, which in turn slows down training; (2) the Gaussian distribution is unimodal, concentrating probability mass around the mean. A small standard deviation limits exploration and may trap f_θ in local optima when the mean is suboptimal, especially early in training; a large one improves exploration but leads to overly random and unstable sampling, reducing control. Thus, the current design lacks flexibility in exploration; (3) sampling exact target values (e.g., 0.00) is infeasible, as a Gaussian assigns zero probability to any exact value. Moreover, accurately estimating precise continuous means (i.e., regression) is difficult. These factors lead to unstable sampling of coordinate offsets, and such instability can significantly alter the structure of the instances, making the sampling process difficult to control.

Improved modified instance sampling. To overcome these issues, we discretize the search space of coordinate offsets. Assuming that the valid range of a coordinate offset is $(-1, 1)$, a continuous coordinate offset $\Delta v_{i,k}^{s_{t+1}}$ can be approximated as a weighted sum of discrete digits at different scales:

$$\Delta v_{i,k}^{s_{t+1}} \approx g(\delta_{i,k}^{s_{t+1}}, \Delta v_{i,k,1}^{s_{t+1}}, ..., \Delta v_{i,k,M}^{s_{t+1}}) = \delta_{i,k}^{s_{t+1}} \cdot \sum_{m=1}^{M} 0.1^m \cdot \Delta v_{i,k,m}^{s_{t+1}} \tag{8}$$

where $\delta_{i,k}^{s_{t+1}} \in \{-1, 1\}$ represents the sign of $\Delta v_{i,k}^{s_{t+1}}$, each $\Delta v_{i,k,m}^{s_{t+1}} \in \{0, 1, 2, \ldots, 9\}$ is a discrete digit, and M is a hyper-parameter that controls the discretization precision.

Through this formulation, the search space of coordinate offset is discretized into multiple sets of candidate digits at different scales, along with a set of candidate sign. For each coordinate dimension k of each node i, f_θ can then be trained to predict a Categorical distribution (i.e., classification) for

the sign $f_\theta^{\delta_{i,k}^{s_{t+1}}}(\cdot|s_t^*, s) \in \mathbb{R}^2$, and for each discrete digit at different scale $f_\theta^{\Delta v_{i,k,m}^{s_{t+1}}}(\cdot|s_t^*, s) \in \mathbb{R}^{10}$. By controlling M, all possible offsets up to M-digit precision can be covered, where each digit set contains only 10 possible values. The sampling of each coordinate offset $\Delta v_{i,k}^{s_{t+1}}$ in Equation 4 can then be rewritten as follows:

$$\begin{gathered}\delta_{i,k}^{s_{t+1}} \sim f_\theta^{\delta_{i,k}^{s_{t+1}}}(\cdot|s_t^*, s), \quad \Delta v_{i,k,m}^{s_{t+1}} \sim f_\theta^{\Delta v_{i,k,m}^{s_{t+1}}}(\cdot|s_t^*, s), \quad m = 1, \ldots, M \\ \Delta v_{i,k}^{s_{t+1}} = g(\delta_{i,k}^{s_{t+1}}, \Delta v_{i,k,1}^{s_{t+1}}, ..., \Delta v_{i,k,M}^{s_{t+1}})\end{gathered} \tag{9}$$

and the overall probability (density) of $\Delta v_{i,k}^{s_{t+1}}$ can be expressed as:

$$f_\theta^{\Delta v_{i,k}^{s_{t+1}}}(\Delta v_{i,k}^{s_{t+1}}|s_t^*, s) = f_\theta^{\delta_{i,k}^{s_{t+1}}}(\delta_{i,k}^{s_{t+1}}|s_t^*, s) \cdot \prod_{m=1}^{M} f_\theta^{\Delta v_{i,k,m}^{s_{t+1}}}(\Delta v_{i,k,m}^{s_{t+1}}|s_t^*, s) \tag{10}$$

This discretized formulation effectively addresses the limitations of the previous continuous formulation: (1) the search space is reduced to a finite space; (2) the Categorical distribution enables flexible exploration beyond the mode of the distribution; (3) the Categorical distribution enables more stable sampling of precise coordinate offset values.

3.4 Guiding f_θ with Self-imitation Learning

Although discretizing coordinate offsets substantially reduces the search space, unsupervised training of f_θ with REINFORCE still requires extensive sampling to converge. In Section 3.2, the basic unsupervised training strategy iteratively samples instance modifications ΔS_{t+1} using f_θ, and updates its parameters based on the the tour length reductions of the sampled modifications via REINFORCE (Eq. 7). However, during early training, f_θ is unguided and tends to generate highly random and overly distorted instances, often producing low-quality tours with negative tour length reductions. Learning from such poor samples yields weak and unstable training signals for cold-start, requiring excessive sampling to discover effective modifications and making the training inefficient.

Improved training strategy. To train f_θ more efficiently, we incorporate self-imitation learning. Specifically, given the current best modification $\Delta s_{t+1}^* = s_{t+1}^* - s$ found by f_θ itself, f_θ is additionally trained to imitate Δs_{t+1}^* by maximizing the likelihood of producing it, such that:

$$J_t^{\mathrm{IL}}(\theta) = \mathbb{E}_{(s_t^*, s, \Delta s_{t+1}^*) \sim D_t}\Big[\log f_\theta^{\Delta s_{t+1}}(\Delta s_{t+1}^*|s_t^*, s)\Big] \tag{11}$$

The overall training objective $J_t^{\mathrm{total}}(\theta)$ can then be expressed as:

$$J_t^{\mathrm{total}}(\theta) = J_t^{\mathrm{RL}}(\theta) + J_t^{\mathrm{IL}}(\theta) \tag{12}$$

where $J_t^{\mathrm{RL}}(\theta)$ is the optimization objective defined in Eq. 6 that maximizes the tour length reduction.

The key of this self-imitation strategy is that when f_θ tends to generate low-quality random modifications in early training, our instance modification formulation provides a natural safeguard: if all sampled instance modifications in ΔS_{t+1} are poor, s_{t+1}^* can be simply set equal to s_t^* (Eq. 2). In other words, f_θ can be initially trained to keep the current best modified instance s_t^* unchanged for cold-start. And as training proceeds with better modifications being discovered, f_θ can then shift to imitate the best sampled modification. Notably, although this cold-start guidance is sub-optimal, it offers a simple yet effective regularization that anchors the search space to slight deviations from the unchanged instance, preventing the model from generating overly random poor-quality modifications and enabling more efficient exploration, which is particularly useful at the early stage of training. And this is reasonable, because modifications are only required for nodes in complex regions where the optimal sub-tour is difficult to find, rather than for every node in the instance. Moreover, it is worth noting that such a safeguard is not available to neural-based heuristic tour constructors that directly predict complete tours, they can only imitate their own poor-quality solutions in the early stage, which often leads to inefficient training.

Notably, Δs_{t+1}^* in this self-imitation learning formulation are often neither near-optimal nor optimal, particularly during the early stages of training, and the quality of Δs_{t+1}^* could vary across different instances in a training batch (some modifications may yield significant improvements in tour length reduction, while others contribute little). Therefore, we weight the imitation objective of each Δs_{t+1}^*

by its corresponding tour length reduction $c_s(\pi^{s_t^*}) - c_s(\pi^{s+\Delta s_{t+1}^*})$. Moreover, since the scale of tour-length reductions can differ across iterations t (larger reductions are more likely at early iterations), we normalize the weights within each iteration. And since each initial guidance might yield no tour-length reduction (i.e., $c_s(\pi^{s_t^*}) - c_s(\pi^{s+\Delta s_{t+1}^*}) = 0$), its normalized weight would always be 0 under the above scheme, causing the model to ignore this cold-start signal, thus we additional assign a fixed small weight to each Δs_{t+1}^*, and the overall weight of each Δs_{t+1}^* is:

$$w^{\Delta s_{t+1}^*} = \frac{c_s(\pi^{s_t^*}) - c_s(\pi^{s_{t+1}^*})}{\sum_{(s_t^*, s, \Delta s_{t+1}^*) \in D_t} c_s(\pi^{s_t^*}) - c_s(\pi^{s_{t+1}^*})} + w^{\text{fixed}} \tag{13}$$

and the objective $J_t^{\text{IL}}(\theta)$ in Eq. 11 can be rewritten as follows:

$$J_t^{\text{IL}}(\theta) \approx \sum_{(s_t^*, s, \Delta s_{t+1}^*) \in D_t} w^{\Delta s_{t+1}^*} \cdot \log f_\theta^{\Delta s_{t+1}}(\Delta s_{t+1}^* | s_t^*, s) \tag{14}$$

3.5 Unifying Target Output Space across Nodes

Given an original instance s and a best modified instance s_t^*, the neural-based instance modifier is currently designed to produce distributions $f_\theta^{\Delta v_{i,k}^{s_{t+1}}}(\cdot | s_t^*, s)$, from which refined coordinate offsets can be sampled and added to the corresponding original coordinate in s (Eq. 4). However, the refined coordinate offset $\Delta v_{i,k}^{s_{t+1}}$ often depends on the previous coordinate offset $\Delta v_{i,k}^{s_t^*}$. Since $\Delta v_{i,k}^{s_t^*}$ can vary significantly not only across nodes within the same TSP instance but also across different instances, the optimal target output space for the refined offsets also differ substantially across nodes. This often leads to slow convergence and long training time before satisfactory performance can be achieved.

To illustrate this, consider a simple case where the target refined coordinate offsets for each node are identical to the current coordinate offset $\Delta v_{i,k}^{s_t^*}$ (i.e., keep the modified instance unchanged). In this case, f_θ only needs to simply reproduce $\Delta v_{i,k}^{s_t^*}$ for $\Delta v_{i,k}^{s_{t+1}}$ for each coordinate dimension of each node. However, since the magnitude of $\Delta v_{i,k}^{s_t^*}$ vary significantly across different nodes and instances, f_θ still requires substantial training time even in this simple scenario. In more complex scenarios, f_θ not only needs to recover the current coordinate offsets but also needs to learn meaningful refinement of the offsets that improve the tour quality on s, which further increases the training difficulty.

Improved modified instance sampling. To unify the target output space across different nodes, we propose to predict the distributions of coordinate offset refinement rather than the distributions of the refined new coordinate offset. In other words, Δs_{t+1} is now set to $\Delta s_{t+1} = s_{t+1} - s_t^*$, rather than $\Delta s_{t+1} = s_{t+1} - s$, and s_{t+1} is now sampled by sampling Δs_{t+1} and adding it to the current modified instance s_t^*, such that:

$$\begin{gathered} \Delta v_{i,k}^{s_{t+1}} \sim f_\theta^{\Delta v_{i,k}^{s_{t+1}}}(\cdot | s_t^*, s), \quad i = 1, \ldots, n, \ k = 1, 2 \\ s_{t+1} = s_t^* + \Delta s_{t+1}, \quad \Delta s_{t+1} = \{(\Delta v_{1,1}^{s_{t+1}}, \Delta v_{1,2}^{s_{t+1}}), \ldots, (\Delta v_{n,1}^{s_{t+1}}, \Delta v_{n,2}^{s_{t+1}})\} \end{gathered} \tag{15}$$

Consider the previous example where no further modification is required: f_θ now only needs to learn that the coordinate offset refinement for all nodes are zero (i.e., $\Delta v_{i,k}^{s_{t+1}} = 0$), and each node shares the same target output. This design effectively eliminates the large variability in the target output space caused by differences in previous coordinate offset across nodes and instances, thereby significantly accelerating the training of f_θ.

Improved training strategy. Accordingly, the optimization objective in Eq. 6 can be rewritten as:

$$J_t(\theta) = \mathbb{E}_{(s_t^*, s) \sim D_t, \Delta s_{t+1} \sim f_\theta^{\Delta s_{t+1}}(\cdot | s_t^*, s)} \left[c_s(\pi^{s_t^*}) - c_s(\pi^{s_t^* + \Delta s_{t+1}}) \right] \tag{16}$$

and the REINFORCE-based gradient update rule (Eq. 7) can then be rewritten as:

$$\nabla_\theta J_t(\theta) = \mathbb{E}_{(s_t^*, s) \sim D_t, \Delta s_{t+1} \sim f_\theta^{\Delta s_{t+1}}(\cdot | s_t^*, s)} \left[A(s_t^*, s_t^* + \Delta s_{t+1}) \cdot \nabla_\theta \log f_\theta^{\Delta s_{t+1}}(\Delta s_{t+1} | s_t^*, s) \right] \tag{17}$$

For the self-imitation learning, the current best modification can be computed as $\Delta s_{t+1}^* = s_{t+1}^* - s_t^*$.

Due to the space limitation, the overall training pipeline of TSP-MDF can be found in Appendix B.1. The network architecture of neural-based instance modifier can be found in Appendix B.2.

Table 1: Results on TSP-$500/1000/10000$. "AS", "MS", and "GS" denote Active Search, Multi-step Sampling, and Gradient Search, respectively. "OOM" indicates that the method ran out of GPU memory (24 GB). "OOT" indicates the the method cannot complete inference within 48 hours.

Method	TSP-500			TSP-1000			TSP-10000		
	Length ↓	Inference Time ↓	Training Time ↓	Length ↓ ↓	Inference Time ↓	Training Time ↓	Length ↓ ↓	Inference Time ↓	Training Time ↓
Concorde	16.55	37m39s	-	23.12	6h39m	-	N/A	N/A	-
LKH-3	16.55	46m40s	-	23.12	2h34m	-	71.77	8h48m	-
DIMES (Unsupervised)	19.10	22s	17m20s	26.37	46s	32m58s	85.90	1m6s	2h5m54s
DIMES+AS (Unsupervised)	17.77	37m10s	17m20s	24.89	1h17m17s	32m58s	**80.62**	59m12s	2h5m54s
DeepACO (Unsupervised)	18.93	16m25s	6m17s	27.47	32m22s	8m33s	OOM	OOM	OOM
Pointerformer (Unsupervised)	**17.14**	53s	547h47m	**24.80**	6m35s	2198h53m	OOM	OOM	OOM
DIFUSCO (Supervised)	17.60	10m46s	129h10m	25.09	45m17s	295h50m	93.70	1h25m15s	OOM
FastT2T (Supervised)	17.68	25s	260h	24.88	1m45s	OOM	**90.84**	18m56s	OOM
FastT2T+MS+GS (Supervised)	**16.71**	6m29s	260h	OOM	OOM	OOM	OOM	OOM	OOM
BQ-NCO (Supervised)	16.73	4m32s	54h31m	**23.65**	19m38s	54h31m	OOT	OOT	OOT
LEHD (Supervised)	16.80	1m21s	139h35m	23.84	3m45s	139h35m	91.33	3h5m10s	139h35m
Nearest Insertion	20.62	0.19s	-	28.95	0.72s	-	90.49	8.34s	-
TSP-MDF ($T = 1$) + Nearest	20.41	3s	2m01s	28.78	7s	5m47s	90.38	55s	1h37m31s
TSP-MDF ($T = 30$) + Nearest	18.94	1m11s	2m01s	27.09	3m6s	5m47s	88.86	24m52s	1h37m31s
Farthest Insertion	18.29	0.17s	-	25.73	0.65s	-	80.69	8.43s	-
TSP-MDF ($T = 1$) + Farthest	18.04	3s	1m55s	25.49	6s	5m21s	80.47	57s	1h35m34s
TSP-MDF ($T = 30$) + Farthest	**17.26**	1m6s	1m55s	**24.44**	2m34s	5m21s	**78.91**	25m56s	1h35m34s

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Datasets. To evaluate our TSP-MDF, we closely follow the standard procedure of previous works (Fu et al., 2021; Qiu et al., 2022; Sun & Yang, 2023; Li et al., 2023; 2024; Ye et al., 2024; Jin et al., 2023; Ye et al., 2023) to conduct experiments on large-scale TSP-n benchmark testsets of $500/1000/10000$ nodes generated by (Fu et al., 2021). Specifically, TSP-500 and TSP-1000 each contain 128 2-D Euclidean TSP test instances, and TSP-10000 contains 16 2-D Euclidean TSP test instances. For each dataset of n nodes, we generate the corresponding training instances on the fly by sampling n nodes independently from a uniform distribution over the unit square $[0, 1]^2$. An extra evaluation on the real-world benchmarks TSPLib (Reinelt, 1991) can be found in Appendix D.4.

Baselines. To evaluate how TSP-MDF enhances traditional deterministic heuristic tour constructors, we adopt the following most famous heuristic tour constructors as the base constructors: (1) **Nearest Insertion** determines node insertion order based on the nearest pairwise distance, and (2) **Farthest Insertion** determines node insertion order based on the farthest pairwise distance.

For reference, we obtain ground-truth tours using traditional exact/near-optimal TSP solvers. We also provide the performance of neural-based heuristic tour constructors (details are in Appendix C.1).

Evaluation Metrics. We evaluate our TSP-MDF based on the following metrics: (1) **Length**: the average length of the best approximated tours; (2) **Inference Time**: the total running time for solving all instances; (3) **Training Time**: the total training time of the neural-based approach.

4.2 OVERALL PERFORMANCE

In this section, we evaluate the overall performance of TSP-MDF, both with and without the optional instance refinement stage. Since the primary focus of this work is on the TSP tour construction, we disable the optional tour refinement stage to ensure a fair and clear comparison. Notably, the goal of this work is not to outperform all neural-based heuristic tour constructors that require extensive training in terms of solution quality. Instead, we aim to evaluate how TSP-MDF can enhance the solution quality of base traditional heuristic tour constructors with a practically short training time.

Table 1 shows the performance of TSP-MDF that samples 100 modified instances for each iteration, it is obviously that the solution quality of both Farthest and Nearest Insertion is significantly enhanced by unlocking the guided-sampling through TSP-MDF. In particular, with the optional instance refinement stage ($T = 30$), by leveraging both parallel and sequential guided-sampling strategies, the enhanced Farthest Insertion can achieve solution quality comparable to neural-based heuristic tour constructors. Notably, our instance modifier adopts the same network architecture as DIMES, and similar to the network architecture used in DIFUSCO and FastT2T. Moreover, although TSP-MDF introduces longer inference time compared to original heuristics, the overall inference time remains efficient compared to neural-based heuristic tour constructors. Regarding the practicability, TSP-MDF does not require any ground-truth supervision and has extremely short training time. Specifically, training the neural-based instance modifier on TSP-$500/1000$ takes less than 10 minutes on a single

RTX 3090 GPU, while most of the neural-based heuristic tour constructors require several days of training. And the neural-based heuristic tour constructors (i.e., DIMES, DeepACO) that does not require costly training yields inferior performance compared to our TSP-MDF–enhanced Farthest Insertion. This highlights that TSP-MDF is both effective, efficient and practical.

4.3 CAN TSP-MDF ENHANCE THE TRADITIONAL HEURISTIC TOUR CONSTRUCTORS?

Given a base traditional deterministic heuristic tour constructor, our TSP-MDF aims to enhance it by unlocking its guided-sampling capability through instance modification, thereby increasing its chance of escaping local optima to discover alternative better tours. To verify whether our TSP-MDF can really enhance the base tour constructor, we evaluate the following two key questions: Q(1) **"Can guided-sampling on modified instances effectively help the base tour constructor escape local optima?"**; Q(2) **"Is guided-sampling necessary, or would random sampling suffice?"**.

Q(1). A reasonable way to evaluate Q(1) is to iteratively refine both the tours constructed by the original base tour constructor and TSP-MDF-enhanced tour constructor via local search (i.e., optional tour refinement stage). If both refined tours converge to similar tours, it suggests that the tour constructed by the TSP-MDF–enhanced tour constructor remains within the same or a nearby local optima as that of the original tour constructor, indicating that the base tour constructor has not been successfully enhanced. To conduct this evaluation, we adopt the popular 2-opt (Croes, 1958) local search to refine the tours. Table 2 shows the length of refined tours on each dataset. Notably, since 2-opt is computationally more expensive than the base tour constructor, for simplicity, here we only refine the best tour found by TSP-MDF–enhanced tour constructor instead of all sampled tours. It can be observed that the tours found by TSP-MDF–enhanced tour constructors converge to significantly shorter refined tours than those produced by the original base tour constructors. This indicates that TSP-MDF effectively helps the base tour constructors escape local optima and guides them toward better regions in the solution space, thereby successfully enhancing their performance.

Table 2: Evaluation on length of tours refined by 2-opt (Croes, 1958) local search.

Method	TSP-500	TSP-1000	TSP-10000
Nearest Insertion + 2-opt	18.82	26.35	84.35
TSP-MDF (T=1) + Nearest + 2-opt	18.78	26.30	84.28
TSP-MDF (T=30) + Nearest + 2-opt	18.33	25.85	83.35
Farthest Insertion + 2-opt	18.05	25.40	79.62
TSP-MDF (T=1) + Farthest + 2-opt	17.84	25.22	79.50
TSP-MDF (T=30) + Farthest + 2-opt	17.23	24.38	78.34

Q(2). To answer Q(2), we construct a variant model, TSP-MDF (random), which removes the neural-based instance modifier and instead samples discrete coordinate offsets randomly. To better demonstrate the behavior of the variant models, we report the average tour length of the best approximated tour both including/excluding the tour computed on the original s as one of the candidate tours. Notably, including s evaluates how the overall performance of the base tour constructor can be enhanced, whereas excluding s evaluates the pure quality of the modified instances. As shown in Table 3, without guided-sampling, the average tour length cannot be improved compared to the original tour constructor (i.e., all modified instances are worse than s). This is because the randomly sampled modified instances distort the original instance in an uncontrolled manner, misleading the base tour constructor to approximate poor-quality tours of extremely long lengths (e.g., larger than 4000 on TSP-10000). Thus, random sampling fails to help the base tour constructor escape local optima, highlighting the necessity of our proposed guided-sampling strategy. Together, the results from Q(1) and Q(2) consistently demonstrate that TSP-MDF can effectively enhance traditional deterministic tour constructors. Due to the space limitation, more experimental results can be found in Appendix D.

Table 3: Evaluation on random instance modification.

Method	TSP-500	TSP-1000	TSP-10000
Nearest Insertion	20.62	28.95	90.49
TSP-MDF (random) (T=30) + Nearest	20.62/203.69	28.95/416.06	90.49/4325.00
TSP-MDF (T=30) + Nearest	18.94/18.94	27.09/27.09	88.86/88.86
Farthest Insertion	18.29	25.73	80.69
TSP-MDF (random) (T=30) + Farthest	18.29/203.26	25.73/415.60	80.69/4323.17
TSP-MDF (T=30) + Farthest	17.26/17.26	24.44/24.44	78.91/78.91

5 CONCLUSION

This paper presents TSP-MDF, a novel instance modification framework for TSP. By leveraging neural-based instance modifier for guided-sampling, TSP-MDF enables traditional deterministic heuristic tour constructors to generate multiple high-quality candidate tours from modified instances, helping them escape local optima and improve solution quality. Meanwhile, the proposed formulation allows the instance modifier to be trained efficiently and practicality in an unsupervised manner. Extensive experiments on large-scale benchmarks demonstrate that TSP-MDF significantly enhances traditional heuristic tour constructors, while being practical with extremely short training time.

AI USE STATEMENT

In this work, we used generative AI tools solely to edit the manuscript for grammar, wording, conciseness, and readability. We did not use generative AI tools for any research tasks requiring disclosure, including developing research ideas or methodology, formulating claims, implementing methods, designing experiments, or interpreting results. All AI-assisted edits were carefully reviewed by the authors to ensure that they preserved the intended technical meaning and did not introduce unsupported claims or inaccurate information. The authors take full responsibility for the final content of the manuscript.

REFERENCES

David Applegate, Ribert Bixby, Vasek Chvatal, and William Cook. Concorde TSP solver. `https://www.math.uwaterloo.ca/tsp/concorde/index.html`, 2006.

Xavier Bresson and Thomas Laurent. An experimental study of neural networks for variable graphs. 2018.

IBM ILOG Cplex. V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157, 2009.

Georges A Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6): 791–812, 1958.

Darko Drakulic, Sofia Michel, Florian Mai, Arnaud Sors, and Jean-Marc Andreoli. Bq-nco: Bisimulation quotienting for efficient neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:77416–77429, 2023.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 7474–7482, 2021.

Yong Liang Goh, Zhiguang Cao, Yining Ma, Yanfei Dong, Mohammed Haroon Dupty, and Wee Sun Lee. Hierarchical neural constructive solver for real-world tsp scenarios. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 884–895, 2024.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL `https://www.gurobi.com`.

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.

André Hottung, Yeong-Dae Kwon, and Kevin Tierney. Efficient active search for combinatorial optimization problems. *arXiv preprint arXiv:2106.05126*, 2021.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

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.

Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *arXiv preprint arXiv:2006.07054*, 2020.

Minsu Kim, Jinkyoo Park, et al. Learning collaborative policies to solve np-hard routing problems. *Advances in Neural Information Processing Systems*, 34:10418–10430, 2021.

Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

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

Qi Li, Zhiguang Cao, Yining Ma, Yaoxin Wu, and Yue-Jiao Gong. Diversity optimization for travelling salesman problem via deep reinforcement learning. *arXiv preprint arXiv:2501.00884*, 2025.

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

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems*, 36:8845–8864, 2023.

Fu Luo, Xi Lin, Yaoxin Wu, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Boosting neural combinatorial optimization for large-scale vehicle routing problems. In *The Thirteenth 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. *arXiv preprint arXiv:2505.13904*, 2025b.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 35:25531–25546, 2022.

Gerhard Reinelt. Tsplib—a traveling salesman problem library. *ORSA journal on computing*, 3(4): 376–384, 1991.

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

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. Deepaco: Neural-enhanced ant systems for combinatorial optimization. *Advances in neural information processing systems*, 36: 43706–43728, 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.

APPENDIX

A EXTENDED RELATED WORKS

A.1 EXACT METHODS FOR TSP

Traditional exact solvers for the TSP (Gurobi Optimization, LLC, 2024; Applegate et al., 2006; Cplex, 2009) are mainly built upon linear programming and mixed integer linear programming

frameworks. Among them, Concorde (Applegate et al., 2006) is one of the most famous state-of-the-art exact solvers based on an advanced branch-and-cut algorithm. Despite their strong optimality guarantees, these exact solvers are computationally expensive and become impractical for large-scale TSP instances.

A.2 Approximation Methods for TSP

To improve efficiency, TSP solutions are commonly approximated using a two-stage framework: an initial tour construction stage that generates one or more candidate tours, followed by an optional tour refinement stage to improve solution quality.

TSP Tour construction. In the construction stage, classical heuristics such as Farthest and Nearest Insertion iteratively build tours by inserting nodes according to distance-based rules. These methods are computationally efficient and produce reasonable solutions, but are inherently deterministic and typically generate only a single tour per instance, limiting exploration.

Recently, neural-based heuristic tour constructors (Qiu et al., 2022; Sun & Yang, 2023; Li et al., 2024; Kwon et al., 2020; Joshi et al., 2020; Jin et al., 2023; Li et al., 2023; Hottung et al., 2021; Fu et al., 2021; Kim et al., 2021; Kool et al., 2018; Ye et al., 2023; Li et al., 2025; Goh et al., 2024; Drakulic et al., 2023; Luo et al., 2023; 2025b) have emerged as popular alternatives for constructing TSP tours. These methods aim to estimate distributions that highlight promising regions of the solution space, from which multiple candidate tours can be sampled and explored. Early neural-based frameworks (Qiu et al., 2022; Kwon et al., 2020; Joshi et al., 2020; Jin et al., 2023; Hottung et al., 2021; Kim et al., 2021; Kool et al., 2018) primarily focus on developing unsupervised reinforcement learning methods to sequentially sample and insert nodes into partial tours. While effective on small-scale instances, these approaches often struggle to scale to larger instances due to the combinatorial complexity of the solution space and the low sampling efficiency of reinforcement learning. More recently, generative diffusion-based models (Sun & Yang, 2023; Li et al., 2023; 2024) trained with supervised learning have demonstrated promising performance on large-scale TSP instances. By generating multiple diverse and high-quality distributions for a given instance, these models are able to effectively capture the multimodal nature of the problem. However, they rely heavily on supervised training and require access to high-quality ground-truth solutions, which are computationally expensive to obtain in practice. Moreover, existing neural-based heuristic tour constructors typically need to be trained entirely from scratch and often incur substantial training costs before achieving satisfactory performance, even on relatively small instances, which limits their practicality in real-world settings.

TSP Tour refinement. To refine the candidate tours constructed by heuristic tour constructors, classical local search methods, such as 2-opt (Croes, 1958) and LKH-3 (Helsgaun, 2017), iteratively refine the tour through edge exchange operations. In particular, LKH-3 can achieve near-optimal solution quality by leveraging powerful k-opt moves combined with extensive multi-start strategies over a large number of randomized initial tours; however, this typically incurs substantial computational cost. More recently, learning-based approaches (Ye et al., 2024; Kim et al., 2021; Fu et al., 2021; Ye et al., 2023; Luo et al., 2025a) have been explored for the tour refinement stage. Specifically, Monte Carlo Tree Search (MCTS) (Fu et al., 2021) has been adopted for tour improvement by searching over the solution distribution estimated by neural-based heuristic tour constructors. Methods such as LCP (Kim et al., 2021) and GLOP (Ye et al., 2024) propose to decompose an initial tour into sub-tours and apply neural networks trained via reinforcement learning to refine these substructures. DeepACO (Ye et al., 2023) introduces neural-guided solution perturbations to enhance the traditional local search. SIL (Luo et al., 2025a) further proposes self-improvement training to enhance the unsupervised training of learning-based tour refinement methods on large-scale instances. These learning-based approaches offer greater flexibility and have the potential to improve both the effectiveness and efficiency of tour refinement, but most of them require training additional neural networks. Nevertheless, the effectiveness of all existing refinement methods remains highly dependent on the quality of the initial tours. When the initial tour is suboptimal, the refinement process may converge to another suboptimal tour within the same local basin and may require a large number of refinement steps, leading to high computational overhead compared to initial tour construction. Consequently, improving the quality of initial tours remains a critical and complementary problem for effective TSP approximation.

Algorithm 1 TSP-MDF Training Procedure

Input: a neural instance modifier f_θ, a traditional heuristic tour constructor $\mathrm{Heuristic}(\cdot)$, number of instance modification iterations T, number of training epochs E;

for $e = 1$ to E **do**
  Randomly sample a batch of TSP instances D;
  Compute $\pi^s \leftarrow \mathrm{Heuristic}(s)$ for each $s \in D$;
  Initialize $s_0^* \leftarrow s$ for each $s \in D$;
  for $t = 0$ to $T - 1$ **do**
    for each $s \in D$ **do**
      Sample a set of instance modifications ΔS_{t+1} from $f_\theta^{\Delta s_{t+1}}(\cdot|s_t^*, s)$ according to Eq. 9, Eq. 15;
      Obtain a set of modified instances $S_{t+1} \leftarrow \{\, s_t^* + \Delta s_{t+1} \mid \Delta s_{t+1} \in \Delta S_{t+1} \,\}$;
      Compute $\pi^{s_{t+1}} \leftarrow \mathrm{Heuristic}(s_{t+1})$ for each $s_{t+1} \in S_{t+1}$;
      $s_{t+1}^* \leftarrow \arg\min_{s' \in \{s_t^*\} \cup S_{t+1}} c_s(\pi^{s'})$;
      $\Delta s_{t+1}^* \leftarrow s_{t+1}^* - s_t^*$;
    end for
    Update f_θ by taking gradient step on $J_t^{\mathrm{total}}(\theta)$ using each $(s, s_t^*, \Delta S_{t+1}, \Delta s_{t+1}^*)$ according to Eq. 12, Eq. 14, Eq. 16, Eq. 17;
  end for
end for

B Detailed Methods

B.1 Training Algorithm

The training pipeline of TSP-MDF is illustrated in Algorithm 1. Note that, for simplicity, a fixed number of iterations T is used during training.

B.2 Model Architecture of Neural-based Instance Modifier

For an original TSP instance s, in each iteration t, given a current best modified instance s_t^* that consists of a set of modified node coordinates, the neural-based instance modifier f_θ takes s and s_t^* as inputs, and aims to further modify s_t^* by predicting the distributions of coordinate offsets. For a fair comparison with neural-based heuristic tour constructors, we follow a similar two-step pipeline commonly used in existing neural-based heuristic tour constructors (Joshi et al., 2020; Fu et al., 2021; Li et al., 2024; Sun & Yang, 2023; Qiu et al., 2022), where we first jointly model s and s_t^* to construct a single graph G_t that consists of a set of nodes and edges, then apply a graph neural network to G_t to estimate the distributions of coordinate offsets for each node.

Graph construction. To efficiently model s and s_t^* jointly as a single graph at each iteration, we first construct a static base graph G based on the original TSP instance s, and then dynamically incorporate information from the modified instance s_t^* into G to obtain G_t.

To construct the static base graph G, we initialize the feature of each node i in G, denoted as $\boldsymbol{x}_i$, with its corresponding $v_i^s = (v_{i,1}^s, v_{i,2}^s)$ in s. We then construct edges by connecting each node to a fixed number of its nearest neighbors according to their Euclidean distances in s, which prevents quadratic edge growth in large graphs. The edge feature $\boldsymbol{e}_{ij}$ for each edge (i, j) in G is set to the corresponding Euclidean distance $\|v_i^s - v_j^s\|_2$.

At each iteration t, we then update G to G_t by concatenating each node feature x_i in G with the information derived from its corresponding modified coordinates $v_i^{s_t^*} = (v_{i,1}^{s_t^*}, v_{i,2}^{s_t^*})$ in s_t^*. Specifically, for each node i, we first compute its total coordinate offsets $v_i^{s_t^*} - v_i^s = (v_{i,1}^{s_t^*} - v_{i,1}^s, v_{i,2}^{s_t^*} - v_{i,2}^s)$, then discretize each input offset $v_{i,k}^{s_t^*} - v_{i,k}^s$ into $M + 1$ discrete components via $g^{-1}(v_{i,k}^{s_t^*} - v_{i,k}^s)$ (Eq. 8) to capture the variations in the input offsets more precisely. Finally, each discrete component is transformed into a one-hot vector, and the feature of node i in G_t, denoted as $\boldsymbol{x}_i^t$, is then obtained by concatenating all these one-hot vectors with its corresponding node feature x_i in G. For simplicity and computational efficiency, G_t preserves the edge set of G, with each edge feature denoted as $\boldsymbol{e}_{ij}^t$.

Network architecture. Given the constructed graph G_t, for a clear evaluation of our framework, we adopt one of the most common network architectures used in previous neural-based heuristic tour constructors (Joshi et al., 2020; Li et al., 2024; Sun & Yang, 2023; Qiu et al., 2022), where an anisotropic graph neural networks (AGNN) is adopted as the backbone of f_θ to simultaneously compute the embeddings of nodes and edges in G_t through an edge-gating mechanism (Bresson & Laurent, 2018).

Let $\boldsymbol{h}_i^l$ and $\boldsymbol{h}_{ij}^l$ denote the embeddings of node i and edge (i, j) at layer l of AGNN, respectively. The initial embeddings $\boldsymbol{h}_i^0$ and $\boldsymbol{h}_{ij}^0$ are obtained by applying nonlinear transformations to the corresponding node features $\boldsymbol{x}_i^t$ and edge features $\boldsymbol{e}_{ij}^t$ in G_t, respectively:

$$\boldsymbol{h}_i^0 = \alpha(\boldsymbol{\theta}_x \boldsymbol{x}_i^t), \quad \boldsymbol{h}_{ij}^0 = \alpha(\boldsymbol{\theta}_e \boldsymbol{e}_{ij}^t) \tag{18}$$

where α denotes the SiLU activation(Elfwing et al., 2018), and $\boldsymbol{\theta}_x$, $\boldsymbol{\theta}_e$ are learnable parameters.

At each layer of AGNN, the embeddings of nodes and edges are updated following an anisotropic message passing scheme:

$$\begin{aligned} \boldsymbol{h}_i^l &= \boldsymbol{h}_i^{l-1} + \alpha(\mathrm{BN}(\boldsymbol{\theta}_1^l h_i^{l-1} + \mathcal{A}_{j\in\mathcal{N}(i)}(\sigma(\boldsymbol{h}_{ij}^{l-1}) \odot \boldsymbol{\theta}_2^l \boldsymbol{h}_j^{l-1}))) \\ \boldsymbol{h}_{ij}^l &= \boldsymbol{h}_{ij}^{l-1} + \alpha(\mathrm{BN}(\boldsymbol{\theta}_3^l \boldsymbol{h}_{ij}^{l-1} + \boldsymbol{\theta}_4^l \boldsymbol{h}_i^{l-1} + \boldsymbol{\theta}_5^l \boldsymbol{h}_j^{l-1})) \end{aligned} \tag{19}$$

where $\boldsymbol{\theta}_1^l$, $\boldsymbol{\theta}_2^l$, $\boldsymbol{\theta}_3^l$, $\boldsymbol{\theta}_4^l$, $\boldsymbol{\theta}_5^l$ are learnable parameters at layer l, BN denotes the Batch Normalization (Ioffe & Szegedy, 2015), $\mathcal{A}$ represents the aggregation function (e.g., mean pooling), $\mathcal{N}(i)$ denotes the neighborhoods of node i, σ is the sigmoid function, and $\odot$ is the Hadamard product.

After a L-layer AGNN, a MLP is applied to each node embedding $\boldsymbol{h}_i^L$ to predict a vector of dimension $2(10M + 2)$, which is then split into $2(M + 1)$ probability distributions corresponding to each discretized component of the coordinate offset.

C Experimental Settings

C.1 Baselines

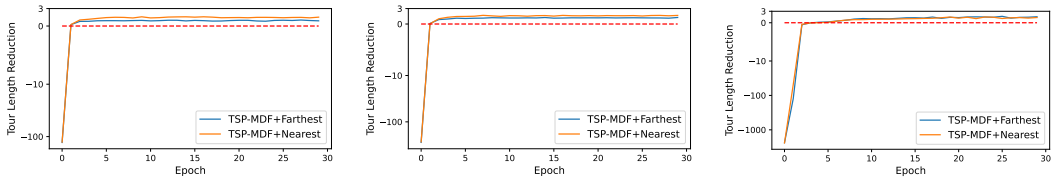
To evaluate how our proposed TSP-MDF enhances traditional deterministic heuristic tour constructors, we adopt the following classical heuristic tour constructors as the base constructors: (1) **Nearest Insertion** determines node insertion order based on the nearest pairwise distance, and (2) **Farthest Insertion** determines node insertion order based on the farthest pairwise distance.

For a clear reference, we obtain ground-truth solutions using the following traditional exact/near-optimal TSP solvers: (1) **Concorde** (Applegate et al., 2006) is an exact TSP solver, and (2) **LKH-3** (Helsgaun, 2017) is a near-optimal solver that iteratively refines multiple random initial tours via k-opt operations.

We further compare the TSP-MDF-enhanced tour constructors with both supervised and unsupervised neural-based heuristic tour constructors. Unsupervised neural-based heuristic tour constructors include: (1) **DIMES** (Qiu et al., 2022), (2) **DeepACO** (Ye et al., 2023), and (3) **Pointerformer** (Jin et al., 2023). Supervised neural-based heuristic tour constructors include: (1) **DIFUSCO** (Sun & Yang, 2023), (2) **FastT2T** (Li et al., 2024), (3) **BQ-NCO** (Drakulic et al., 2023), and (4) **LEHD** (Luo et al., 2023).

Similar to the optional instance refinement stage of our TSP-MDF, some neural-based heuristic tour constructors also support optional sequential tour sampling strategies during inference, i.e., **DIMES** can employ **Active Search** (Qiu et al., 2022; Hottung et al., 2021), while **FastT2T** can adopt **Multi-step Sampling** (Li et al., 2024) and **Gradient Search** (Li et al., 2024).

Notably, some of the latest neural-based heuristic tour constructors (e.g., (Goh et al., 2024; Li et al., 2025; Luo et al., 2025b)) are not included in our comparison, as they are either not publicly available, lack pre-trained checkpoints and require extremely long training time, or were originally evaluated only on small-scale instances and cannot be feasibly trained on our large-scale datasets using a single NVIDIA RTX 3090 (24GB) GPU.



(a) TSP-500 (b) TSP-1000 (c) TSP-10000

Figure 2: Average tour length reduction of TSP-MDF-enhanced traditional heuristic tour constructors on training TSP instances.

Table 4: Decomposition of training time for each instance modification iteration.

Method	Component	TSP-500	TSP-1000	TSP-10000
TSP-MDF+Nearest	Neural-based instance modifier	0.066s	0.11s	0.30s
	Nearest Insertion	0.064s	0.27s	6.20s
	Total	0.13s	0.38s	6.5s
TSP-MDF+Farthest	Neural-based instance modifier	0.062s	0.12s	0.24s
	Farthest Insertion	0.058s	0.24s	6.13s
	Total	0.12s	0.36s	6.37s

C.2 Implementation Details

Training. The neural-based instance modifier is trained using the AdamW optimizer (Loshchilov & Hutter, 2017) with a learning rate of 0.001. We set the total number of training epochs $E = 30$. In each training epoch, we sample 16 original TSP instances (i.e., batch size) for the 500-node and 1000-node datasets, and 4 instances for the $10,000$-node dataset. The total number of modification iterations per training epoch is set to $T = 30$, and $|S_t| = 50$ modified instances are sampled in each iteration for each original instance, with the discretization precision of sampling is set to $M = 4$ for the 500-node and 1000-node datasets, and $M = 6$ for the $10,000$-node dataset. For the training objective, we simply set $w^{\text{fixed}} = 0.01$.

Model architecture. For graph construction, each node is connected to its 50 nearest neighbors. For the network parameters of neural-based instance modifier, we follow previous neural-based heuristic tour constructor (Qiu et al., 2022) to adopt a 12-layer AGNN with a hidden dimension of 32.

C.3 Computational Resources

All experiments are conducted on a single Nvidia GeForce RTX 3090 (24GB) GPU and an Intel i9-12900K 24-core CPU with 128GB RAM. In our TSP-MDF framework, the base traditional heuristic tour constructors are implemented in C++ and executed on the CPU, while the neural-based instance modifier is implemented in PyTorch and runs on the GPU.

C.4 Source Code

The source code is available at `https://anonymous.4open.science/r/TSP-MDF-F214`.

D Additional Experimental Results

D.1 Practicability Analysis: Training Efficiency

In this section, we analyze the practicability of our TSP-MDF in terms of the training efficiency, which can be decomposed into two components: the training time and the training dynamic.

Training time. Table 4 presents the breakdown of training time for each instance modification iteration (the inference time follows the similar decomposition). For the dataset with 500 nodes (i.e.,

Table 5: Effect of parallel sampling $|S_t|$ and sequential steps T on tour length/inference time.

(a) TSP-500 Nearest Insertion

$\lvert S_t\rvert \setminus T$	1	10	20	30	40	50
10	20.54/1s	20.06/10s	19.78/19s	19.61/29s	19.48/39s	19.38/48s
50	20.46/2s	19.74/16s	19.37/31s	19.16/46s	19.02/60s	18.92/76s
100	20.41/3s	19.63/26s	19.23/52s	18.94/71s	18.86/103s	18.77/128s
300	20.39/6s	19.47/56s	19.06/111s	18.84/166s	18.69/221s	18.61/277s
500	20.36/8s	19.41/83s	18.99/166s	18.76/250s	18.63/333s	18.53/416s

(b) TSP-500 Farthest Insertion

$\lvert S_t\rvert \setminus T$	1	10	20	30	40	50
10	18.19/1s	17.80/8s	17.64/16s	17.56/25s	17.50/33s	17.46/41s
50	18.07/2s	17.54/15s	17.38/30s	17.31/44s	17.26/59s	17.24/74s
100	18.04/3s	17.47/22s	17.31/42s	17.26/66s	17.21/84s	17.20/106s
300	17.98/5s	17.39/53s	17.25/106s	17.19/160s	17.17/213s	17.16/266s
500	17.96/7s	17.35/72s	17.21/144s	17.17/216s	17.15/289s	17.14/361s

(c) TSP-1000 Nearest Insertion

$\lvert S_t\rvert \setminus T$	1	10	20	30	40	50
10	28.87/3s	28.44/25s	28.13/50s	27.90/74s	27.73/99s	27.59/124s
50	28.81/5s	28.10/37s	27.65/72s	27.34/109s	27.11/144s	26.94/180s
100	28.78/7s	27.99/56s	27.49/113s	27.09/186s	26.91/226s	26.72/282s
300	28.75/14s	27.81/139s	27.24/278s	26.88/416s	26.62/555s	26.43/693s
500	28.73/22s	27.75/219s	27.17/437s	26.80/655s	26.54/873s	26.34/1090s

(d) TSP-1000 Farthest Insertion

$\lvert S_t\rvert \setminus T$	1	10	20	30	40	50
10	25.63/3s	25.22/21s	25.01/41s	24.89/62s	24.71/82s	24.73/102s
50	25.53/4s	24.95/34s	24.70/69s	24.57/103s	24.48/137s	24.42/170s
100	25.49/6s	24.86/53s	24.59/106s	24.44/154s	24.37/211s	24.32/263s
300	25.44/13s	24.72/127s	24.46/253s	24.32/378s	24.24/505s	24.19/632s
500	25.41/25s	24.66/247s	24.39/494s	24.26/740s	24.19/986s	24.15/1233s

(e) TSP-10000 Nearest Insertion

$\lvert S_t\rvert \setminus T$	1	10	20	30	40	50
10	90.44/14s	90.09/99s	89.78/194s	89.58/290s	89.41/384s	89.27/480s
50	90.40/33s	89.83/291s	89.38/578s	89.03/863s	88.76/1151s	88.55/1438s
100	90.38/55s	89.74/706s	89.24/1396s	88.86/1492s	88.53/2794s	88.26/3493s
300	90.36/213s	89.59/2100s	89.00/4231s	88.56/6381s	88.22/8540s	87.90/10691s
500	90.37/367s	89.57/3580s	88.94/7111s	88.47/10721s	88.08/14309s	87.76/17881s

(f) TSP-10000 Farthest Insertion

$\lvert S_t\rvert \setminus T$	1	10	20	30	40	50
10	80.62/14s	80.21/103s	79.97/203s	79.77/303s	79.62/403s	79.48/503s
50	80.53/35s	79.89/303s	79.50/597s	79.23/893s	79.01/1188s	78.84/1483s
100	80.47/57s	79.79/520s	79.39/1035s	78.91/1556s	78.84/2068.98s	78.66/2588s
300	80.48/198s	79.66/1978s	79.17/3988s	78.82/5982s	78.56/7988s	78.34/9982s
500	80.45/234s	79.55/2327s	79.03/4648s	78.69/6962s	78.42/9270s	78.19/11581s

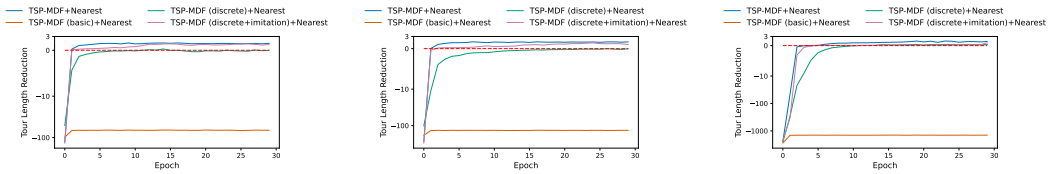
TSP-500), the training time of the neural-based instance modifier (including both forward pass and backward propagation) is similar to the running time of the base heuristic tour constructor. As the instance size increases, the running time of the base tour constructor grows rapidly and dominates the overall training time, whereas the training time of our neural-based instance modifier remains efficient even for TSP-10000, which demonstrates the efficiency of neural-based instance modifier. Notably, the base heuristic tour constructor approximates tours on multiple sampled modified instances in parallel on the CPU, thus this process can be further accelerated by running it on a CPU with more cores.

Training dynamic. Figure 2 shows the average tour length reduction on the training TSP instances achieved by TSP-MDF-enhanced traditional heuristic tour constructors. Notably, the tour length reduction for each base tour constructor is computed by the difference between the length of tour originally approximated on s and the length of the best tour found among all modified instances across all iterations (excluding s) sampled by the neural-based instance modifier. It can be observed that the neural-based instance modifier initially generates instance modifications of poor quality that mislead the base tour constructor to approximate extremely long tours, resulting in negative tour length reductions. But benefiting from our design, within only a few epochs, the neural-based instance modifier can quickly correct the direction and learn to generate meaningful instance modifications that result in positive tour reductions.

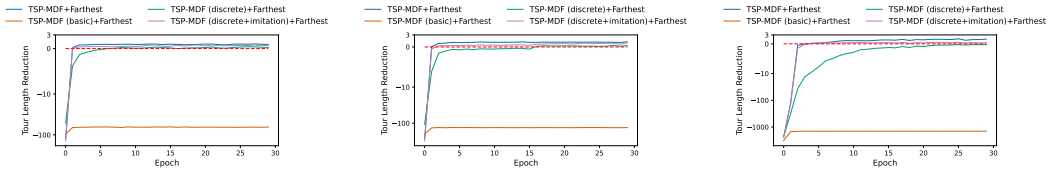
Taken together, both the computational efficiency and learning capability of our neural-based instance modifier allow it to be trained efficiently, thus making TSP-MDF a practical framework.

D.2 Parallel vs. Sequential Guided-sampling

Given a base traditional heuristic tour constructor, our TSP-MDF enhances it by leveraging both parallel and sequential guided sampling. Table 5 investigates the effects of different numbers of parallel samples $|S_t|$ and sequential sampling steps T on both solution quality and efficiency. First, it is evident that increasing either parallel or sequential sampling generally leads to shorter tour lengths produced by the base heuristic tour constructor, with linearly increased inference time. However, when relying solely on parallel sampling without sequential refinement (i.e., $T = 1$), the improvement in tour length is rather limited when $|S_t| > 100$. By incorporating our proposed sequential sampling strategy, the tour length can be reduced much more rapidly. This is because obtaining high-quality modifications in a single shot is challenging; sequential sampling effectively decomposes this complex task into multiple simpler steps. Interestingly, we observe that certain combinations with more sequential sampling steps and fewer parallel samples dominate those with fewer sequential steps but more parallel samples, in terms of both tour length and inference time. For example, the configuration $(|S_t| = 100, T = 30)$ consistently outperforms $(|S_t| = 300, T = 20)$ across all datasets and for both heuristic tour constructors, achieving shorter tours while requiring less inference time. Notably, although TSP-MDF is trained using a fixed number of sequential steps $(T = 30)$, it remains effective during inference with an arbitrarily number of steps, including both



(a) TSP-500 Nearest Insertion (b) TSP-1000 Nearest Insertion (c) TSP-10000 Nearest Insertion



(d) TSP-500 Farthest Insertion (e) TSP-1000 Farthest Insertion (f) TSP-10000 Farthest Insertion

Figure 3: Ablation study: average tour length reduction on training TSP instances.

Table 6: Ablation study on test instances.

Method	TSP-500	TSP-1000	TSP-10000
Nearest Insertion	20.62	28.95	90.49
TSP-MDF (random) (T=30) + Nearest	20.62/203.69	28.95/416.06	90.49/4325.00
TSP-MDF (basic) (T=30) + Nearest	20.62/86.15	28.95/162.85	90.49/1518.46
TSP-MDF (discrete) (T=30) + Nearest	20.55/20.75	28.92/29.24	90.13/90.13
TSP-MDF (discrete + imitation) (T=30) + Nearest	19.21/19.21	27.53/27.53	89.41/89.41
TSP-MDF (T=30) + Nearest	18.94/18.94	27.09/27.09	88.86/88.86
Farthest Insertion	18.29	25.73	80.69
TSP-MDF (random) (T=30) + Farthest	18.29/203.26	25.73/415.60	80.69/4323.17
TSP-MDF (basic) (T=30) + Farthest	18.29/82.30	25.73/157.53	80.69/1505.33
TSP-MDF (discrete) (T=30) + Farthest	18.01/18.03	25.52/25.52	80.67/80.81
TSP-MDF (discrete + imitation) (T=30) + Farthest	17.55/17.55	24.68/24.68	80.26/80.26
TSP-MDF (T=30) + Farthest	17.26/17.26	24.44/24.44	78.91/78.91

fewer ($T < 30$) and more ($T > 30$) sequential steps, which demonstrates the flexibility of our sequential sampling.

D.3 Ablation Study

Basic design of neural-based instance modifier. To evaluate the performance improvements brought by the key designs of the neural-based instance modifier, we start from a base model, TSP-MDF (basic), which adopts only the basic design described in Section 3.2. Table 6 shows the performance of neural-based instance modifier on test instances, and Figure 3 presents the tour length reduction on the training instances. It is clear to see that although this basic neural-based instance modifier outperforms random modification (i.e., TSP-MDF (random)), it still struggles to identify meaningful modifications, thus fails to help the base tour constructor to find better tours on both training and test instances.

Discretization of coordinate offset. To evaluate the effectiveness of coordinate offset discretization, we introduce a variant model, TSP-MDF (discrete), which extends the basic neural-based instance modifier by discretizing the coordinate offsets as described in Section 3.3. As shown in Table 6 and Figure 3, the proposed discretization significantly improves performance compared to the basic design with continuous offsets. In particular, the neural-based instance modifier begins to identify meaningful modifications that effectively guide the base tour constructor toward better tours on both training and test instances. This improvement can be attributed to the fact that discretizing coordinate offsets reduces the search space to a finite set, while the resulting Categorical distribution enables more flexible yet stable sampling. However, discretizing the coordinate offsets alone is still

insufficient for the neural-based instance modifier to identify high-quality modifications that can substantially enhance the base tour constructor.

Self-imitation learning. To further improve the neural-based instance modifier, we introduce self-imitation learning into its training. To evaluate this design, we create a variant model, TSP-MDF (discrete + imitation), which extends TSP-MDF (discrete) by additionally applying self-imitation learning as described in Section 3.4. As shown in Figure 3, self-imitation learning enables the neural-based instance modifier to quickly discover meaningful modifications that yield positive tour length reductions on training instances, even at the early stage of training within only a few epochs. Thus, the neural-based instance modifier can sample high-quality modifications that significantly enhance the performance of the base tour constructors on test instances, as reported in Table 6.

This improvement arises because, when relying solely on REINFORCE (i.e., Eq. 7), the neural-based instance modifier must explore the search space randomly at cold start and update its parameters based only on these randomly generated modified instances. However, as demonstrated in Section 4.3 (i.e., TSP-MDF (random)), such random modifications can severely distort the original instances and result in tours of very poor quality. As a result, training the instance modifier purely on these random instances is highly inefficient during the cold-start phase, leading to substantial training time being wasted on unpromising regions of the search space.

By leveraging the self-imitation learning strategy, we anchor the exploration process to relatively promising regions of the search space, thereby preventing the instance modifier from sampling destructive random modifications. This substantially improves training efficiency and enables the neural-based instance modifier to rapidly learn how to generate high-quality instance modifications.

Unification of target output space. To evaluate the necessity of unifying the target output space across nodes, we compare TSP-MDF with TSP-MDF (discrete + imitation). The latter can be viewed as a variant of TSP-MDF that directly predicts the distribution of refined coordinate offsets for each node, i.e., without unifying the target output space across nodes. As shown in Figure 3 and Table 6, TSP-MDF achieves higher training efficiency and yields greater tour length reductions on test instances compared to TSP-MDF (discrete + imitation). This performance gap arises because, when directly predicting refined coordinate offsets, the instance modifier must not only learn how to refine the previous offsets but also implicitly infer the magnitude of those previous offsets. As a result, the target output space varies substantially across nodes, increasing the difficulty of training for both the reinforcement learning and self-imitation learning. In contrast, by unifying the target output space and predicting only the distribution of coordinate offset refinement, TSP-MDF simplifies and stabilizes the training. Therefore, unifying the target output space is essential for efficient and effective training of the neural-based instance modifier.

Table 7: Results on TSPLib. Each instance modification iteration of TSP-MDF samples 100 modified instances. "OOM" indicates that the method ran out of GPU memory (24 GB).

Method	TSPLIB($1-500$)		TSPLib($501-1000$)		TSPLib($1001-10000$)	
	Length ↓	Inference Time ↓	Length ↓	Inference Time ↓	Length ↓	Inference Time ↓
DIMES	34422.59	3.16s	2250053.44	2.41s	1336129.23	24.72s
DIMES+AS	**31314.18**	5m44s	**2069587.26**	4m8s	**1260123.03**	21m41s
DeepACO	33611.65	2m24s	2391352.04	1m40s	OOM	OOM
Pointerformer	31370.40	13.30s	2265651.43	12.58s	OOM	OOM
DIFUSCO	31995.12	1m48s	2155408.74	2m16s	1682753.14	22m31s
FastT2T	32757.16	8.36s	2090940.55	5.43s	1577897.58	2m9s
FastT2T+MS+GS	30953.51	1m4s	2025285.39	34.43s	OOM	OOM
BQ-NCO	30488.18	23.61s	**2016163.67**	39.79s	**1407028.70**	91m13s
LEHD	**30404.22**	13.35s	2049309.22	10.18s	1519914.07	13m8s
Nearest Insertion	35896.11	0.01s	2321590.17	0.017s	1365494.55	0.67s
TSP-MDF ($T=1$) + Nearest	34833.57	0.32s	2312856.72	0.21s	1349780.89	8.47s
TSP-MDF ($T=30$) + Nearest	33040.38	10.68s	2198357.65	9.10s	1330555.17	4m27s
Farthest Insertion	32051.45	0.01s	2089399.54	0.014s	1274799.74	0.64s
TSP-MDF ($T=1$) + Farthest	31440.27	0.31s	2078129.38	0.19s	1264057.85	8.96s
TSP-MDF ($T=30$) + Farthest	**30468.03**	10.48s	**1995797.52**	8.73s	**1238924.88**	4m26s

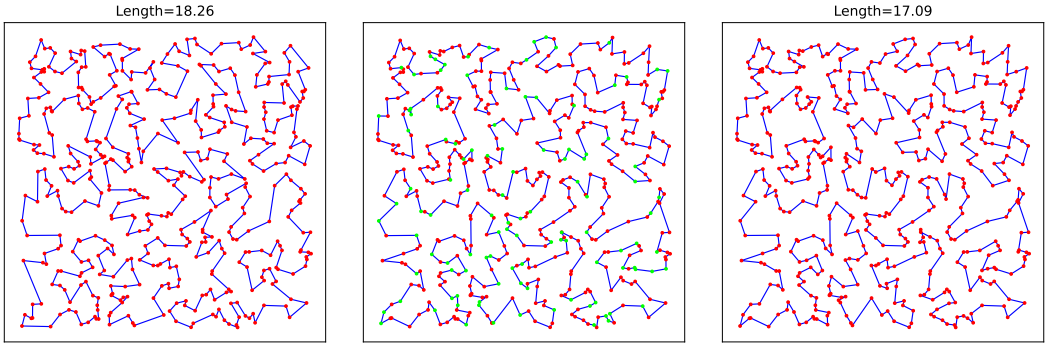
D.4 Generalization Ability to Real-world Benchmarks

To evaluate the generalization ability of TSP-MDF on real-world datasets, we consider the neural-based instance modifier trained solely on random instances (i.e., the training sets of TSP-n datasets) and test on the well-known real-world benchmarks TSPLib (Reinelt, 1991) with various distributions. We use all 87 2-D TSPLib instances with up to 10000 nodes, and group them according to instance size into three subsets: TSPLIB$(1-500)$, TSPLib$(501-1000)$, and TSPLib$(1001-10000)$. Specifically, we evaluate TSP-MDF trained on random instances of 500, 1000 and 10000 nodes for TSPLIB$(1-500)$, TSPLib$(501-1000)$, and TSPLib$(1001-10000)$, respectively.

Table 7 presents the performance results on TSPLib. The results clearly demonstrate that TSP-MDF consistently enhances traditional heuristic tour constructors, even when evaluated on test instances whose distributions and sizes differ from those of the training instances. In particular, while Farthest Insertion originally performs worse than some neural-based heuristic tour constructors, the TSP-MDF-enhanced Farthest Insertion consistently outperforms neural-based heuristic tour constructors across all three TSPLib subsets, while maintaining high computational efficiency. Notably, DIMES+AS is fine-tuned on each individual test instance via Active Search (Hottung et al., 2021), yet it still fails to outperform our TSP-MDF-enhanced Farthest Insertion. Overall, these results demonstrate the strong generalization capability of TSP-MDF on real-world TSP instances.

D.5 Visualization

We visualize how traditional heuristic tour constructors are enhanced by our TSP-MDF framework. Figure 4 presents a representative example on a TSP-500 test instance. Specifically, Figures 4(a) and (d) show the tours produced by the base heuristic tour constructors on the original instances, while Figures 4(b) and (e) illustrate the tours obtained on the modified instances. Figures 4(c) and (f) further visualize the tours from the modified instances after mapping them back to the original instances. By slightly perturbing the node coordinates, TSP-MDF enables traditional heuristic tour constructors to explore substantially different solution trajectories, helping them escape poor local optima and ultimately approximate significantly shorter tours.



(a) Tour approximated by the Farthest Insertion on the original instance.

(b) Tour approximated by the Farthest Insertion on the modified instance.

(c) Tour approximated in (b), mapped back to the original instance.

(d) Tour approximated by the Nearest Insertion on the original instance.

(e) Tour approximated by the Nearest Insertion on the modified instance.

(f) Tour approximated in (e), mapped back to the original instance.

Figure 4: An example of TSP-MDF-enhanced heuristic tour constructors on a TSP-500 test instance. Green nodes in the modified instance indicate nodes whose coordinates have been modified.