# GLOP: Learning Global Partition and Local Construction for Solving Large-scale Routing Problems in Real-time

**Haoran Ye**[1], **Jiarui Wang**[1], **Helan Liang**[1*], **Zhiguang Cao**[2], **Yong Li**[3], **Fanzhang Li**[1]

[1]Soochow University, China
[2]Singapore Management University, Singapore
[3]Tsinghua University, China
{hrye,jrwangfurffico}@stu.suda.edu.cn, {hlliang,lfzh}@suda.edu.cn
zgcao@smu.edu.sg, liyong07@tsinghua.edu.cn

**Abstract**

The recent end-to-end neural solvers have shown promise for small-scale routing problems but suffered from limited real-time scaling-up performance. This paper proposes GLOP (**G**lobal and **L**ocal **O**ptimization **P**olicies), a unified hierarchical framework that efficiently scales toward large-scale routing problems. GLOP partitions large routing problems into Travelling Salesman Problems (TSPs) and TSPs into Shortest Hamiltonian Path Problems. For the first time, we hybridize non-autoregressive neural heuristics for coarse-grained problem partitions and autoregressive neural heuristics for fine-grained route constructions, leveraging the scalability of the former and the meticulousness of the latter. Experimental results show that GLOP achieves competitive and state-of-the-art real-time performance on large-scale routing problems, including TSP, ATSP, CVRP, and PCTSP. [1]

## 1 Introduction

Routing problems pervade logistics, supply chain, transportation, robotic systems, etc. Modern industries have witnessed ever-increasing demands for the massive and expeditious routing of goods, services, and people. The traditional solvers based on mathematical programming or iterative heuristics struggle to keep pace with such growing complexity and real-time requirements.

Recent advances in Neural Combinatorial Optimization (NCO) (Bengio, Lodi, and Prouvost 2021) seek end-to-end solutions for routing problems, where neural solvers are exploited and empowered by massive training while enjoying potentially efficient inference. However, most existing NCO methods still struggle with real-time scaling-up performance; they are unable to solve routing problems involving thousands or tens of thousands of nodes in seconds, falling short of the need of modern industries (Hou et al. 2023).

In answer to that, this work proposes GLOP (**G**lobal and **L**ocal **O**ptimization **P**olicies) which partitions a large routing problem into sub-Travelling Salesman Problems (TSPs) and further partitions potentially large (sub-)TSPs into small Shortest Hamiltonian Path Problems (SHPPs). GLOP hybridizes non-autoregressive (NAR) global partition and autoregressive (AR) local construction policies, where the global policy learns the first partition and the local policy learns to solve SHPPs. We intend to integrate the strengths while circumventing the drawbacks of NAR and AR paradigms (detailed in Section 2 and Appendix B). Particularly, on the one hand, partitioning nodes into subsets (each corresponding to a TSP) well suits the nature of NAR heuristics because it is a large-scale but coarse-grained task agnostic of within-subset node ordering. On the other hand, solving SHPPs could be efficiently handled with the AR heuristic because it is a small-scale but fine-grained task.

The solution pipeline of GLOP is applicable to variations of routing problems, such as those tackled in (Li et al. 2021a; Zhang et al. 2022a, 2021; Alesiani, Ermis, and Gkiotsalitis 2022; Miranda-Bront et al. 2017; Li et al. 2021b). We evaluate GLOP on canonical TSP, Asymmetric TSP (ATSP), Capacitated Vehicle Routing Problem (CVRP), and Prize Collecting TSP (PCTSP). GLOP for (A)TSP, as opposed to most methods that require scale-specific and distribution-specific training, can perform consistently and competitively across scales, across distributions, and on real-world benchmarks with the same set of local policies. Notably, it is the first neural solver to effectively scale to TSP100K, obtaining a 5.1% optimality gap and a $174\times$ speedup compared with 1-run 1-trial LKH-3. GLOP for CVRP clearly outperforms prior state-of-the-art (SOTA) real-time solvers (Hou et al. 2023) while using $10\times$ less execution time. On PCTSP, GLOP surpasses both recent neural solvers and conventional solvers.

Accordingly, we summarize our contributions as follows:

- We propose GLOP, a versatile framework that extends existing neural solvers to large-scale problems. To our knowledge, it makes the first effective attempt at hybridizing NAR and AR end-to-end NCO paradigms.
- We propose to learn global partition heatmaps for decomposing large-scale routing problems, leveraging NAR heatmap learning in a novel way.
- We propose a one-size-fits-all real-time (A)TSP solver that learns small SHPP solution construction for arbitrarily large (A)TSP. We dispense with learning upper-level TSP policies suggested in (Kim, Park, and Kim 2021; Pan et al. 2023) while achieving better performance.
- On (A)TSP, GLOP delivers competitive scaling-up and

*Corresponding author.

[1]Our code is available: https://github.com/henry-yeh/GLOP.

cross-distribution performance and is the first neural solver to scale to TSP100K effectively. On CVRP and PCTSP, GLOP achieves SOTA real-time performance.

# 2 Background and related work

## 2.1 Neural Combinatorial Optimization (NCO)

Recent advances in NCO show promise for solving combinatorial optimization problems in an end-to-end manner (Berto et al. 2023; Bogyrbayeva et al. 2022; Mazyavkina et al. 2021). The end-to-end neural routing solvers can be categorized into two paradigms, i.e., AR solution construction and NAR heatmap generation coupled with subsequent decoding. We defer the related work and discussions to Appendix C.

## 2.2 Divide and conquer for VRP

The idea of "divide and conquer" has long been applied to VRP variants in traditional (meta) heuristics (Zhang et al. 2021; Alesiani, Ermis, and Gkiotsalitis 2022; Xiao et al. 2019; Taillard and Helsgaun 2019). Recently, such an idea has been introduced in neural routing solvers. Li, Yan, and Wu (2021) propose learning to delegate (L2D) the improvement of subtours to LKH-3 (Helsgaun 2017). Zong et al. (2022a) introduce Rewriting-by-Generating (RBG) framework that involves repeated learning-based merging and rule-based decomposition. However, both methods rely on iterative refinement, therefore holding back the real-time performance. More related to GLOP, Hou et al. (2023) present a Two-stage Divide Method (TAM), the prior SOTA real-time neural solver for large-scale CVRP, where a dividing model learns to partition CVRP into sub-TSPs autoregressively, and the sub-TSPs are then solved by sub-solvers such as Attention Model (AM) (Kool, van Hoof, and Welling 2019) or LKH-3. Unlike other prior works, both TAM and GLOP target large-scale CVRP under real-time settings. By comparison, GLOP outperforms TAM on CVRP by leveraging more effective global representations and better neural sub-TSP solvers, and can also handle routing problems that TAM does not address.

## 2.3 Local construction for TSP

Learning local subtour reconstruction for TSP is initially introduced by Kim, Park, and Kim (2021) in Learning Collaborative Policy (LCP). LCP generates diversified initial solutions (seeds) with neural models (seeders), then repeatedly decomposes and reconstructs them. However, LCP, limited mainly by the design of seeders, can hardly scale up to TSP with hundreds of nodes. More recently, Pan et al. (2023) propose H-TSP, a hierarchical TSP solver interleaving forming open-loop TSP (a.k.a., SHPP) with upper-level policies and conquering it. By comparison, GLOP dispenses with learning any upper-level TSP policy but is able to outperform H-TSP. Another concurrent work, namely select-and-optimize (SO) (Cheng et al. 2023), utilizes a TSP solution pipeline similar to GLOP. But SO heavily relies on sophisticated heuristics specific to TSP, resulting in prolonged computational time. By comparison, GLOP achieves competitive solutions while being hundreds of times more efficient.

# 3 Methodology overview

Schematically illustrated in Figure 1, GLOP learns local policies to solve (sub-)TSP and global policies to partition large routing problems into sub-TSPs. Our (sub-)TSP solver generates initial TSP tours using Random Insertion, divides the complete tours into independent subtours, and learns to reconstruct them for improvements. Our general routing solver first performs clustering or subsetting guided by the learned partition heatmap, generating sub-TSP(s) which are then solved by our sub-TSP solver. We elaborate on our local policy and global policy in Section 4 and Section 5, respectively, and provide more details in Appendix A.

# 4 (Sub-)TSP solver

## 4.1 Inference pipeline

GLOP learns local policies to improve a TSP solution by decomposing and reconstructing it.

**Initialization** GLOP generates an initial TSP tour with Random Insertion (RI), a simple and generic heuristic. RI greedily picks the insertion place that minimizes the insertion cost for each node.

Then, GLOP performs improvements on the initial tour. Following Kim, Park, and Kim (2021), we refer to a round of improvement as a "revision"; we refer to a local policy parameterized by an autoregressive NN and trained to solve SHPP$n$ (SHPP of $n$ nodes) as "Reviser-$n$". A revision involves decomposing and reconstructing the initial tour, comprising four sequential steps outlined below.

**Decomposition** When improved by Reviser-$n$, a complete tour with $N$ nodes is randomly decomposed into $\lfloor \frac{N}{n} \rfloor$ subtours, each with $n$ nodes. There is no overlap between every two subtours. A "tail subtour" with $N \bmod n$ nodes, if any, is left untouched until composition. Each subtour corresponds to an SHPP graph, and reconstructing a subtour is equivalent to solving an SHPP instance. We pick the decomposition positions uniformly when performing repeated revisions.

**Transformation and augmentation** To improve the predictability and homogeneity of the model inputs, we apply Min-max Normalization and an optional rotation to the SHPP graphs. They scale the x-axis coordinates to the range $[0, 1]$ and set the lower bound of the y-axis to 0. In addition, we augment the SHPP instances by flipping the node coordinates to enhance the model performance.

**Solving SHPPs with local policies** We autoregressively reconstruct the subtours (i.e., solve the SHPP instances) with trainable revisers. Any SHPP solutions that are worse than the current ones will be discarded. This key step is detailed in Section 4.2.

**Composition** The $\lfloor \frac{N}{n} \rfloor$ reconstructed (or original) subtours and a tail subtour, if any, compose an improved complete tour by connecting the starting/terminating nodes of SHPPs in their original order.

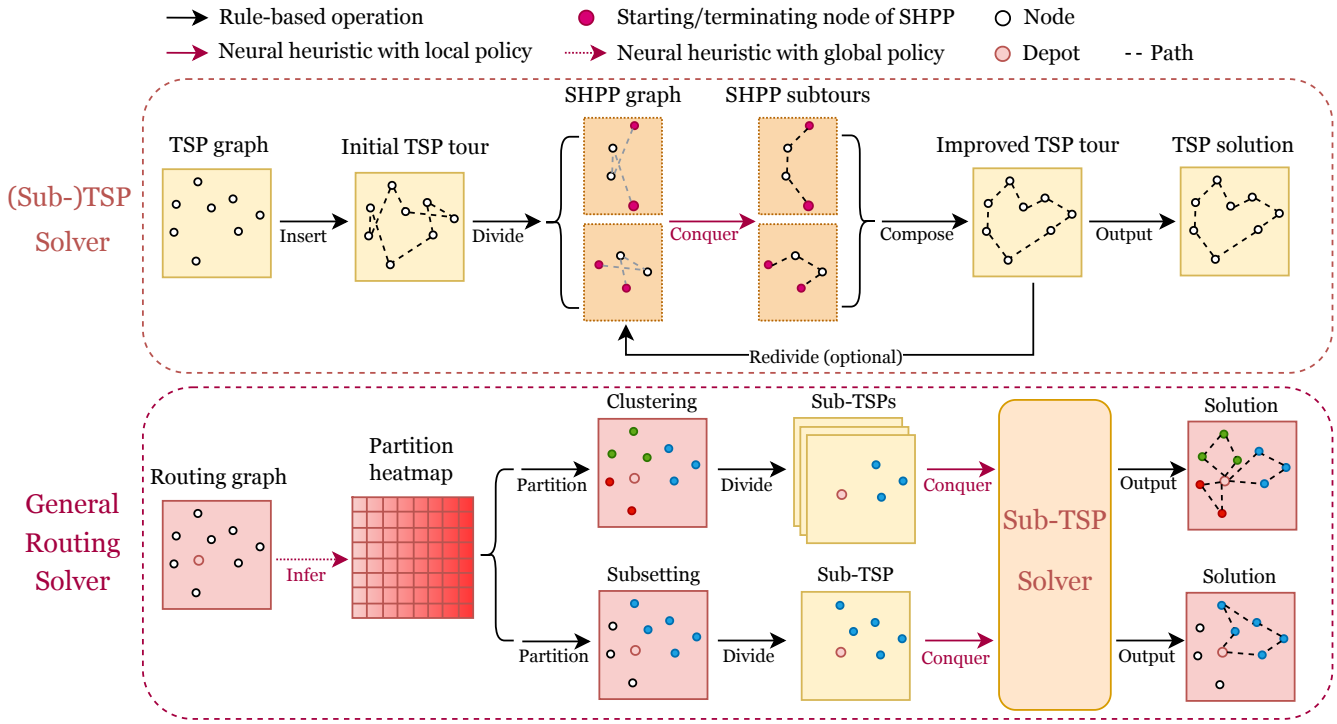


Figure 1: The pipeline of GLOP.

GLOP can apply multiple revisers to solve a problem from different angles. Also, a single reviser can decompose the tour at different points and repeat its revisions. After all revisions, GLOP outputs the improved tour as its final solution. The full details of this pipeline are given in Appendix A. Notably, GLOP allows applying a single set of small-SHPP-trained models for arbitrarily large TSP. For solving real-world instances, being a one-size-fits-all solution framework gives it an advantage over SOTA neural solvers that are specialized for specific problem scales and distributions (Sun and Yang 2023; Min, Bai, and Gomes 2023).

## 4.2 Solving SHPP with local policy

**Problem formulation and motivation** SHPP is also referred to as open-loop TSP. With starting/terminating nodes fixed, it aims to minimize the length of a Hamiltonian path visiting all nodes in between exactly once. Solving small SHPPs with neural networks, instead of directly solving TSP or with traditional heuristics, makes GLOP a highly parallelizable one-size-fits-all solution.

**Model** We parameterize our local policies based on Attention Model (AM) (Kool, van Hoof, and Welling 2019). To apply it to SHPP, we adjust its context embedding following Kim, Park, and Kim (2021) and leverage the solution symmetries by autoregressively constructing solutions from both starting/determining nodes.

**Local policy** Given an SHPP instance $\boldsymbol{s}$ with starting/terminating node 1 and $n$, our stochastic local policy $p_{\boldsymbol{\theta}}(\boldsymbol{\omega}_{fd}, \boldsymbol{\omega}_{bd} \mid \boldsymbol{s})$, parameterized by the neural model $\boldsymbol{\theta}$, denotes the conditional probability of constructing forward and backward-decoded solutions $\boldsymbol{\omega}_{fd}$ and $\boldsymbol{\omega}_{bd}$. We let $\boldsymbol{\omega}_{1:t-1}$ denote the partial solution at time step $t$, then the local policy can be factorized into probability distribution of per-step construction:

$$\begin{aligned} p_{\boldsymbol{\theta}}(\boldsymbol{\omega}_{fd}, \boldsymbol{\omega}_{bd} \mid \boldsymbol{s}) &= p_{\boldsymbol{\theta}}(\boldsymbol{\omega}_{fd} \mid \boldsymbol{s}) \times p_{\boldsymbol{\theta}}(\boldsymbol{\omega}_{bd} \mid \boldsymbol{s}) \\ &= \prod_{t=1}^{n-2} p_{\boldsymbol{\theta}}(\omega_t \mid \boldsymbol{s}, \boldsymbol{\omega}_{1:t-1}, n) \times p_{\boldsymbol{\theta}}(\omega_t \mid \boldsymbol{s}, \boldsymbol{\omega}_{1:t-1}, 1). \end{aligned} \tag{1}$$

We accept the better one between $\boldsymbol{\omega}_{fd}$ and $\boldsymbol{\omega}_{bd}$ during inference while making use of both for training.

## 4.3 Training algorithm

We train our parameterized local policy, i.e., a reviser, by minimizing the expected length of its constructed SHPP solutions:

$$\begin{aligned} \text{minimize} \quad \mathcal{L}(\boldsymbol{\theta} \mid \boldsymbol{s}) = \mathbb{E}_{\boldsymbol{\omega}_{fd}, \boldsymbol{\omega}_{bd} \sim p_{\boldsymbol{\theta}}(\boldsymbol{\omega}_{fd}, \boldsymbol{\omega}_{bd} \mid \boldsymbol{s})} \\ [f_{SHPP}(\boldsymbol{\omega}_{fd}, \boldsymbol{s}) + f_{SHPP}(\boldsymbol{\omega}_{bd}, \boldsymbol{s})], \end{aligned} \tag{2}$$

where $f_{SHPP}$ maps an SHPP solution to its length. We apply the REINFORCE-based gradient estimator (Williams 1992) using the average path length of two greedy rollouts as a baseline. This training algorithm doubles the experience learned on each instance and enables a more reliable baseline by weighing the greedy rollouts of both directions.

**Two-stage curriculum learning** According to our coordinate transformation, we design a two-stage curriculum to improve the homogeneity and consistency between training

and inference instances. We are motivated by the following observation: the inputs to revisers are 1) SHPP graphs with y-axis upper bounds ranging from 0 to 1 after our coordinate transformation, and also 2) the outputs of its preceding module. Therefore, stage 1 in our curriculum trains revisers using multi-distribution SHPPs with varied y-axis upper bounds, and stage 2 collaboratively fine-tunes all revisers following the inference pipeline.

# 5 General routing solver

Many routing problems can be formulated hierarchically, which requires node clustering (e.g. CVRP, mTSP, Capacitated Arc Routing Problem) or node subsetting (e.g. PCTSP, Orienteering Problem, Covering Salesman Problem), followed by solving multiple sub-TSPs or a single sub-TSP, respectively (Li et al. 2021a; Zhang et al. 2022a, 2021; Alesiani, Ermis, and Gkiotsalitis 2022; Xiao et al. 2019; Li et al. 2021b). For these general routing problem, GLOP involves an additional global partition policy defined by a parameterized partition heatmap (Section 5.1) and trained with parallel on-policy sampling without costly step-by-step neural decoding (Section 5.2).

## 5.1 Global policy as partition heatmap

**Partition heatmap** We introduce a parameterized partition heatmap $\boldsymbol{\mathcal{H}}_{\boldsymbol{\phi}}(\boldsymbol{\rho}) = [h_{ij}(\boldsymbol{\rho})]_{(n+1)\times(n+1)}$ where $\boldsymbol{\rho}$ is the input instance with $n+1$ nodes including node 0 as the depot. $h_{ij} \in \mathbb{R}^+$ represents the unnormalized probability of nodes $i$ and $j$ belonging to the same subset.

**Model and input graph** The partition heatmap is parameterized by an isomorphic GNN $\phi$ (Joshi, Laurent, and Bresson 2019; Qiu, Sun, and Yang 2022). Inputs to the model are sparsified graphs with features designed separately for different problems. We defer the full details to Appendix A.4.

**Global policy** For node clustering, GLOP partitions all nodes into multiple subsets, each corresponding to a sub-TSP to solve. For node subsetting, GLOP partitions all nodes into two subsets, i.e., the to-visit subset and the others, where the to-visit subset forms a sub-TSP to solve. Let $\boldsymbol{\pi} = \{\boldsymbol{\pi}^r\}_{r=1}^{|\boldsymbol{\pi}|}$ denote a complete partition and $\boldsymbol{\pi}^r = \{\pi_t^r\}_{t=1}^{|\boldsymbol{\pi}^r|}$ the $r$-th subset containing both regular nodes and the depot. Each subset begins and terminates at the depot; that is, $\pi_1^r = \pi_{|\boldsymbol{\pi}^r|}^r = 0$. Given $\boldsymbol{\mathcal{H}}_{\boldsymbol{\phi}}(\boldsymbol{\rho})$, our global policy partitions all nodes into $|\boldsymbol{\pi}|$ subsets by sequentially sampling nodes while satisfying problem-specific constraints $\Theta$:

$$p_{\boldsymbol{\phi}}(\boldsymbol{\pi}|\boldsymbol{\rho}) = \begin{cases} \prod_{r=1}^{|\boldsymbol{\pi}|} \prod_{t=1}^{|\boldsymbol{\pi}^r|-1} \frac{h_{\pi_t^r,\pi_{t+1}^r}(\boldsymbol{\rho})}{\sum_{k\in\mathcal{N}(\boldsymbol{\pi}^p)} h_{\pi_t^r,k}(\boldsymbol{\rho})}, & \text{if } \boldsymbol{\pi} \in \Theta, \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $\mathcal{N}(\boldsymbol{\pi}^p)$ is the set of feasible actions given the current partial partition. For our benchmark problems, the applied constraints $\Theta$ are given in Appendix A.4.

**Decoding** We apply greedy (GLOP-G) and sampling (GLOP-S) heatmap decoding (Qiu, Sun, and Yang 2022) to draw the node partitions following our global policy.

## 5.2 Training algorithm

We train our global policy to output partitions that could lead to the best-performing final solutions after solving sub-TSPs. Given each instance $\boldsymbol{\rho}$, the training algorithm infers partition heatmap $\boldsymbol{\mathcal{H}}_{\boldsymbol{\phi}}(\boldsymbol{\rho})$, samples node partitions in parallel, feeds the sampled partitions into GLOP for sub-TSP solutions, and optimizes the expected final performance:

$$\min \mathcal{L}(\boldsymbol{\phi}|\boldsymbol{\rho}) = \mathbb{E}_{\boldsymbol{\pi}\sim p_{\boldsymbol{\phi}}(\boldsymbol{\pi}|\boldsymbol{\rho})}[\sum_{r=1}^{|\boldsymbol{\pi}|} f_{TSP}(GLOP_{\boldsymbol{\theta}}(\boldsymbol{\pi}^r, \boldsymbol{\rho}))], \tag{4}$$

where $f_{TSP}$ is a mapping from a sub-TSP solution to its length, and $GLOP_{\boldsymbol{\theta}}$ generates sub-TSP solutions with well-trained local policies. We apply the REINFORCE algorithm (Williams 1992) with the averaged reward of sampled solutions for the same instance as a baseline. The baseline is respectively computed for each instance within each training batch. GLOP for sub-TSP, i.e. $GLOP_{\boldsymbol{\theta}}$, enables efficient training of our global policy on large-scale problems due to its parallelizability and scalability.

## 5.3 Applicability

Many routing problems can be formulated hierarchically, involving node clustering and/or node subsetting, depending on the problem formulation. Node clustering is used when the problem requires formulating multiple routes that cover all nodes, while node subsetting is used when the problem requires formulating a single route that covers a subset of nodes. In some cases, a routing problem may require both subsetting and clustering. Our global policy offers a unified formulation for all these scenarios. Additionally, it can easily handle constraints via masking if they can be anticipated while constructing node subsets. To handle more complex constraints, one can assign a large negative value to the rewards of infeasible solutions or apply post-processing techniques before solution evaluation.

# 6 Experimentation

## 6.1 Experimental Setup

**Datasets** We refer the readers to Kool, van Hoof, and Welling (2019) for more specific definitions of benchmark problems. (1) We evaluate GLOP on uniformly sampled large-scale TSP instances (i.e., TSP500, 1K, and 10K) used in Fu, Qiu, and Zha (2021) and an additionally generated TSP100K instance. We perform a cross-distribution evaluation on test instances used in (Bi et al. 2022). For evaluation on real-world benchmarks, we draw all 49 symmetric TSP instances featuring *EUC_2D* and containing fewer than 1000 nodes (since most baselines cannot process larger-scale instances) from TSPLIB and map all instances to the $[0, 1]^2$ square through Min-max Normalization. (2) For CVRP, we adhere to the settings in TAM (Hou et al. 2023) and use the code of AM (Kool, van Hoof, and Welling 2019) to generate test datasets on CVRP1K, 2K, 5K, and 7K, each containing 100 instances. The test datasets of ATSP are generated following Kwon et al. (2021). (3) For PCTSP, we follow the settings in AM (Kool, van Hoof, and Welling

| Method | TSP500 | | | TSP1K | | | TSP10K | | |
|---|---|---|---|---|---|---|---|---|---|
| | Obj. | Gap(%) | Time | Obj. | Gap(%) | Time | Obj. | Gap(%) | Time |
| Concorde (Applegate et al. 2006) | 16.55 | 0.00 | 40.9m | 23.12 | 0.00 | 8.2h | - | - | - |
| LKH-3 (Helsgaun 2017) | 16.55 | 0.00 | 5.5m | 23.12 | 0.00 | 24m | 71.77 | 0.00 | 13h |
| Random Insertion | 18.59 | 12.3 | <1s | 26.12 | 13.0 | <1s | 81.84 | 14.0 | 5.7s |
| AM (Kool, van Hoof, and Welling 2019) | 22.60 | 36.6 | 5.8m | 42.53 | 84.0 | 22m | 430 | 499 | 3.5m |
| LCP (Kim, Park, and Kim 2021) | 20.82 | 25.8 | 29m | 36.34 | 57.2 | 34m | 357 | 397 | 4.3m |
| GCN+MCTS ×12 (Fu, Qiu, and Zha 2021) | 16.96 | 2.48 | 2.4m+33s | 23.86 | 3.20 | 4.9m+1.2m | 75.73 | 5.50 | 7.1m+6.0m |
| POMO-EAS (Hottung, Kwon, and Tierney 2022) | 24.04 | 45.3 | 1.0h | 47.79 | 107 | 8.6h | | OOM | |
| DIMES+S (Qiu, Sun, and Yang 2022) | 19.06 | 15.0 | 2.0m | 26.96 | 16.1 | 2.4m | 86.25 | 20.0 | 3.1m |
| DIMES+MCTS ×12 (Qiu, Sun, and Yang 2022) | 17.01 | 2.78 | 1.0m+2.1m | 23.86 | 3.20 | 2.6m+1.0m | 76.02 | 5.90 | 13.7m+20m |
| Tspformer* (Yang et al. 2023) | 17.57 | 5.97 | 3.1m | 27.02 | 16.9 | 5.0m | - | - | - |
| H-TSP (Pan et al. 2023) | - | - | - | 24.65 | 6.62 | 47s | 77.75 | 7.32 | 48s |
| Pointerformer (Jin et al. 2023) | 17.14 | 3.56 | 1.0m | 24.80 | 7.30 | 6.5m | - | - | - |
| DeepACO (Ye et al. 2023) | 16.94 | 2.36 | 4.3m | 23.85 | 3.16 | 1.1h | - | - | - |
| GLOP | 17.07 | 3.14 | **19s** | 24.01 | 3.85 | **34s** | 75.62 | 5.36 | **32s** |
| GLOP (more revisions) | **16.91** | **1.99** | 1.5m | **23.84** | **3.11** | 3.0m | **75.29** | **4.90** | 1.8m |

Table 1: Comparison results on 128 TSP500, 128 TSP1K, and 16 TSP10K. For all experiments on TSP, "Time" is the total runtime for solving all instances. If it has two terms, they correspond to the runtime of heatmap generation and MCTS, respectively. OOM: out of our graphics memory (24GB). *: Results are drawn from the original literature with runtime proportionally adjusted (128/100) to match the size of our test datasets. See Appendix A.6 and E for full implementation details of GLOP and the baselines, respectively.

| Method | TSP100K | | |
|---|---|---|---|
| | Obj. | Gap(%) | Time |
| LKH-3 $\{T = 1\}$ | 226.4 | 0.00 | 8.1h |
| Random Insertion | 258.5 | 14.2 | 1.7m |
| AM | | OOM | |
| LCP | | OOM | |
| GCN+MCTS ×12 | | OOM | |
| POMO-EAS | | OOM | |
| DIMES+MCTS ×12 | | OOM | |
| DIMES+S | 286.1 | 26.4 | 2.0m |
| H-TSP | | OOM | |
| Pointerformer | | OOM | |
| GLOP | 240.0 | 6.01 | **1.8m** |
| GLOP (more revisions) | **238.0** | **5.10** | 2.8m |

Table 2: Comparison results on a TSP100K instance.

| Method | Avg. gap(%) | Time |
|---|---|---|
| LCP $\{M = 1280\}$ | 99.9 | 3.6m |
| DACT $\{T = 1K\}$ | 865 | 50m |
| GCN+MCTS ×1 | 1.10 | 7.5m |
| POMO-EAS $\{T = 60\}$ | 18.8 | 20m |
| DIMES+MCTS ×1 | 2.21 | 7.4m |
| AMDKD+EAS $\{T = 100\}$ | 7.86* | 48m |
| Pointerformer | 6.04 | 48s |
| GLOP | 1.53 | **42s** |
| GLOP (more revisions) | **0.69** | 2.6m |

*: Two instances are skipped due to OOM issue.

Table 3: Comparison results on TSPLIB instances.

| Method | MatNet (Kwon et al. 2021) | GLOP |
|---|---|---|
| ATSP150 | 2.88 (7.2s) | **1.89 (6.4s)** |
| ATSP250 | 4.49 (12s) | **2.10 (9.6s)** |
| ATSP1000 | - | **2.79 (39s)** |

Table 4: Comparison results on ATSP.

2019) for data generation on PCTSP500, 1K, and 5K. As suggested by Kool, van Hoof, and Welling (2019), we specify $K^n = 9, 12, 20$ to sample prizes for $n$ =500, 1K, 5K, respectively , i.e., $\beta_i \sim \text{Uniform}(0, 3\frac{K^n}{n})$.

**Baselines** Evaluating an NCO method typically involves two metrics: the objective value and the runtime. To ensure the validity of our comparisons, we select SOTA baselines with adjustable runtime that can match GLOP. We defer detailed implementations of the baselines to Appendix E.

**Hardware** Unless otherwise stated, GLOP and the baselines are executed on a 12-core Intel(R) Xeon(R) Platinum 8255C CPU and an NVIDIA RTX 3090 Graphics Card.

## 6.2 Travelling Salesman Problem

**Large-scale TSP** Comparison results on large-scale TSP are shown in Table 1 and Table 2. For GCN+MCTS (Fu, Qiu, and Zha 2021) and DIMES+MCTS (Qiu, Sun, and Yang 2022), we use all 12 CPU cores for MCTS and limit its running time to ensure comparable results. Concluded from the results, GLOP is highly efficient due to its decomposed solution scheme (see Appendix D.5 for the analysis of time complexity). Furthermore, the memory consumption of GLOP can be basically invariant of the problem scale if reconstructing the subtours using a fixed batch size. Hence, it is the first neural solver to effectively scale to TSP100K,

| Method | Time | Uniform | Expansion | | Explosion | | Implosion | |
|---|---|---|---|---|---|---|---|---|
| | | Gap(%) | Gap(%) | Det.(%) | Gap(%) | Det.(%) | Gap(%) | Det.(%) |
| AM (Kool, van Hoof, and Welling 2019) | 0.5h | 2.310 | 17.97 | 678 | 3.817 | 65 | 2.431 | 5.2 |
| AM+HAC (Zhang et al. 2022b) | 0.5h | 2.484 | 3.997 | **61** | 3.084 | 24 | 2.595 | 4.5 |
| GLOP | | **0.091** | **0.166** | 82 | **0.066** | **-27** | **0.082** | **-9.9** |
| AMDKD+EAS (Bi et al. 2022) | 2.0h | 0.078 | 0.165 | 112 | 0.048 | -39 | 0.079 | 1.3 |
| GLOP (more revisions) | | **0.048** | **0.076** | **60** | **0.028** | **-41** | **0.044** | **-8.3** |

Table 5: Comparison results on the OoD datasets. Det $= \text{Gap}_{OoD}/\text{Gap}_U - 1$, where $\text{Gap}_{OoD}$ and $\text{Gap}_U$ are the optimality gaps on an OoD dataset and the Uniform dataset, respectively.

| Method | CVRP1K | | CVRP2K | | CVRP5K | | CVRP7K | |
|---|---|---|---|---|---|---|---|---|
| | Obj. | Time (s) | Obj. | Time (s) | Obj. | Time (s) | Obj. | Time (s) |
| LKH-3 (Helsgaun 2017) | 46.4 | 6.2 | 64.9 | 20 | 175.7 | 152 | 245.0 | 501 |
| AM (Kool, van Hoof, and Welling 2019) | 61.4 | 0.6 | 114.4 | 1.9 | 257.1 | 12 | 354.3 | 26 |
| L2I (Lu, Zhang, and Yang 2020) | 93.2 | 6.3 | 138.8 | 25 | - | - | - | - |
| NLNS (Hottung and Tierney 2019) | 53.5 | 198 | - | - | - | - | - | - |
| L2D (Li, Yan, and Wu 2021) | 46.3 | 1.5 | 65.2 | 38 | - | - | - | - |
| RBG (Zong et al. 2022a) | 74.0 | 13 | 137.6 | 42 | - | - | - | - |
| TAM-AM (Hou et al. 2023) | 50.1 | 0.8 | 74.3 | 2.2 | 172.2 | 12 | 233.4 | 26 |
| TAM-LKH3 (Hou et al. 2023) | 46.3 | 1.8 | 64.8 | 5.6 | 144.6 | 17 | 196.9 | 33 |
| TAM-HGS (Hou et al. 2023) | - | - | - | - | 142.8 | 30 | 193.6 | 52 |
| GLOP-G | 47.1 | **0.4** | 63.5 | **1.2** | 141.9 | **1.7** | 191.7 | **2.4** |
| GLOP-G (LKH-3) | **45.9** | 1.1 | **63.0** | 1.5 | **140.6** | 4.0 | **191.2** | 5.8 |

Table 6: Comparison results on large-scale CVRP following the settings in (Hou et al. 2023). "Time" corresponds to the per-instance runtime. GLOP-G (LKH-3) applies LKH-3 as its sub-TSP solver.

obtaining a 5.1% optimality gap and a $174\times$ speed-up compared to LKH-3. Compared with LCP (Kim, Park, and Kim 2021) and H-TSP (Pan et al. 2023), GLOP dispenses with learning upper-level TSP policies while achieving better performance. Compared with NAR methods conducting MCTS refinements (Fu, Qiu, and Zha 2021; Qiu, Sun, and Yang 2022), GLOP generates reasonable solutions even before they have finished initialization or produced prerequisite heatmaps for solution decoding. Hence, GLOP exhibits clear advantages for real-time applications.

**Cross-distribution TSP** Table 5 gathers the comparison between GLOP and two baselines specially devised for cross-distribution performance (Zhang et al. 2022b; Bi et al. 2022) on four TSP100 datasets with different distributions, i.e., uniform, expansion, explosion, and implosion. Recall that we use uniformly distributed samples to train GLOP. Hence, the latter three datasets contain out-of-distribution (OoD) instances. Results show that GLOP obtains smaller gaps and less Det. on most OoD datasets. We argue that the holistic solution scheme of GLOP is the main contributor to its cross-distribution performance. The inputs to the neural models are local SHPP graphs, making GLOP insensitive to the overall TSP distribution. We conduct further discussions in Appendix D.3.

**Real-world TSPLIB** We evaluate GLOP and the baselines on real-world TSPLIB instances and collect the results in Table 3, where GLOP performs favorably against the baselines due to its consistent performance across scales and distributions. Note that we test each instance individually without parallel computation.

**Asymmetric TSP** GLOP is compatible with any neural architecture and can be extended to asymmetric distance. Table 4 exemplifies this flexibility on ATSP, where we replace AM with MatNet (Kwon et al. 2021) which is specially designed for ATSP. We follow the experimental setup in (Kwon et al. 2021) and generalize MatNet100 as baseline. For GLOP, we apply MatNet50 checkpoint as our reviser without retraining. The results validate that GLOP can successfully extend MatNet to solve large ATSP instances. Note that MatNet is limited to problem scales no larger than 256 due to its one-hot initialization while there is no such limitation for GLOP-empowered MatNet.

## 6.3 Capacitated Vehicle Routing Problem

GLOP for CVRP involves node clustering with our global policy, followed by solving the produced sub-TSPs with our sub-TSP solver.

**Large-scale CVRP** Table 6 summarizes the results of the comparison in large-scale CVRP, where the RBG (Zong

| Instance | Scale | AM | | TAM-AM | | LKH-3 | | TAM-LKH3 | | GLOP | | GLOP-LKH3 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Gap | Time | Gap | Time | Gap | Time | Gap | Time | Gap | Time | Gap | Time |
| LEUVEN1 | 3001 | 46.9 | 10s | 20.2 | 10s | 18.1 | 69s | 19.3 | 16s | **16.9** | **2s** | **16.6** | **8s** |
| LEUVEN2 | 4001 | 53.3 | 13s | 38.6 | 14s | 22.1 | 74s | 15.9 | 24s | 21.8 | **3s** | 21.1 | **3s** |
| ANTWERP1 | 6001 | 39.3 | 13s | 24.9 | 13s | 24.2 | 596s | 24.0 | 25s | **20.3** | **3s** | **19.3** | **14s** |
| ANTWERP2 | 7001 | 50.3 | 15s | 33.2 | 15s | 31.1 | 479s | 22.6 | 32s | **19.4** | **4s** | **19.4** | **7s** |
| GHENT1 | 10001 | 46.9 | 21s | 30.2 | 22s | - | | 29.5 | 37s | **20.3** | **5s** | **18.3** | **22s** |
| GHENT2 | 11001 | 52.2 | 39s | 33.3 | 38s | - | | 23.7 | 56s | **19.8** | **6s** | **18.1** | **8s** |
| BRUSSELS1 | 15001 | 52.4 | 131s | 43.4 | 139s | - | | 27.2 | 167s | 27.6 | **8s** | 27.5 | **26s** |
| BRUSSELS2 | 16001 | 52.4 | 166s | 39.0 | 159s | - | | 37.1 | 187s | **22.4** | **9s** | **20.1** | **14s** |

Table 7: Comparison results on large-scale CVRPLIB instances.

| Method | PCTSP500 | | PCTSP1K | | PCTSP5K | |
|---|---|---|---|---|---|---|
| | Obj. | Time | Obj. | Time | Obj. | Time |
| OR Tools | 15.0 | 1h | 24.9 | 1h | 63.3 | 1h |
| OR Tools (more iterations) | 14.4 | 16h | 20.6 | 16h | 54.4 | 16h |
| AM (Kool, van Hoof, and Welling 2019) | 19.3 | 14m | 34.8 | 23m | 175 | 21m |
| MDAM (Xin et al. 2021a) | 14.8 | 2.8m | 22.2 | 17m | 58.9 | 3h |
| GLOP-G | 14.6 | **26s** | 20.0 | **47s** | 46.0 | **3.7m** |
| GLOP-S | **14.3** | 1.5m | **19.8** | 2.5m | **44.9** | 16m |

Table 8: Comparison results of GLOP and the baselines on 128 PCTSP500, 1K, and 5K. "Time" corresponds to the total execution time for solving all instances.

et al. 2022a) and L2D (Li, Yan, and Wu 2021) models are generalized for evaluation here due to their different training settings, and the results of other baselines are drawn from (Hou et al. 2023). Here, we apply the global policy trained on CVRP2K to both CVRP5K and CVRP7K, verifying the generalization performance of GLOP. Compared to the methods that entail iterative solution refinement (Helsgaun 2017; Li, Yan, and Wu 2021; Lu, Zhang, and Yang 2020; Zong et al. 2022a), both TAM (Hou et al. 2023) and GLOP can deliver more real-time solutions. Compared with prior SOTA real-time solver TAM, GLOP learns more effective global/local policies and enables higher decoding efficiency. Hence, GLOP outperforms TAM regarding both solution quality and efficiency.

**Real-world CVRPLIB** We test GLOP on large-scale CVRPLIB instances and present results in Table 7. We generalize the model trained on CVRP2000 to these large instances and compare GLOP to prior SOTA real-time solver TAM (Hou et al. 2023). The results of TAM and other baselines are drawn from (Hou et al. 2023). The comparison also demonstrates the superiority of GLOP in both solution quality and efficiency, especially for very large instances.

### 6.4 Prize Collecting Travelling Salesman Problem

GLOP for PCTSP involves node subsetting with our global policies, followed by solving a sub-TSP instance with our sub-TSP solvers. Large-scale PCTSP entails both a scalable global policy to generate a promising node partition and a scalable local policy to tackle the equivalently large sub-TSP. We evaluate greedy (GLOP-G) and sampling (GLOP-S) decoding for our global policy. The comparison results on large-scale PCTSP are displayed in Table 8, where GLOP surpasses recent neural solvers and conventional solvers in terms of both solution quality and efficiency.

## 7 Conclusion and limitation

This paper proposes GLOP to learn global policies for coarse-grained problem partitioning and local policies for fine-grained route construction. GLOP leverages the scalability of the NAR paradigm and meticulousness of the AR paradigm, making the first effective attempt at hybridizing them. Extensive evaluations on large-scale TSP, ATSP, CVRP, and PCTSP demonstrate its competitive and SOTA real-time performance.

However, GLOP might be less competitive in application scenarios where prolonged execution time is allowed. In terms of its ability to trade off the execution time for solution quality, GLOP might be inferior to the methods based on iterative solution refinement. Our future focus will be on addressing this limitation. We also plan to investigate the emerging possibilities that arise when viewing AR and NAR methods from a unified perspective. In addition, it may be promising to exploit unsupervised Deep Graph Clustering techniques (Yue et al. 2022) or formulate node classification tasks to solve large-scale routing problems hierarchically.

## References


Alesiani, F.; Ermis, G.; and Gkiotsalitis, K. 2022. Constrained Clustering for the Capacitated Vehicle Routing Problem (CC-CVRP). *Applied artificial intelligence*, 36(1): 1995658.

Applegate, D.; Bixby, R.; Chvatal, V.; and Cook, W. 2006. Concorde TSP solver.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2016. Neural combinatorial optimization with reinforcement learning. *ArXiv preprint*, abs/1611.09940.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421.

Berto, F.; Hua, C.; Park, J.; Kim, M.; Kim, H.; Son, J.; Kim, H.; Kim, J.; and Park, J. 2023. Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark. *arXiv preprint arXiv:2306.17100*.

Bi, J.; Ma, Y.; Wang, J.; Cao, Z.; Chen, J.; Sun, Y.; and Chee, Y. M. 2022. Learning Generalizable Models for Vehicle Routing Problems via Knowledge Distillation. *ArXiv preprint*, abs/2210.07686.

Bogyrbayeva, A.; Meraliyev, M.; Mustakhov, T.; and Dauletbayev, B. 2022. Learning to Solve Vehicle Routing Problems: A Survey. *ArXiv preprint*, abs/2205.02453.

Bresson, X.; and Laurent, T. 2021. The transformer network for the traveling salesman problem. *ArXiv preprint*, abs/2103.03012.

Chen, J.; Zong, Z.; Zhuang, Y.; Yan, H.; Jin, D.; and Li, Y. 2022. Reinforcement Learning for Practical Express Systems with Mixed Deliveries and Pickups. *ACM Transactions on Knowledge Discovery from Data (TKDD)*.

Cheng, H.; Zheng, H.; Cong, Y.; Jiang, W.; and Pu, S. 2023. Select and Optimize: Learning to solve large-scale TSP instances. In Ruiz, F.; Dy, J.; and van de Meent, J.-W., eds., *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, 1219–1231. PMLR.

Choo, J.; Kwon, Y.-D.; Kim, J.; Jae, J.; Hottung, A.; Tierney, K.; and Gwon, Y. 2022. Simulation-guided beam search for neural combinatorial optimization. *ArXiv preprint*, abs/2207.06190.

d O Costa, P. R.; Rhuggenaath, J.; Zhang, Y.; and Akcay, A. 2020. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian Conference on Machine Learning*, 465–480. PMLR.

Fan, M.; Wu, Y.; Liao, T.; Cao, Z.; Guo, H.; Sartoretti, G.; and Wu, G. 2022. Deep Reinforcement Learning for UAV Routing in the Presence of Multiple Charging Stations. *IEEE Transactions on Vehicular Technology*.

Fu, Z.; Qiu, K.; and Zha, H. 2021. Generalize a Small Pretrained Model to Arbitrarily Large TSP Instances. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, 7474–7482. AAAI Press.

Goh, Y. L.; Lee, W. S.; Bresson, X.; Laurent, T.; and Lim, N. 2022. Combining reinforcement learning and optimal transport for the traveling salesman problem. *ArXiv preprint*, abs/2203.00903.

Grinsztajn, N.; Furelos-Blanco, D.; and Barrett, T. D. 2022. Population-Based Reinforcement Learning for Combinatorial Optimization. *ArXiv preprint*, abs/2210.03475.

Helsgaun, K. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, 24–50.

Hottung, A.; Kwon, Y.; and Tierney, K. 2022. Efficient Active Search for Combinatorial Optimization Problems. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Hottung, A.; and Tierney, K. 2019. Neural large neighborhood search for the capacitated vehicle routing problem. *ArXiv preprint*, abs/1911.09539.

Hou, Q.; Yang, J.; Su, Y.; Wang, X.; and Deng, Y. 2023. Generalize Learned Heuristics to Solve Large-scale Vehicle Routing Problems in Real-time. In *The Eleventh International Conference on Learning Representations*.

Hudson, B.; Li, Q.; Malencia, M.; and Prorok, A. 2022. Graph Neural Network Guided Local Search for the Traveling Salesperson Problem. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Jiang, Y.; Wu, Y.; Cao, Z.; and Zhang, J. 2022. Learning to Solve Routing Problems via Distributionally Robust Optimization. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, 9786–9794. AAAI Press.

Jin, Y.; Ding, Y.; Pan, X.; He, K.; Zhao, L.; Qin, T.; Song, L.; and Bian, J. 2023. Pointerformer: Deep Reinforced Multi-Pointer Transformer for the Traveling Salesman Problem. *ArXiv preprint*, abs/2304.09407.

Joshi, C. K.; Cappart, Q.; Rousseau, L.-M.; and Laurent, T. 2022. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 1–29.

Joshi, C. K.; Laurent, T.; and Bresson, X. 2019. An efficient graph convolutional network technique for the travelling salesman problem. *ArXiv preprint*, abs/1906.01227.

Kim, M.; Park, J.; and Kim, J. 2021. Learning Collaborative Policies to Solve NP-hard Routing Problems. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 10418–10430.

Kim, M.; Park, J.; and Park, J. 2022. Sym-NCO: Leveraging Symmetricity for Neural Combinatorial Optimization. *ArXiv preprint*, abs/2205.13209.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In Bengio, Y.; and LeCun, Y., eds., *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kool, W.; van Hoof, H.; Gromicho, J.; and Welling, M. 2022. Deep policy dynamic programming for vehicle routing problems. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 190–213. Springer.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, Learn to Solve Routing Problems! In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Kwon, Y.; Choo, J.; Kim, B.; Yoon, I.; Gwon, Y.; and Min, S. 2020. POMO: Policy Optimization with Multiple Optima for Reinforcement Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Kwon, Y.; Choo, J.; Yoon, I.; Park, M.; Park, D.; and Gwon, Y. 2021. Matrix encoding networks for neural combinatorial optimization. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 5138–5149.

Li, J.; Ma, Y.; Gao, R.; Cao, Z.; Lim, A.; Song, W.; and Zhang, J. 2021a. Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem. *IEEE Transactions on Cybernetics*, 52(12): 13572–13585.

Li, K.; Zhang, T.; Wang, R.; Wang, Y.; Han, Y.; and Wang, L. 2021b. Deep reinforcement learning for combinatorial optimization: Covering salesman problems. *IEEE transactions on cybernetics*, 52(12): 13142–13155.

Li, S.; Yan, Z.; and Wu, C. 2021. Learning to delegate for large-scale vehicle routing. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 26198–26211.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search. In Bengio, S.; Wallach, H. M.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 537–546.

Loshchilov, I.; and Hutter, F. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Lu, H.; Zhang, X.; and Yang, S. 2020. A Learning-based Iterative Method for Solving Vehicle Routing Problems. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Ma, Q.; Ge, S.; He, D.; Thaker, D.; and Drori, I. 2019. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *ArXiv preprint*, abs/1911.04936.

Ma, Y.; Li, J.; Cao, Z.; Song, W.; Zhang, L.; Chen, Z.; and Tang, J. 2021. Learning to Iteratively Solve Routing Problems with Dual-Aspect Collaborative Transformer. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 11096–11107.

Mazyavkina, N.; Sviridov, S.; Ivanov, S.; and Burnaev, E. 2021. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134: 105400.

Min, Y.; Bai, Y.; and Gomes, C. P. 2023. Unsupervised Learning for Solving the Travelling Salesman Problem. *ArXiv preprint*, abs/2303.10538.

Miranda-Bront, J. J.; Curcio, B.; Méndez-Díaz, I.; Montero, A.; Pousa, F.; and Zabala, P. 2017. A cluster-first route-second approach for the swap body vehicle routing problem. *Annals of Operations Research*, 253: 935–956.

Nowak, A.; Villar, S.; Bandeira, A. S.; and Bruna, J. 2018. Revised note on learning quadratic assignment with graph neural networks. In *2018 IEEE Data Science Workshop (DSW)*, 1–5. IEEE.

Pan, X.; Jin, Y.; Ding, Y.; Feng, M.; Zhao, L.; Song, L.; and Bian, J. 2023. H-TSP: Hierarchically Solving the Large-Scale Travelling Salesman Problem. *ArXiv preprint*, abs/2304.09395.

Qiu, R.; Sun, Z.; and Yang, Y. 2022. DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems. *ArXiv preprint*, abs/2210.04123.

Song, W.; Mi, N.; Li, Q.; Zhuang, J.; and Cao, Z. 2023. Stochastic Economic Lot Scheduling via Self-Attention Based Deep Reinforcement Learning. *IEEE Transactions on Automation Science and Engineering*.

Sui, J.; Ding, S.; Liu, R.; Xu, L.; and Bu, D. 2021. Learning 3-opt heuristics for traveling salesman problem via deep reinforcement learning. In *Asian Conference on Machine Learning*, 1301–1316. PMLR.

Sun, Z.; and Yang, Y. 2023. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization. *ArXiv preprint*, abs/2302.08224.

Taillard, É. D.; and Helsgaun, K. 2019. POPMUSIC for the travelling salesman problem. *European Journal of Operational Research*, 272(2): 420–429.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer Networks. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2692–2700.

Wang, C.; Yang, Y.; Slumbers, O.; Han, C.; Guo, T.; Zhang, H.; and Wang, J. 2021. A Game-Theoretic Approach for Improving Generalization Ability of TSP Solvers. *ArXiv preprint*, abs/2110.15105.

Wang, C.; Yu, Z.; McAleer, S.; Yu, T.; and Yang, Y. 2023. ASP: Learn a Universal Neural Solver! *ArXiv preprint*, abs/2303.00466.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, 5–32.

Wu, Y.; Song, W.; Cao, Z.; and Zhang, J. 2021a. Learning Large Neighborhood Search Policy for Integer Programming. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 30075–30087.

Wu, Y.; Song, W.; Cao, Z.; Zhang, J.; and Lim, A. 2021b. Learning improvement heuristics for solving routing problems.. *IEEE transactions on neural networks and learning systems*.

Wu, Y.; Zhou, J.; Xia, Y.; Zhang, X.; Cao, Z.; and Zhang, J. 2023. Neural Airport Ground Handling. *ArXiv preprint*, abs/2303.02442.

Xiao, J.; Zhang, T.; Du, J.; and Zhang, X. 2019. An evolutionary multiobjective route grouping-based heuristic algorithm for large-scale capacitated vehicle routing problems. *IEEE transactions on cybernetics*, 51(8): 4173–4186.

Xin, L.; Song, W.; Cao, Z.; and Zhang, J. 2021a. Multi-Decoder Attention Model with Embedding Glimpse for Solving Vehicle Routing Problems. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, 12042–12049. AAAI Press.

Xin, L.; Song, W.; Cao, Z.; and Zhang, J. 2021b. NeuroLKH: Combining Deep Learning Model with Lin-Kernighan-Helsgaun Heuristic for Solving the Traveling Salesman Problem. In Ranzato, M.; Beygelzimer, A.; Dauphin, Y. N.; Liang, P.; and Vaughan, J. W., eds., *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, 7472–7483.

Yang, H.; Zhao, M.; Yuan, L.; Yu, Y.; Li, Z.; and Gu, M. 2023. Memory-efficient Transformer-based network model for Traveling Salesman Problem. *Neural Networks*, 161: 589–597.

Ye, H.; Wang, J.; Cao, Z.; Liang, H.; and Li, Y. 2023. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization. *ArXiv preprint*, abs/2309.14032.

Yue, L.; Jun, X.; Sihang, Z.; Siwei, W.; Xifeng, G.; Xihong, Y.; Ke, L.; Wenxuan, T.; Wang, L. X.; et al. 2022. A survey of deep graph clustering: Taxonomy, challenge, and application. *ArXiv preprint*, abs/2211.12875.

Zhang, R.; Zhang, C.; Cao, Z.; Song, W.; Tan, P. S.; Zhang, J.; Wen, B.; and Dauwels, J. 2022a. Learning to solve multiple-TSP with time window and rejections via deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*.

Zhang, Y.; Mei, Y.; Huang, S.; Zheng, X.; and Zhang, C. 2021. A route clustering and search heuristic for large-scale multidepot-capacitated arc routing problem. *IEEE Transactions on Cybernetics*, 52(8): 8286–8299.

Zhang, Z.; Zhang, Z.; Wang, X.; and Zhu, W. 2022b. Learning to Solve Travelling Salesman Problem with Hardness-Adaptive Curriculum. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, 9136–9144. AAAI Press.

Zhou, J.; Wu, Y.; Song, W.; Cao, Z.; and Zhang, J. 2023. Towards Omni-generalizable Neural Methods for Vehicle Routing Problems. In *the 40th International Conference on Machine Learning (ICML 2023)*.

Zong, Z.; Wang, H.; Wang, J.; Zheng, M.; and Li, Y. 2022a. RBG: Hierarchically Solving Large-Scale Routing Problems in Logistic Systems via Reinforcement Learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 4648–4658.

Zong, Z.; Zheng, M.; Li, Y.; and Jin, D. 2022b. MAPDP: Cooperative Multi-Agent Reinforcement Learning to Solve Pickup and Delivery Problems. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, 9980–9988. AAAI Press.

# A Details of GLOP

## A.1 Pseudo code of solving (sub-)TSP

We present the pseudo code of solving (sub-)TSP in Algorithm 1. Note that we can generate multiple initial solutions (i.e., $W > 1$) and pick the best one after all revisions. In line 12, the strategy for shifting the decomposition point is to minimize the subtour overlap between revisions.

## A.2 Coordinate transformation

Recall that we solve SHPPs with local policies parameterized by deep neural models, i.e., revisers, and the coordinates of a subtour are the inputs to a reviser. In the inference phase, we apply a coordinate transformation to improve the predictability and homogeneity of the model inputs. It facilitates training and benefits inference performance. Let $(x'_i, y'_i)$ denotes the coordinates of the $i$th node after transformation; $x_{\max}$, $x_{\min}$, $y_{\max}$, and $y_{\min}$ denote the bounds of an SHPP graph. Then the coordinate transformation is formulated as

$$x'_i = \begin{cases} sc(x_i - x_{\min}) & \text{if } x_{\max} - x_{\min} > y_{\max} - y_{\min}, \\ sc(y_i - y_{\min}) & \text{otherwise,} \end{cases}$$
$$y'_i = \begin{cases} sc(y_i - y_{\min}) & \text{if } x_{\max} - x_{\min} > y_{\max} - y_{\min}, \\ sc(x_i - x_{\min}) & \text{otherwise,} \end{cases} \tag{5}$$

where the scale coefficient $sc = \frac{1}{\max(x_{\max} - x_{\min}, y_{\max} - y_{\min})}$.

With the above Min-max Normalization and an optional graph rotation, we scale the x-axis coordinates to $[0, 1]$ and set the y-axis lower bound to 0. Nevertheless, the revisers need to handle inputs of varied y-axis upper bound. It motivates us to develop the multi-distribution curriculum.

## A.3 Curriculum learning

**Stage 1: multi-distribution training** The first curriculum stage trains all revisers using multi-distribution SHPPs. For each training instance, we first sample a y-axis upper bound: $y_{\max} \sim (0, 1]$, then the instance: $(x, y) \sim [0, 1] \times [0, y_{\max}]$, both uniformly.

**Stage 2: collaborative training** We first generate TSP instances with nodes sampled uniformly in $[0, 1]^2$, then decompose their insertion-generated TSP tours into SHPPs. These SHPPs are used to fine-tune the first reviser. Each fine-tuned reviser is applied to infer its training instances, and the output subtours are decomposed into smaller-scale SHPPs to fine-tune the next reviser.

## A.4 Details of our global policy

**Constraints** $\Theta$ in Eq. (3) requires specification for certain problems. For CVRP, we constrain $|\boldsymbol{\pi}|$ to the maximum number of vehicles by using the mask function suggested by Hou et al. (2023) and prohibit revisiting the same nodes or exceeding vehicle capacity. For PCTSP, we set $|\boldsymbol{\pi}|$ to 1 and prohibit revisiting the same nodes or violating the minimum prize constraint.

**Inputs** For the CVRP input graph, the node features are the normalized demand and polar coordinates w.r.t. the depot; the edge attributes are the relative Euclidean distance and polar angle. We sparsify the input graph by restricting each node to only connect with its $k$ nearest neighbors based on their polar angles. $k$ is set to 100 for CVRP1K and 200 for the rest. For the PCTSP input graph, the node features are the normalized prize and penalty; the edge attribute is the relative Euclidean distance. We sparsify the graph based on the Euclidean distance, setting $k$ to 50, 100, and 200 for PCTSP500, 1K, and 5K, respectively.

## A.5 Model architecture and training settings

**Local policy** We deploy revisers of four scales: Reviser-100, 50, 20, and 10. The hyperparameter configurations of our revisers are the same as (Kool, van Hoof, and Welling 2019), except that we use 6 encoder layers instead of 3. We apply Adam optimizer (Kingma and Ba 2015) ($\eta = 10^{-4}$). The 1st-stage curriculum trains Reviser-10 for 100 epochs, while others for 200 epochs. An epoch contains 1.28M SHPP instances. The 2nd-stage curriculum trains all revisers for 300 epochs with a learning rate decay of 0.99. A training epoch goes through a training dataset of 1M SHPP instances.

**Global policy** We employ the same model architecture as (Qiu, Sun, and Yang 2022), except that we set the dimension of node embedding to 48 on CVRP. We apply AdamW optimizer ($\eta = 1 \times 10^{-4}$ for PCTSP; $\eta = 3 \times 10^{-4}$ for CVRP) (Loshchilov and Hutter 2019) with Cosine Annealing scheduler (Loshchilov and Hutter 2017) and train each global policy with 5.12K instances.

## A.6 Inference settings

**TSP** The used hyperparameters are gathered in Table 9. For the cross-distribution evaluation on TSP100, we skip instance augmentation and directly treat TSP100 as SHPP100 when implementing Reviser-100.

**CVRP** Our global policies implement greedy decoding for CVRP. The hyperparameter settings for the local policies are displayed in Table 10.

**PCTSP** On PCTSP, our local policies apply both greedy and sampling decoding. For the latter, the number of sampled partitions is set to 10. As for the local policies, we deploy Reviser-100, 50, and 20, and conduct 10, 10, and 5 revisions, respectively.

# B Discussing two paradigms

From a unified perspective, both autoregressive (AR) and non-autoregressive (NAR) paradigms can be viewed as learning a probabilistic construction graph (Joshi et al. 2022; Goh et al. 2022).

The AR heuristics usually learn "fine-grained" construction graphs in the sense that, through forward passing of a neural network (NN) (e.g., attention mechanism using current context as query (Kool, van Hoof, and Welling 2019)), per-step construction is heavily conditioned on the obtained

Algorithm 1: Solving (sub-)TSP

**Input:** A TSP instance $tsp$ with $N$ nodes; trained Revisers and a list of their sizes $RS$ (e.g., $\{100, 50, 20\}$); the number of initial tours $W$; revision iterations $I_n, \forall n \in RS$
**Output:** The best solution $\boldsymbol{\pi}^*$
Generate the initial tours: $\{\boldsymbol{\pi}^1, \dots, \boldsymbol{\pi}^W\} \leftarrow RandomInsertion(tsp, W)$
**for** $n \in RS$ **do**
  Initialize the decomposition point: $p \leftarrow 0$
  Calculate the number of subtours decomposed from a tour: $K_n = \lfloor \frac{N}{n} \rfloor$
  **for** $iter = 1 \rightarrow I_n$ **do**
    $subtours \leftarrow \{\{\boldsymbol{\pi}^1_{p:p+n}, \dots, \boldsymbol{\pi}^1_{p+n(K_n-1):p+nK_n}\}, \dots, \{\boldsymbol{\pi}^W_{p:p+n}, \dots, \boldsymbol{\pi}^W_{p+n(K_n-1):p+nK_n}\}\}$
    Apply coordinate transformation and instance augmentation to $subtours$
    $subtours \leftarrow$ Reviser-$n(subtours, tsp)$
    $\{\boldsymbol{\pi}^1, \dots, \boldsymbol{\pi}^W\} \leftarrow Composition(subtours)$
    Shift the decomposition point: $p \leftarrow p + \max(1, \lfloor \frac{n}{I_n} \rfloor)$
  **end for**
**end for**
$\boldsymbol{\pi}^* = PickBest(tsp, \{\boldsymbol{\pi}^1, \dots, \boldsymbol{\pi}^W\})$

| | GLOP for TSP | | | | | GLOP (more revisions) for TSP | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $W$ | $I_{10}$ | $I_{20}$ | $I_{50}$ | $I_{100}$ | $W$ | $I_{10}$ | $I_{20}$ | $I_{50}$ | $I_{100}$ |
| TSP100 | 35 | 5 | 10 | 10 | 20 | 140 | 5 | 10 | 10 | 20 |
| TSP500 | 1 | - | 5 | 25 | 20 | 10 | - | 5 | 25 | 20 |
| TSP1K | 1 | - | 5 | 25 | 20 | 10 | - | 5 | 25 | 20 |
| TSP10K | 1 | - | 5 | 20 | 10 | 1 | - | 5 | 25 | 50 |
| TSP100K | 1 | - | 5 | 5 | 5 | 1 | - | 5 | 25 | 50 |

Table 9: Hyperparameter settings used for TSP inference.

| | $W$ | $I_{20}$ | $I_{50}$ |
|---|---|---|---|
| CVRP1K | 1 | 5 | - |
| CVRP2K | 1 | 5 | 5 |
| CVRP5K | 1 | 5 | - |
| CVRP7K | 1 | 5 | - |

Table 10: Hyperparameter settings used for CVRP inference.

partial solution. So it takes more meticulous actions by referring to the rich decoding context. However, it entails accurate context representation for effective actions and costly step-by-step neural decoding for sampling training trajectories, which hinder the scalability of AR heuristics (Joshi et al. 2022).

The NAR heuristics usually learn "coarse-grained" construction graphs only conditioned on the input problem instance. It enables construction graph (heatmap) generations in one shot and dramatically simplifies the context representation for solution decoding (e.g., decoding while simply masking the visited nodes). While being much more scalable, they underperform with vanilla sampling-based decoding due to the paucity of rich decoding context (Joshi, Laurent, and Bresson 2019; Qiu, Sun, and Yang 2022; Sun and Yang 2023). In this sense, those NAR heuristics need to incorporate additional solution refinements to trade off execution time for higher solution quality, making them less suitable for real-time applications.

## C Additional related work

The end-to-end NCO solvers can be categorized into two paradigms, i.e., autoregressive (AR) solution construction and non-autoregressive (NAR) heatmap generation coupled with subsequent decoding.

AR heuristics allow the neural models to output the assignments to decision variables sequentially, one step at a time (Vinyals, Fortunato, and Jaitly 2015; Bello et al. 2016; Kool, van Hoof, and Welling 2019; Ma et al. 2019; Kwon et al. 2020; Xin et al. 2021a; Hottung and Tierney 2019; Bresson and Laurent 2021; Choo et al. 2022; Kim, Park, and Park 2022; Zong et al. 2022b; Grinsztajn, Furelos-Blanco, and Barrett 2022; Zhang et al. 2022a; Jin et al. 2023). Learning global AR methods suffers from poor scaling-up generalization, as well as the prohibitive cost and ineffectiveness of training on large problems (Joshi et al. 2022). By contrast, NAR heuristics typically apply GNNs in one shot to output a heatmap predicting how promising each solution component is. Then, one can directly generate solutions with such a heatmap using greedy decoding, sampling, or beam search (Nowak et al. 2018; Joshi, Laurent, and Bresson 2019; Qiu, Sun, and Yang 2022; Sun and Yang 2023; Ye et al. 2023), which are all considered as vanilla sampling-based decoding strategies. Despite recent success in scaling NAR methods to large-scale problems (Fu, Qiu, and Zha 2021; Qiu, Sun,

and Yang 2022; Sun and Yang 2023; Min, Bai, and Gomes 2023), they underperform with vanilla sampling-based decoding and have to be coupled with iterative solution refinement such as Monte Carlo Tree Search (MCTS) (Fu, Qiu, and Zha 2021), falling short of real-time needs.

To the best of our knowledge, GLOP is the first neural solver to effectively integrate both AR and NAR components. Regarding TSP solution quality, GLOP is not in competition with the methods equipped with sophisticated and iterative improvement operators such as MCTS and LKH-3. Instead, it is a complement for scenarios requiring highly real-time solutions. Moreover, despite the recent preliminary endeavor to learn a cross-distribution TSP solver (Wang et al. 2021; Zhang et al. 2022b; Jiang et al. 2022; Bi et al. 2022; Wang et al. 2023; Zhou et al. 2023), GLOP performs SOTA consistency across TSP scales and distributions.

Less related to GLOP, some works explore hybridizing heatmap with other algorithms instead of providing end-to-end solutions. They includes utilizing heatmaps to assist MCTS (Fu, Qiu, and Zha 2021), Guided Local Search (Hudson et al. 2022), Guided Tree Search (Li, Chen, and Koltun 2018), LKH-3 (Xin et al. 2021b), and Dynamic Programming (Kool et al. 2022). Another line of methods learn repeated decision-making for algorithmic enhancement or solution refinement (Ma et al. 2021; Wu et al. 2021b; d O Costa et al. 2020; Sui et al. 2021; Zong et al. 2022a; Wu et al. 2021a; Lu, Zhang, and Yang 2020). While these methods can continuously improve the solution quality, they often lack scalability and fail to meet the requirements for real-time solutions. In addition, a surge of recent works broaden the applications of NCO techniques (Wu et al. 2023; Zong et al. 2022b; Chen et al. 2022; Fan et al. 2022; Song et al. 2023).

# D Extended evaluations

## D.1 Further comparison with baselines using MCTS

GLOP compares favorably with baselines that implement MCTS (Fu, Qiu, and Zha 2021; Qiu, Sun, and Yang 2022) when instant solutions are desired. Specifically, GLOP can produce reasonable solutions before the baselines obtain heatmaps, while the latter can eventually outperform GLOP due to the continued improvements provided by MCTS. The trends are shown in Fig. 2 using TSP10K for illustration.


Figure 2: Further comparison with two strong baselines that implement MCTS. The starting point of GLOP applies $W = 1$ and no augmentation. The curves of GCN+MCTS and DIMES start when they finish heatmap generation and the first MCTS iteration.

## D.2 Ablation study

**Neural heuristics against LKH-3 in local policy** We use neural heuristics as SHPP solvers for parallelizability and efficiency. Our local policies involve solving many SHPP instances, e.g., 32K SHPP20, 64K SHPP50, and 25.6K SHPP100 for 128 TSP1000, when using the hyperparameters (less revisions) in Table 9. Neural heuristics can solve thousands of them simultaneously and instantly in one batch, enabling parallelism and efficiency required for real-time solutions. We evaluate neural heuristics against LKH-3 in our local policy to validate our motivation, with results on TSP and PCTSP displayed in Table 11. Here, we replace the neural heuristics with LKH-3 (with the parameter 'runs' set to 1 to save time). Implementing neural heuristics shows clear advantage in terms of solution efficiency.

| | w/ LKH-3 | | w/ neural heuristics (GLOP) | |
|---|---|---|---|---|
| | Obj. | Time | Obj. | Time |
| TSP500 | 16.86 | 34m | 17.07 | 19s (×**107**) |
| TSP1K | 23.74 | 68m | 24.01 | 34s (×**120**) |
| TSP10K | 74.64 | 50m | 75.62 | 32s (×**95**) |
| PCTSP500 | 14.5 | 17m | 14.6 | 26s (×**39**) |
| PCTSP1K | 19.8 | 25m | 20.0 | 47s (×**32**) |
| PCTSP5K | 45.7 | 1.7h | 46.0 | 3.7m (×**27**) |

Table 11: Neural heuristics against LKH-3 as SHPP solver.

**Components in local policy** We conduct ablation studies of the components in our inference pipeline for (sub-)TSP, including bidirectional decoding (BD), coordinate transformation (CT), and two-stage curriculum learning (CL1 and CL2). The results in Table 12 validate the design of each component. Here, we use more revisions to realize the full potential of our local policies.

**Global policy** On CVRP, we demonstrate the advantage of our NAR global policy over AR dividing policy by comparing GLOP-G (LKH3) with TAM-LKH3 (Hou et al. 2023) in Table 6. On PCTSP, Table 13 further verifies the importance of our global policy by comparing it with random partition.

| Components | | | | TSP500 | TSP1K |
|---|---|---|---|---|---|
| BD | CT | CL1 | CL2 | | |
| × | | | | 16.88 | 23.85 |
| | × | | | 16.92 | 24.17 |
| | | × | | 16.91 | 23.89 |
| | | | × | 16.94 | 23.95 |
| | | | | **16.80** | **23.73** |

Table 12: Ablation studies of our local policy.

| Problem | GLOP-RP | GLOP-G |
|---|---|---|
| PCTSP500 | 19.0 | **14.6** |
| PCTSP1K | 26.1 | **20.0** |
| PCTSP5K | 53.0 | **46.0** |

Table 13: Ablation studies of our global policy.

### D.3 Discussing cross-distribution performance of local policies

**Training schemes** In Table 14, we investigate four training schemes: utilizing uniformly sampled SHPPs, only the 1st-stage curriculum, only the 2nd-stage curriculum, and the original scheme. Compared with the cross-distribution baseline (Zhang et al. 2022b), GLOP shows lower Det. under all settings. It verifies that the holistic solution process of dividing and conquering, instead of a particular training scheme, is the main contributor to the cross-distribution performance. By comparing GLOP with different training schemes, we also conclude that the 1st-stage curriculum enhances cross-distribution performance while the 2nd-stage curriculum makes GLOP more specialized in solving uniform TSP. The results should not come as a surprise: GLOP learns to solve SHPP with varied y-axis upper bound in the 1st-stage curriculum, and we sample uniform TSPs to construct the training datasets in the 2nd-stage curriculum.

**More revisers and more revisions** Intuitively, it may make GLOP more distribution-invariant to implement various local policies with different specialties and perform more revisions by decomposing TSP tours differently. However, Table 15 demonstrates that more revisers or more revisions can effectively reduce "Gap" but have less impact on "Det.". We further conclude that the holistic solution process of dividing and conquering, instead of the implementation of multiple revisers or revisions, is the main contributor to the cross-distribution performance.

### D.4 Stability analysis of GLOP

Figure 3 showcases the stability of GLOP on TSP500 and TSP1K, illustrating the box plots of the objective values for 10 independent runs. Each subplot depicts the minimum, lower quartile, mean, upper quartile, maximum, and possible outliers. The results suggest that implementing more revisions effectively guarantees better stability. The ranges of the box plots for GLOP (more revisions) are within 0.02, indicating desirable stability.

### D.5 Discussing time complexity

In GLOP, a revision decomposes a TSP into a batch of SHPPs and exploits NN to solve them simultaneously. Since the scale of SHPP is a constant for an arbitrary TSP and solving SHPPs is parallelizable, a revision has constant time complexity. Table 16 presents the time needed for solving a single instance of different scales but under identical settings (50 and 20 revisions for Reviser-50, and 20, respectively). It shows a near-constant empirical complexity when the time of revisions dominates.

For CVRP/PCTSP, GLOP shows a (lower-than-)linear empirical complexity, as presented in Table 6 and 8, because the partition part has linear complexity due to graph sparsification and solving sub-TSPs has near-constant empirical complexity as discussed above.

## E Implementation details of baselines

Our experiments involve the following neural baselines:

- **TSP.** AM (Kool, van Hoof, and Welling 2019), LCP (Kim, Park, and Kim 2021), GCN+MCTS (Fu, Qiu, and Zha 2021), POMO-EAS (Hottung, Kwon, and Tierney 2022), DIMES (Qiu, Sun, and Yang 2022), Tspformer (Yang et al. 2023), H-TSP (Pan et al. 2023), Pointerformer (Jin et al. 2023), DACT (Ma et al. 2021), AMDKD+EAS (Bi et al. 2022), AM+HAC (Zhang et al. 2022b), and MatNet (Kwon et al. 2021).
- **CVRP.** AM (Kool, van Hoof, and Welling 2019), L2I (Lu, Zhang, and Yang 2020), NLNS (Hottung and Tierney 2019), L2D (Li, Yan, and Wu 2021), RBG (Zong et al. 2022a), and TAM (Hou et al. 2023).
- **PCTSP.** AM (Kool, van Hoof, and Welling 2019) and MDAM (Xin et al. 2021a).

Most baselines are reproduced by strictly following their open-sourced implementation. However, some others have to be tuned and adapted for proper and comparable evaluations.

**Non-learning baselines** Most results of non-learning baselines are reproduced following Kool, van Hoof, and Welling (2019). One exception is that we set the number of trails to 1 for LKH-3 on TSP100K to reduce runtime.

**AM (Kool, van Hoof, and Welling 2019)** The checkpoints trained on TSP / PCTSP100 are generalized for comparison. We strictly follow the original implementation using sampling decoding.

**POMO-EAS (Hottung, Kwon, and Tierney 2022)** We implement EAS-Tab using the POMO checkpoint trained on TSP100 and set $T = 60$.

**DACT (Ma et al. 2021)** For the TSPLIB benchmarks, we apply the model trained on TSP20 for instances with $< 50$ nodes, the model trained on TSP50 for those with $50 \leq n < 100$ nodes, and the model trained on TSP100 for the rest, all with $T = 1000$ and without augmentation.

**GCN+MCTS (Fu, Qiu, and Zha 2021)** We implement the CPU-version MCTS. The parameter $T$ that controls MCTS execution time is set smaller for a fair comparison. Note that $T$ can be directly derived from the total duration reported for MCTS. For example, on TSP500, it takes 33 seconds to complete MCTS for 128 instances on 12 CPU cores using multiprocessing (a single thread for each instance). Therefore, the time limit $T$ can be calculated as: 33s $\div$ 128 $\times$ 12. On TSPLIB benchmarks, we use the original TSP20 parameter settings for TSP20-49 instances, TSP50 for TSP50-99 instances, TSP100 for TSP100-149 instances, TSP200 for TSP150-349 instances, TSP500 for TSP350-749 instances, TSP1000 otherwise.

| Method $\{W = 40\}$ | Uniform Gap(%) | Expansion Gap(%) | Expansion Det.(%) | Explosion Gap(%) | Explosion Det.(%) | Implosion Gap(%) | Implosion Det.(%) | Avg. Gap(%) | Avg. Det.(%) |
|---|---|---|---|---|---|---|---|---|---|
| AM+HAC $\{M = 1280\}$ | 2.484 | 3.997 | 61 | 3.084 | 24 | 2.595 | 4.5 | 3.040 | 30 |
| GLOP (Uniform) | 0.215 | 0.235 | 9.3 | 0.119 | -44 | 0.197 | -8.4 | 0.192 | -14 |
| GLOP (C1) | 0.183 | 0.189 | 3.3 | 0.089 | -51 | 0.162 | -11 | 0.156 | -20 |
| GLOP (C2) | 0.180 | 0.260 | 44 | 0.105 | -42 | 0.159 | -12 | 0.176 | -3 |
| GLOP (C1 + C2) | 0.091 | 0.166 | 82 | 0.066 | -27 | 0.082 | -9.9 | 0.101 | 15 |

Table 14: The impact of different training schemes on the cross-distribution performance of GLOP. GLOP (Uniform), GLOP (C1), and GLOP (C2) are trained with uniformly sampled SHPPs, the 1st-stage curriculum, and the 2nd-stage curriculum, respectively.

| Method $\{W = 40\}$ | Uniform Gap(%) | Expansion Gap(%) | Expansion Det.(%) | Explosion Gap(%) | Explosion Det.(%) | Implosion Gap(%) | Implosion Det.(%) | Avg. Gap(%) | Avg. Det.(%) |
|---|---|---|---|---|---|---|---|---|---|
| AM+HAC $\{M = 1280\}$ | 2.484 | 3.997 | 61 | 3.084 | 24 | 2.595 | 4.5 | 3.040 | 30 |
| GLOP ($RS = \{50\}$) | 0.380 | 0.740 | 94 | 0.285 | -25 | 0.362 | -4.8 | 0.442 | 22 |
| GLOP ($I_n = 1$) | 0.738 | 1.194 | 62 | 0.621 | -16 | 0.693 | -6.1 | 0.811 | 13 |
| GLOP | 0.091 | 0.166 | 82 | 0.066 | -27 | 0.082 | -9.9 | 0.101 | 15 |

Table 15: The impact of more revisers or more revisions on the cross-distribution performance of GLOP. GLOP ($RS = \{50\}$) implements Reviser-50 alone; GLOP ($I_n = 1$) implements a single revision for all revisers.

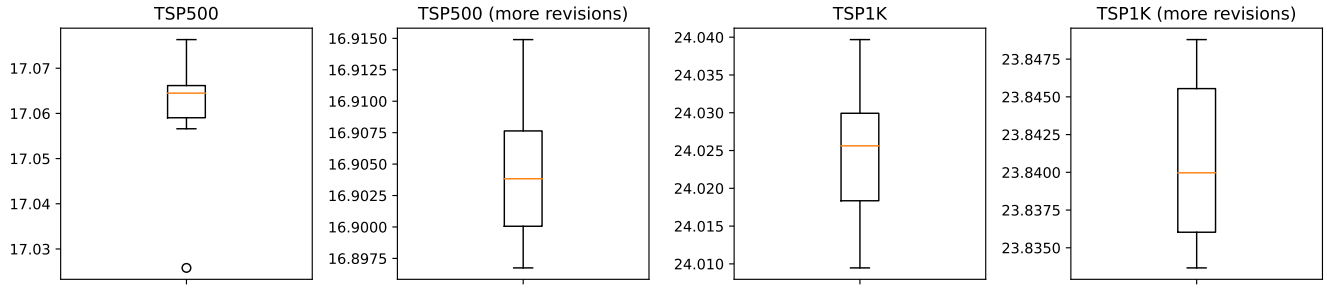


Figure 3: Box plots of the objective values obtained by GLOP for 10 independent runs of 128 test instances (with random seeds 0-9).

**DIMES (Qiu, Sun, and Yang 2022)** We follow the original implementations for TSP500, 1000, and 10000, except that we adjust the MCTS execution time $T$ to obtain comparable results. We generalize the model trained on TSP10K to TSP100K. For TSPLIB benchmarks, we additionally train a model on TSP50 and TSP200. We apply the TSP50 model and parameter settings for TSP20-199 instances, TSP200 for TSP200-349 instances, TSP500 for TSP350-749 instances, and TSP1000 otherwise.

**AM+HAC (Zhang et al. 2022b)** We fine-tune the AM trained on TSP100 with 100 epochs of Hardness-adaptive Curriculum (HAC) and set $M = 1280$ for all evaluations.

**AMDKD+EAS (Bi et al. 2022)** We set the EAS iterations $T$ to 100.

**L2D (Li, Yan, and Wu 2021) and RBG (Zong et al. 2022a)** L2D and RBG adopt different CVRP settings: both set vehicle capacity to 50 for all scales, and RBG additionally fixes the depot to the center of the unit square. Their provided checkpoints are directly adapted for the evaluation in this work.

**NLNS (Hottung and Tierney 2019)** We follow the original implementation of single-instance search and use the checkpoint trained on set XE_17.

**MDAM (Xin et al. 2021a)** The checkpoint trained on PCTSP100 is generalized for comparison.

## F Discussing connections with non-learning OR methods

NCO solvers show promise due to their parallelizability and data-driven nature in design automation, and they are gradually closing the gap with traditional OR methods. Following previous NCO works, GLOP serves as a versatile framework that effectively extends small neural solvers to much larger problems. As GLOP is founded on existing neural solvers, it

| Scale | TSP200 | TSP500 | TSP1K | TSP2K | TSP5K | TSP10K |
|---|---|---|---|---|---|---|
| Time (s) | 6.3 | 6.3 | 6.9 | 7.3 | 7.8 | 8.6 |

Table 16: Time needed for solving a single instance of different scales.

stands to continually benefit from the rapid development of NCO models and increases in GPU power.

The idea of solving routing problems hierarchically has long been utilized in non-learning OR methods (Zhang et al. 2021; Alesiani, Ermis, and Gkiotsalitis 2022; Xiao et al. 2019; Taillard and Helsgaun 2019). A seminal work of this kind is POPMUSIC for TSP (Taillard and Helsgaun 2019). Our (Sub-)TSP solver, as well as other hierarchical TSP solvers (Pan et al. 2023; Cheng et al. 2023; Kim, Park, and Kim 2021), shares the ideology of POPMUSIC. Nevertheless, we discuss that NCO solvers and POPMUSIC have complementary natures. Employing the sophisticated initialization of POPMUSIC can improve the performance of NCO solvers. On the other hand, if replacing LKH with NCO solvers, POPMUSIC can gain from the high parallelizability of NCO at little cost of solution quality, given that current NCO solvers can solve small problems near-optimally.