

NeuFACO: Neural Focused Ant Colony Optimization for Traveling Salesman Problem

Tran Thanh Dat^{*†}, Pham Anh Khoi^{†1}, Tran Quang Khai^{*1}, Do Duc Dong^{‡2}, and Vu Van Khu^{*2}

^{*}College of Engineering and Computer Science, VinUniversity, Hanoi, Vietnam

[†]School of Information and Communications Technology, Hanoi University of Science and Technology, Hanoi, Vietnam

[‡]University of Engineering and Technology, Vietnam National University, Hanoi, Vietnam

^{*}{dat.tt3,23khai.tq,khu.vv}@vinuni.edu.vn [†]khoi.pa230043@sis.hust.edu.vn [‡]dongdoduc@vnu.edu.vn

Abstract—This study presents Neural Focused Ant Colony Optimization (NeuFACO), a non-autoregressive framework for the Traveling Salesman Problem (TSP) that combines advanced reinforcement learning with enhanced Ant Colony Optimization (ACO). NeuFACO employs Proximal Policy Optimization (PPO) with entropy regularization to train a graph neural network for instance-specific heuristic guidance, which is integrated into an optimized ACO framework featuring candidate lists, restricted tour refinement, and scalable local search. By leveraging amortized inference alongside ACO’s stochastic exploration, NeuFACO efficiently produces high-quality solutions across diverse TSP instances. Code is available at <https://github.com/shoraaa/NeuFACO>

Index Terms—Neural Combinatorial Optimization, Traveling Salesman Problem, Ant Colony Optimization, Proximal Policy Optimization, Graph Neural Networks

I. INTRODUCTION

The Traveling Salesman Problem (TSP) is a classical combinatorial optimization problem (COP) that seeks the shortest Hamiltonian tour visiting each city once and returning to the start. Due to its relevance in logistics, production planning, and circuit design, it serves as a benchmark for evaluating optimization algorithms [1].

Exact approaches like integer linear programming guarantee optimality but scale poorly due to exponential complexity [2]. Heuristic and metaheuristic methods offer efficient near-optimal solutions but often lack problem-specific knowledge, rely on handcrafted heuristics, and risk premature convergence to local optima [3], [4].

Deep learning has inspired data-driven approaches to COPs such as TSP [5]. Supervised methods train neural networks to imitate solvers for fast inference but depend on labeled solutions, limiting generalization. Reinforcement learning (RL) avoids this by directly optimizing solution quality through interaction, yet RL-based methods still suffer from weak local refinement, poor scalability, and high sample complexity [6].

To address these issues, hybrid frameworks combine neural models with metaheuristics. Non-autoregressive (NAR) “learn-to-predict” methods use neural networks to generate heuristics that guide metaheuristics, rather than directly constructing solutions [7], [8]. However, current approaches face high variance

and low sample efficiency due to simplistic training, while refinement often relies on standard Ant Colony Optimization (ACO) [9], which exploits TSP structure inefficiently and limits scalability to larger or heterogeneous instances.

A. Paper overview

We introduce Neural Focused Ant Colony Optimization (NeuFACO), a hybrid framework that integrates deep reinforcement learning with a refined variant of ACO. NeuFACO employs Proximal Policy Optimization (PPO) [10] to train a neural policy, offering greater stability and sample efficiency than prior approaches.

For refinement, NeuFACO adapts Focused ACO (FACO) [11], an advanced extension that performs targeted modifications around a reference solution. Unlike conventional ACO, which rebuilds full tours each iteration, FACO selectively relocates a few nodes, preserving strong substructures while improving weaker regions. This focused search narrows exploration to promising areas, enhancing efficiency.

By combining neural priors with restricted refinement, NeuFACO balances global guidance and local exploitation. This reduces disruption to near-optimal tours, accelerates convergence, and scales effectively to large TSP instances.

Experiments show that NeuFACO outperforms neural and classical baselines on random TSPs and TSPLIB benchmarks, solving problems with up to 1,500 nodes. Results also confirm the advantage of PPO in producing high-quality heuristic priors, establishing NeuFACO as a robust and generalizable framework for neural-augmented combinatorial optimization.

II. RELATED WORK

A. Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a classical NP-hard problem in combinatorial optimization [1]. It can be formally defined on a complete graph $G = (V, E)$, where the vertex set $V = \{v_1, v_2, \dots, v_n\}$ corresponds to the n cities and the edge set E denotes all pairwise connections among them. Each edge $(i, j) \in E$ has an associated non-negative cost $d_{i,j}$, often representing distance or travel time. For simplicity, this paper focuses on the two-dimensional Euclidean TSP as an illustrative example, where each edge $(i, j) \in E$ has a distance feature $d_{i,j} = \|v_i - v_j\|$. The task is to determine a

¹Equal contribution.

²Corresponding authors.

Hamiltonian tour $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ that traverses every city once before returning to the origin, with the goal of minimizing the aggregate travel distance or cost $C(\pi) = \sum_{i=1}^n d_{\pi_i, \pi_{i+1}}$, where π is a permutation of V representing the visiting order of the cities and $\pi_{n+1} = \pi_1$.

B. Classical Methods

Classical approaches to the TSP fall into two categories: exact and heuristic-based [3]. Exact methods, such as branch-and-bound [12] and Integer Linear Programming [2], guarantee optimality but suffer factorial time complexity, making them impractical for large instances.

To improve scalability, heuristic and metaheuristic algorithms provide near-optimal solutions in reasonable time [13]. Local search methods like 2-opt, 3-opt, and Lin–Kernighan [14], [15] refine tours by replacing subpaths to reduce cost, while metaheuristics such as Ant Colony Optimization [16] and Genetic Algorithms [17] stochastically explore the solution space by simulating natural processes.

C. Ant Colony Optimization

Ant Colony Optimization is a bio-inspired metaheuristic that emulates the collective foraging behavior of natural ant colonies [16]. In the context of the TSP, artificial ants construct solutions by probabilistically choosing the next city according to two key factors: the pheromone trail τ_{ij} , which encodes the learned desirability of selecting edge (i, j) , and the heuristic desirability $\eta_{ij} = 1/d_{ij}$, which favors shorter edges. The probability p_{ij}^k that ant k moves from city i to city j is typically defined as

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{l \in \mathcal{N}_i^k} \tau_{il}^\alpha \eta_{il}^\beta} & \text{if } j \in \mathcal{N}_i^k, \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where \mathcal{N}_i^k denotes the set of cities not yet visited by ant k , and α and β determine the relative weighting of pheromone intensity and heuristic desirability. Once all ants have completed their tours, pheromone trails are updated both to reinforce high-quality paths and to allow for evaporation, which encourages exploration:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \sum_{k=1}^m \Delta\tau_{ij}^k, \quad (2)$$

where $\rho \in (0, 1]$ is the evaporation rate, and $\Delta\tau_{ij}^k$ represents the pheromone deposited by ant k proportional to the quality of its solution. Over time, ACO converges toward promising regions of the solution space, but it may encounter slow convergence or premature stagnation.

D. Neural Combinatorial Optimization

Neural Combinatorial Optimization (NCO) employs deep learning to derive heuristics for NP-hard problems such as the Traveling Salesman Problem (TSP) [5]. Unlike handcrafted approaches, NCO learns directly from data via supervised imitation of high-quality solutions or reinforcement learning on reward signals such as negative tour length. Architectures

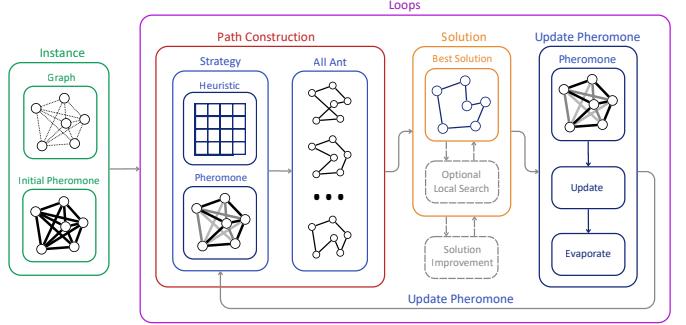


Fig. 1: Original Ant Colony Optimization.

explored include pointer networks [18], GNNs [19], and more recently Transformers [20] and diffusion models [21].

NCO methods are generally categorized as constructive, incrementally generating solutions, or improvement-based, refining initial ones with techniques like 2-opt [6]. Non-autoregressive (NAR) variants follow a hybrid paradigm: neural encoders extract heuristic signals that metaheuristics exploit for solution construction, contrasting with end-to-end autoregressive models [8].

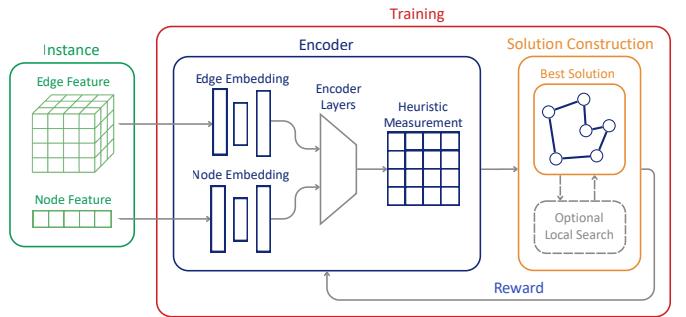


Fig. 2: NAR training procedure.

In Ant Colony Optimization (ACO), classical heuristics are handcrafted and domain-dependent, limiting scalability and adaptability. NAR methods overcome these issues by learning heuristics from data and embedding them into the ACO pipeline. Early implementations relied on REINFORCE [22], but its high variance and slow convergence hindered performance, motivating more stable reinforcement learning techniques.

III. TRAINING METHODOLOGY

A. Markov Decision Process (MDP) formulation

We model TSP solution construction as an MDP. At step t , the state $s_t = (\mathcal{X}, S_t, i_t)$ includes the instance, visited set, and current node. The action $a_t \in \mathcal{N}_{i_t} \setminus S_t$ is chosen from $\pi_\theta(\cdot | i_t)$ with feasibility masking. Transitions append a_t to the tour, and the episode ends after all nodes are visited, yielding reward $R = -C(\pi; \mathcal{X})$.

B. Policy Parameterization

A graph neural network f_θ , parameterized by θ , maps graph data \mathcal{X} to:

- A heuristic matrix $H_\theta \in \mathbb{R}^{n \times n}$, encoding learned priors over edge transitions.
- A value estimate $V_\theta(\mathcal{X}) \in \mathbb{R}$, predicting the expected return of the instance.

We train H_θ as an instance-conditioned heuristic prior using stable on-policy updates. Training is strictly *pheromone-free*: tours are sampled directly from the stochastic policy induced by H_θ , enabling a standard PPO formulation without non-stationary sampler dynamics.

C. Rollouts and Rewards

For each instance \mathcal{X} , we sample a batch of M tours $\{\pi^{(m)}\}_{m=1}^M$ by chaining the policy over the shrinking feasible set:

$$\pi_{t+1}^{(m)} \sim \pi_\theta(\cdot | \pi_t^{(m)}), \quad t = 1, \dots, n-1.$$

Each tour receives a scalar terminal reward equal to the negative tour length:

$$R^{(m)} = -C(\pi^{(m)}; \mathcal{X}), \quad m = 1, \dots, M. \quad (3)$$

We use a per-instance, per-batch baseline $\bar{R} = \frac{1}{M} \sum_{m=1}^M R^{(m)}$ for variance reduction and to normalize reward and calculated advantage:

$$\tilde{R}^{(m)} = R^{(m)} - \bar{R}, \quad A^{(m)} = \tilde{R}^{(m)} - V_\theta(X). \quad (4)$$

D. PPO Objective

For each sampled path $\pi^{(m)}$, let $\log p_\theta(\pi^{(m)})$ denote the log probability of sampling under the current policy, and $\log p_{\theta_{\text{old}}}(\pi^{(m)})$ under the previous policy. Then the probability ratio is:

$$r^{(m)}(\theta) = \frac{p_\theta(\pi^{(m)})}{p_{\theta_{\text{old}}}(\pi^{(m)})}. \quad (5)$$

And the clipped PPO surrogate objective is:

$$\begin{aligned} \mathcal{L}_{\text{policy}}(\theta) = & -\frac{1}{M} \sum_{m=1}^M \min \left(r^{(m)} A^{(m)}, \right. \\ & \left. \text{clip}(r^{(m)}, 1-\varepsilon, 1+\varepsilon) A^{(m)} \right). \end{aligned} \quad (6)$$

The value losses calculated using mean squared error (MSE):

$$\mathcal{L}_{\text{value}}(\theta) = \frac{1}{M} \sum_{m=1}^M \left(V_\theta(X) - \tilde{R}^{(m)} \right)^2 \quad (7)$$

To promote exploration, entropy is computed on the normalized heuristic distribution: $\tilde{H}_{ij} = H_{ij}/(\sum_k H_{ik})$ and

$$\mathcal{H}(\tilde{H}) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^n \tilde{H}_{ij} \log(\tilde{H}_{ij} + \varepsilon). \quad (8)$$

Then we define the entropy loss as:

$$\mathcal{L}_{\text{entropy}} = -\beta \mathcal{H}(\tilde{H}), \quad (9)$$

and the final loss to minimize is $\mathcal{L} = \mathcal{L}_{\text{policy}} + \mathcal{L}_{\text{value}} + \mathcal{L}_{\text{entropy}}$.

IV. SOLUTION SAMPLING

When inference, the ACO constructs a probability distribution over transitions: $p_{ij} \propto \tau_{ij}^\alpha \cdot H_{ij}^\beta$, where τ_{ij} is the pheromone and H_{ij} is the learned heuristic, α and β are weighting parameters.

Our method incorporates multiple state-of-the-art enhancements aimed at improving both the computational efficiency and solution quality of ACO algorithms. These improvements address three key aspects of solution construction: efficient node selection, focused modifications to high-quality tours, and scalable local search. Fig. 3 demonstrates the final algorithm for solution sampling.

A. Incorporating the Min-Max Ant System

To enhance solution quality and stabilize the search process, NeuFACO adopts the Min-Max Ant System (MMAS) [23] for pheromone updates. MMAS modifies the classical Ant System in two key aspects: (1) pheromone trails are restricted to the interval $[\tau_{\min}, \tau_{\max}]$ to prevent premature convergence, and (2) updates rely solely on the best-so-far or iteration-best solution, rather than aggregating contributions from all ants.

In our implementation, pheromone updates follow the standard MMAS rule:

$$\tau_{ij} \leftarrow \begin{cases} (1-\rho)\tau_{ij} + \Delta\tau_{ij}^{\text{best}} & \text{if edge } (i,j) \text{ chosen,} \\ (1-\rho)\tau_{ij} & \text{otherwise,} \end{cases} \quad (10)$$

$$\tau_{ij} \leftarrow \text{clip}(\tau_{ij}, \tau_{\min}, \tau_{\max}). \quad (11)$$

This mechanism is particularly critical when combined with neural-guided heuristics, as it prevents the learned model from prematurely dominating the search dynamics.

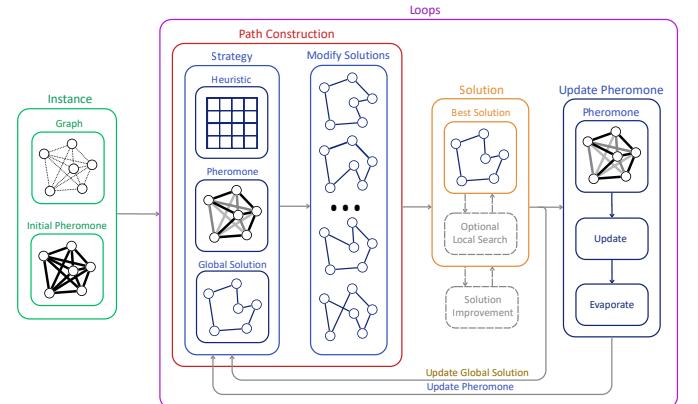


Fig. 3: State-of-the-art Ant Colony Optimization.

B. Node Selection via Candidate and Backup Lists

Candidate lists are widely used in ACO to restrict node selection to the k -nearest neighbors, thereby reducing complexity. Formally, for current node i , the candidate set is

$$\mathcal{C}_i = \{u_{i,j} \mid j \leq k, \text{visited}(u_{i,j}) = 0\}. \quad (12)$$

When $\mathcal{C}_i = \emptyset$, classical ACO scans all unvisited nodes, an $O(n)$ operation. To avoid this, we adopt the backup strategy

of [24]: each node maintains a precomputed backup list BKP_i , from which the first unvisited neighbor is selected. Only if both C_i and BKP_i are empty do we fall back to

$$J = \arg \min_{u: \text{visited}(u)=0} d(i, u). \quad (13)$$

This hierarchical rule preserves stochastic exploration early on while substantially reducing overhead in the later stages of tour construction.

C. Node Relocation to maintain path structure

We adopt the focused modification strategy introduced in [11], which aims to explore the solution space by making minimal impactful changes to existing high-quality tours. At the start of construction, each ant stochastically copies a complete tour from either the global-best (with probability p_g) or the iteration-best candidate and subsequently modifies it by introducing new nodes based on the standard transition rule.

From the selected edge (u, v) and the original tour π , a relocate move produces a new tour π' by removing v from its current position and reinserting it immediately after u . Let $p = \text{pred}(v)$ be the predecessor of v in π , $s = \text{succ}(v)$ be the successor of v , and s_u be the successor of u . Then the cost change from performing such a move is

$$\Delta C = -d_{p,v} - d_{v,s} - d_{u,s_u} + d_{p,u} + d_{u,v} + d_{v,s_u}. \quad (14)$$

To maintain structural similarity to the reference tour, modification is limited: once a predefined minimum number of new edges (MNE) is introduced (we set MNE = 8), the ant stops altering the route and continues following the remaining portion of the original tour. This strategy restricts the number of relocated nodes, thereby concentrating the search effort on promising subregions of the solution space while avoiding the computational burden of constructing tours entirely from scratch. Fig. 4 demonstrates an example of a node relocation operation.

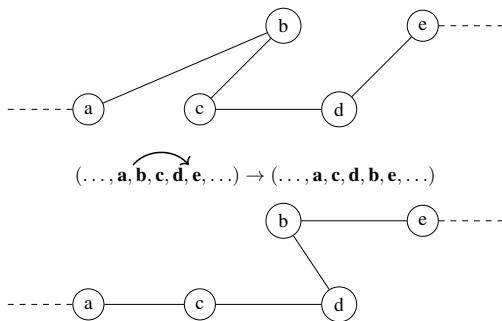


Fig. 4: Example of a node relocation procedure for edge (a, b) .

D. Scalable Local Search

Local search, particularly the 2-opt heuristic, is highly effective for improving ACO-generated solutions but is often applied sparingly due to its computational cost.

To enable frequent refinement, our algorithm incorporates two optimizations from [11]. First, it tracks modified edges in each constructed tour and applies 2-opt only to those differing from the previously optimized tour. Second, it restricts 2-opt to candidate edges, thereby reducing the search space.

With a constant-size candidate list and a limited set of modified edges, this optimized 2-opt can be applied to every solution efficiently, fully exploiting high-quality tours without prohibitive overhead.

V. EXPERIMENTS

This section presents experimental results to validate the effectiveness of our algorithm. We compare it against other ACO-based methods as well as state-of-the-art neural approaches for solving the TSP. Our source code is built upon codebases of previous ACO-based methods: GFACS (AISTATS 2025) [25], which itself builds upon DeepACO (NeurIPS 2023) [7].

A. Setup

1) Training: We evaluate NeuFACO against DeepACO and GFACS. Training uses 20 PPO steps per epoch, batch size 20, learning rate lr = 0.001, clip ratio $\epsilon = 0.2$, and entropy coefficient 0.01. Experiments run on an AMD RX6800 GPU and Intel i5-13400 CPU (2.50 GHz). Fig. 5 shows test objective cost over training steps, illustrating convergence across instance sizes.

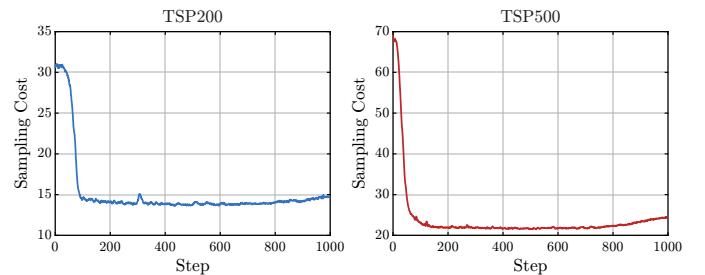


Fig. 5: Test objective cost over training step.

2) Benchmark: We evaluate NeuFACO on randomized TSP instances of size 200, 500, and 1000 (128 each) [7], and on TSPLib [26] with up to 1500 nodes. Results are averaged over $R = 10$ runs for TSPLib and over 128 instances for randomized TSP; baseline results are taken from prior papers. Errors are computed as $E = (\text{cost} - \text{optimal})/\text{optimal} \times 100\%$, with optimal values obtained using Concorde [27].

Our evaluation consists of two parts: (1) controlled comparison with NAR baselines under identical hyperparameters and hardware, reporting runtime and quality; and (2) literature comparison, compiling solution quality and reported runtimes. As experimental setups vary, runtimes are indicative, while solution quality is emphasized as the primary metric.

B. Comparison with Other NAR Solvers

We compare NeuFACO to two non-autoregressive baselines, DeepACO and GFACS, both of which outperform classical ACO without neural guidance [7], [25]. For all NAR models

| Method | TSPLib100-299 | | TSPLib300-699 | | TSPLib700-1499 | | Method | TSP200 | | TSP500 | | TSP1000 | |
|---------|---------------|--------------|---------------|--------------|----------------|--------------|---------|-------------|--------------|-------------|--------------|-------------|--------------|
| | Gap(%) | Time | Gap(%) | Time | Gap(%) | Time | | Gap(%) | Time | Gap(%) | Time | Gap(%) | Time |
| DeepACO | 1.50 | 13.70s | 3.43 | 143.92s | 4.86 | 1047.59s | DeepACO | 1.89 | 16.86s | 3.03 | 119.58s | 4.32 | 659.43s |
| GFACS | 1.44 | 13.75s | 2.46 | 150.49s | 3.99 | 1141.24s | GFACS | 1.80 | 17.36s | 1.54 | 136.14s | 2.48 | 730.56s |
| NeuFACO | 1.39 | 0.22s | 2.06 | 0.48s | 2.98 | 1.03s | NeuFACO | 1.66 | 0.28s | 1.50 | 0.74s | 2.00 | 1.98s |

Fig. 6: Comparison of DeepACO, GFACS and NeuFACO priors. All models are retrained with their recommended parameters, using the same number of epochs and evaluation settings.

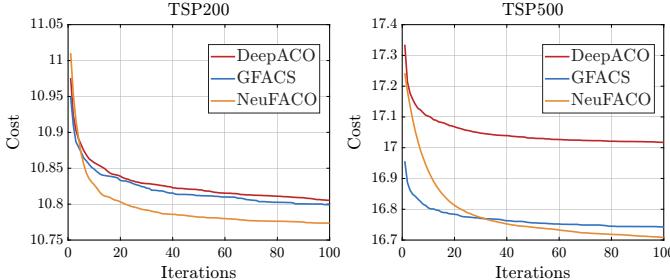


Fig. 7: Objective cost over ACO iterations between different priors.

we fix the number of ants $M = 100$, run $I = 100$ ACO iterations, $\alpha = 1$, $\beta = 1$, and an evaporation rate of $\rho = 0.1$. Unlike DeepACO and GFACS, which employ neural-guided perturbation, NeuFACO applies scalable local search at every iteration, providing effective refinement without additional perturbation mechanisms. NeuFACO-specific hyperparameters are: global path probability $p_g = 0.01$, minimum new edges per iteration MNE = 8, candidate list size $k = 20$, backup list size BKP = 64,

$$\tau_{\max} = \frac{1}{(1 - \rho) g_b}, \quad \tau_{\min} = \min \left(\tau_{\max}, \frac{\tau_{\max} (1 - p_{\text{best}}^{1/k})}{(k - 1) p_{\text{best}}^{1/k}} \right),$$

with g_b denoting the global-best objective and $p_{\text{best}} = 0.1$ as suggested in [11].

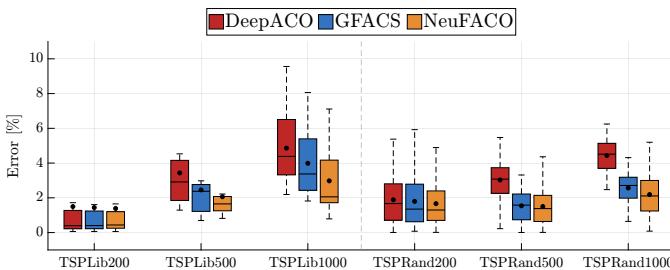


Fig. 8: Minimum, maximum and average errors comparison.

Fig. 6 shows that NeuFACO achieves the highest solution quality among DeepACO, GFACS, and itself across nearly all datasets. Under identical settings, NeuFACO attains up to 60× lower wall-clock time than prior NAR methods, thanks to a refined sampler that exploits neural heuristics with minimal quality loss. PPO with entropy regularization further mitigates policy degradation, ensuring solution quality comparable to

Results for other methods are taken from [25], [32], [33]. We reported the averaged objective value over 128 instances of TSP, originally from [39].

| Method | TSP500 | | TSP1000 | | | |
|---------------------------|--------|--------|------------------|-------|----------------|------------------|
| | Obj. | Gap(%) | Time | Obj. | Gap(%) | Time |
| Concorde | 16.55 | — | 10.7s | 23.12 | — | 108s |
| AM | 21.46 | 29.07 | 0.6s | 33.55 | 45.10 | 1.5s |
| POMO | 20.57 | 24.40 | 0.6s | 32.90 | 42.30 | 4.1s |
| DIMES | 17.01 | 2.78 | 11s | 24.45 | 5.75 | 32s |
| SO | 16.94 | 2.40 | 15s | 23.77 | 2.80 | 26s |
| Pointerformer | 17.14 | 3.56 | 14s | 24.80 | 7.90 | 40s |
| H-TSP (AAAI'23) | 17.55 | 6.22 | 0.18s | 24.65 | 6.62 | 0.33s |
| GLOP (AAAI'24) | 16.91 | 1.99 | 1.5m | 23.84 | 3.11 | 3.0m |
| INViT-3V+Greedy (ICML'24) | 16.78 | 1.56 | 5.48s | 24.66 | 6.66 | 9.0s |
| LEHD+Greedy (NeurIPS'23) | 16.78 | 1.56 | 0.13s | 23.84 | 3.11 | 0.8s |
| BQ+Greedy (NeurIPS'23) | 16.72 | 1.18 | 0.36s | 23.65 | 2.30 | 0.9s |
| SIGD+Greedy (NeurIPS'23) | 16.71 | 1.17 | 0.23s | 23.57 | 1.96 | 1.2s |
| SIT+Greedy (NeurIPS'25) | 16.7 | 1.08 | 14.95s | 23.57 | 1.95 | 0.2s |
| DeepACO (NeurIPS'23) | 16.84 | 1.77 | 15s | 23.78 | 2.87 | 1.1m |
| GFACS (AISTATS'25) | 16.80 | 1.56 | 15s | 23.72 | 2.63 | 1.1m |
| NeuFACO | 16.43 | 1.33 | 0.91s +14.95s | 23.34 | 1.16 +0.03s | 0.98s +40.03s |

Fig. 9: Broader literature comparison on TSP500 and TSP1000.

, or exceeding existing models. Gains are most pronounced on large instances, such as TSPLib graphs with up to 1500 nodes, highlighting NeuFACO’s efficiency on large-scale TSPs.

Fig. 7 confirms faster convergence of average objective costs, while Fig. 8 shows consistently lower variance and superior performance across instance sizes.

C. Comparison with other RL solvers

In this section, we compare NeuFACO with problem-specific RL approaches for the TSP, including established baselines (AM [20], POMO [28], Pointerformer [29]), heatmap-based DIMES [30], improvement-based SO [31], and more recent methods such as H-TSP [32], GLOP [33], INViT [34], LEHD [35], BQ [36], SIGD [37], and SIT [38]. NeuFACO is run with $M = 256$ ants and $I = 1000$ iterations to leverage its efficient sampling, and reported times include both heuristic inference and solution sampling.

Results demonstrate that NeuFACO consistently achieves superior or highly competitive performance across all baselines. It surpasses every established methods and remains competitive with recently proposed approaches, often delivering better solution quality at the cost of slightly longer runtimes. The

experiments also highlight a key bottleneck of NAR methods, including ours: solution sampling remains CPU-bound, leading to higher runtime despite comparable amortized inference. Nevertheless, NeuFACO marks a significant advance for both NAR and RL-based approaches to the TSP.

VI. CONCLUSION

In conclusion, NeuFACO integrates deep reinforcement learning with a refined ACO framework to address limitations of non-autoregressive models for the TSP. By combining PPO-based policy learning with targeted refinement around high-quality solutions, it achieves a strong balance between global guidance and local exploitation. This synergy preserves solution structure, accelerates convergence, and scales effectively to large instances. Experiments show that NeuFACO consistently achieves superior or highly competitive performance compared to a wide range of neural baselines on both randomized and benchmark datasets. While runtimes are longer due to CPU-bound sampling, the method delivers higher solution quality, underscoring the effectiveness of PPO-guided priors in enhancing ACO and establishing NeuFACO as a robust and generalizable framework for neural-augmented combinatorial optimization.

REFERENCES

- [1] R. Matai, S. P. Singh, and M. L. Mittal, *Traveling salesman problem: an overview of applications, formulations, and solution approaches*. InTech Rijeka, 2010, vol. 1, no. 1, pp. 1–25.
- [2] H. Crowder and M. W. Padberg, “Solving large-scale symmetric travelling salesman problems to optimality,” *Management Science*, vol. 26, no. 5, pp. 495–509, 1980.
- [3] S. Lin, “Computer solutions of the traveling salesman problem,” *Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.
- [4] X.-S. Yang, *Nature-inspired optimization algorithms*. Academic Press, 2020.
- [5] Y. Bengio, A. Lodi, and A. Prouvost, “Machine learning for combinatorial optimization: a methodological tour d’horizon,” pp. 405–421, 2021.
- [6] A. I. Garmendia, J. Ceberio, and A. Mendiburu, “Neural combinatorial optimization: a new player in the field,” 2022.
- [7] H. Ye, J. Wang, Z. Cao, H. Liang, and Y. Li, “Deepaco: Neural-enhanced ant systems for combinatorial optimization,” pp. 43 706–43 728, 2023.
- [8] F. Berto, C. Hua, J. Park, L. Luttmann, Y. Ma, F. Bu, J. Wang, H. Ye, M. Kim, S. Choi *et al.*, “Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark,” in *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, ser. KDD ’25, 2025, pp. 5278–5289.
- [9] T. Stützle, M. Dorigo *et al.*, “Aco algorithms for the traveling salesman problem,” *Evolutionary algorithms in engineering and computer science*, vol. 4, pp. 163–183, 1999.
- [10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms.” *CoRR*, vol. abs/1707.06347, 2017.
- [11] R. Skinderowicz, “Focused aco with node relocation procedure for solving large tsp instances,” *Procedia Computer Science*, vol. 225, pp. 2992–3000, 2023, 27th International Conference on Knowledge Based and Intelligent Information and Engineering Systems (KES 2023).
- [12] M. Padberg and G. Rinaldi, “A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems,” *SIAM review*, vol. 33, no. 1, pp. 60–100, 1991.
- [13] T. Dokeroğlu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, “A survey on new generation metaheuristic algorithms,” *Computers & Industrial Engineering*, vol. 137, p. 106040, 2019.
- [14] G. A. Croes, “A method for solving traveling-salesman problems,” *Operations Research*, vol. 6, pp. 791–812, 1958.
- [15] K. Helsgaun, “An effective implementation of the lin-kernighan traveling salesman heuristic,” *European journal of operational research*, vol. 126, no. 1, pp. 106–130, 2000.
- [16] M. Dorigo and T. Stützle, *Ant colony optimization: overview and recent advances*. Springer, 2019.
- [17] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.
- [18] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer networks,” in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015.
- [19] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, “Graph neural networks: A review of methods and applications,” pp. 57–81, 2020.
- [20] W. Kool, H. van Hoof, and M. Welling, “Attention, learn to solve routing problems!” 2018.
- [21] Z. Sun and Y. Yang, “Difusco: Graph-based diffusion solvers for combinatorial optimization,” pp. 3706–3731, 2023.
- [22] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine learning*, vol. 8, no. 3–4, p. 229–256, May 1992.
- [23] T. Stützle and H. H. Hoos, “Max–min ant system,” *Future generation computer systems*, vol. 16, no. 8, pp. 889–914, 2000.
- [24] P. A. Martínez and J. M. García, “Acotsp-mf: A memory-friendly and highly scalable acotsp approach,” *Engineering Applications of Artificial Intelligence*, vol. 99, p. 104131, 2021.
- [25] M. Kim, S. Choi, H. Kim, J. Son, J. Park, and Y. Bengio, “Ant colony sampling with gflownets for combinatorial optimization,” in *Proceedings of The 28th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. Li, S. Mandt, S. Agrawal, and E. Khan, Eds., vol. 258. PMLR, 03–05 May 2025, pp. 469–477.
- [26] G. Reinelt, “Tsplib - a traveling salesman problem library,” *INFORMS J. Comput.*, vol. 3, pp. 376–384, 1991.
- [27] D. Applegate, R. Bixby, V. Chvátal, and W. J. Cook, “Concorde tsp solver,” 2006. [Online]. Available: <https://www.math.uwaterloo.ca/tsp/concorde.html>
- [28] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, “Pomo: Policy optimization with multiple optima for reinforcement learning,” pp. 21 188–21 198, 2020.
- [29] Y. Jin, Y. Ding, X. Pan, K. He, L. Zhao, T. Qin, L. Song, and J. Bian, “Pointformer: Deep reinforced multi-pointer transformer for the traveling salesman problem,” vol. 37, no. 7, Jun. 2023, pp. 8132–8140.
- [30] R. Qiu, Z. Sun, and Y. Yang, “Dimes: A differentiable meta solver for combinatorial optimization problems,” pp. 25 531–25 546, 2022.
- [31] H. Cheng, H. Zheng, Y. Cong, W. Jiang, and S. Pu, “Select and optimize: Learning to solve large-scale tsp instances,” ser. Proceedings of Machine Learning Research, F. Ruiz, J. Dy, and J.-W. van de Meent, Eds., vol. 206. PMLR, 25–27 Apr 2023, pp. 1219–1231.
- [32] X. Pan, Y. Jin, Y. Ding, M. Feng, L. Zhao, L. Song, and J. Bian, “H-tsp: hierarchically solving the large-scale traveling salesman problem,” 2023.
- [33] H. Ye, J. Wang, H. Liang, Z. Cao, Y. Li, and F. Li, “Glop: Learning global partition and local construction for solving large-scale routing problems in real-time,” pp. 20 284–20 292, 2024.
- [34] H. Fang, Z. Song, P. Weng, and Y. Ban, “Invit: A generalizable routing problem solver with invariant nested view transformer,” 2024.
- [35] F. Luo, X. Lin, F. Liu, Q. Zhang, and Z. Wang, “Neural combinatorial optimization with heavy decoder: toward large scale generalization,” in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS ’23, 2023.
- [36] D. Drakulic, S. Michel, F. Mai, A. Sors, and J.-M. Andreoli, “Bq-nco: bisimulation quotienting for efficient neural combinatorial optimization,” in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS ’23, 2023.
- [37] J. Pirnay and D. G. Grimm, “Self-improvement for neural combinatorial optimization: Sample without replacement, but improvement,” *Transactions on Machine Learning Research*, 2024, featured Certification.
- [38] F. Luo, X. Lin, Y. Wu, Z. Wang, T. Xialiang, M. Yuan, and Q. Zhang, “Boosting neural combinatorial optimization for large-scale vehicle routing problems,” in *The Thirteenth International Conference on Learning Representations*, 2025.
- [39] M. Li, S. Tu, and L. Xu, “Generalizing graph network models for the traveling salesman problem with lin-kernighan-helsgaun heuristics,” in *Neural Information Processing: 30th International Conference, ICONIP 2023, Changsha, China, November 20–23, 2023, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2023, p. 528–539.