Large Language Models powered Neural Solvers for Vehicle Routing Problems

Cong Dao Tran[1], Quan Nguyen-Tri[1], Huynh Thi Thanh Binh[2], Hoang Thanh-Tung[3]

[1]*FPT Software AI Center,*
[2]*Hanoi University of Science and Technology,*
[3]*University of Engineering and Technology, Vietnam National University Hanoi*

April 2025

Content

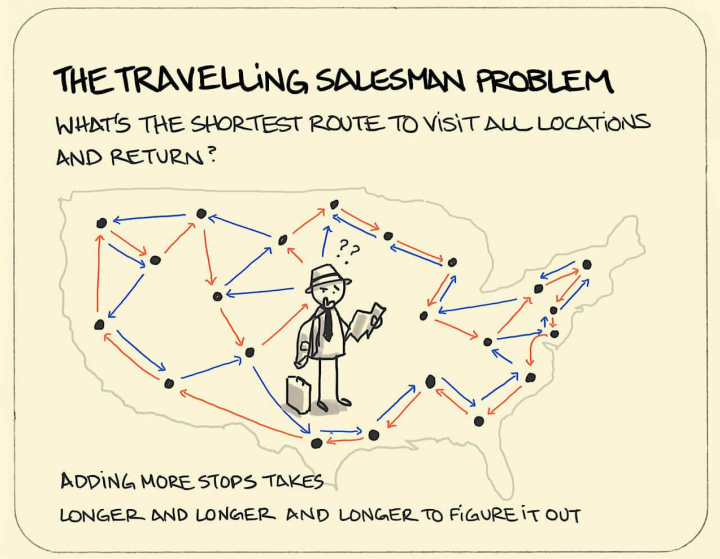
1. Background
2. Related work
3. Methodology
4. Experiment and Result
5. Conclusion

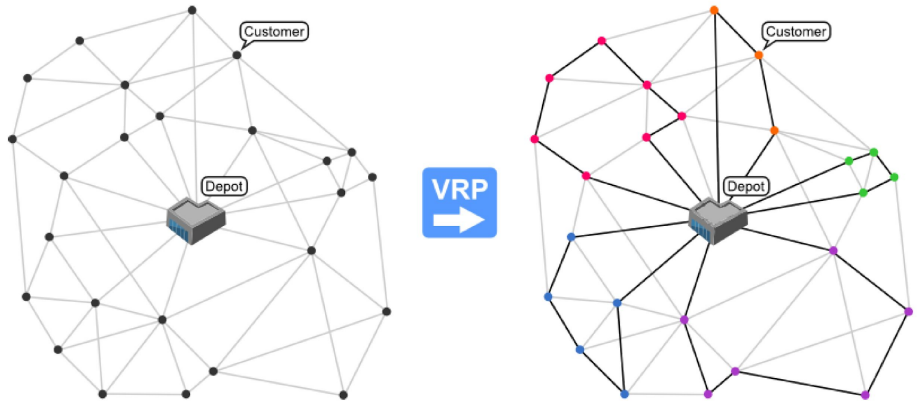
Content

1. Background
2. Related work
3. Methodology
4. Experiment and Result
5. Conclusion

Vehicle routing problems (VRPs)

- VRPs are fundamental NP-hard Combinatorial Optimization Problems, essential in logistics, public transportation, and operations research.

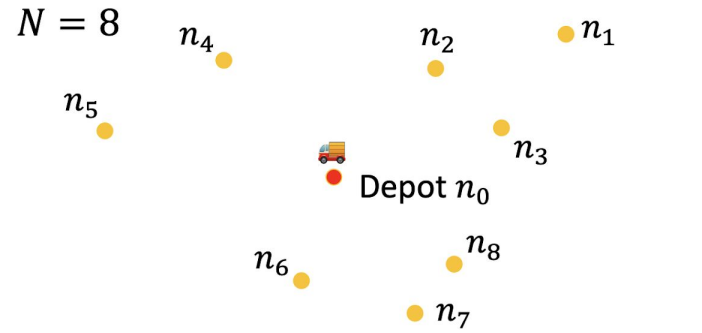




- Due to their complexity, solving VRPs becomes increasingly difficult and time-consuming as the problem size grows, which poses significant challenges for traditional exact and heuristic algorithms.

VRP: Problem formulation

- A VRP instance is characterized by a node set V containing n nodes, where each node is denoted as $n_i \in V$.
- The optimal solution for a VRP is the tour $\pi^* = (\pi_1, \pi_2, \ldots, \pi_l)$ that visits all nodes while minimizing the overall distance.

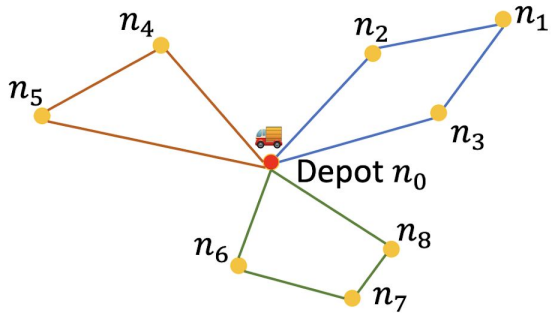


Vehicle capacity: Q Problem scale: N

Node info: $\{n_i = (x_i, y_i, d_i), i = 1,2,..,N\}$

coordinates demand

(a)



$\{n_0, n_2, n_1, n_3, n_0, n_4, n_5, n_0, n_6, n_7, n_8, n_0\}$

route 1 route 2 route 3

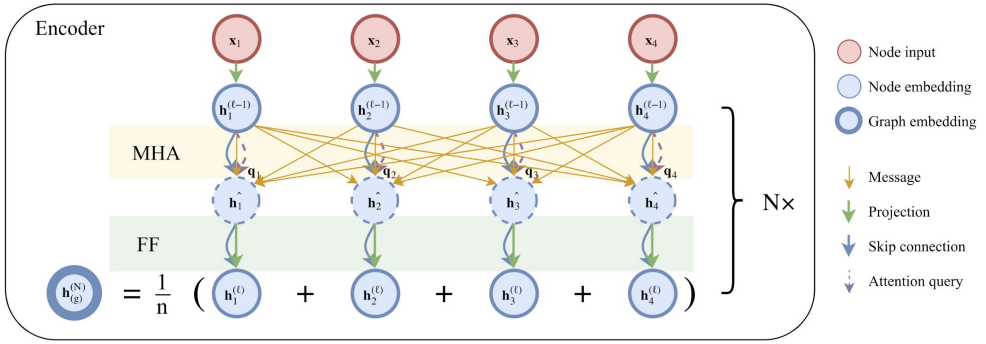
(b)

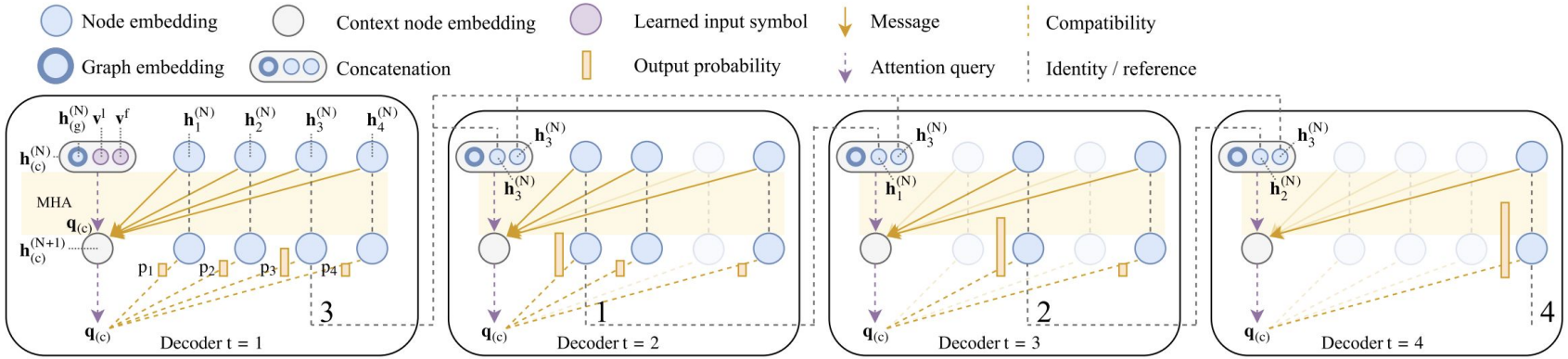
Neural Combinatorial Optimization (NCO)

- **Constructive neural solvers** are commonly employed for solving VRPs using a Markov Decision Process (MDP) and incorporate a **Transformer-based encoder-decoder** framework as their policy network.
- **Encoder** captures node features, while the **Decoder** generates a tour π based on the extracted features and the action history of node selections.
- Constructive NCO process can be factorized into a chain of conditional probabilities as:

$$\mathbf{p}\left(\pi|\mathbf{s},\theta\right)=\prod_{t=1}^{l}\mathbf{p}\left(\pi_t|\pi_{1:t-1},\mathbf{s},\theta\right),$$

where $\mathbf{p}$ represents the policy parameterized by θ, and l represents the number of actions taken to complete the tour.





Content

1. Background
2. Related work
3. Methodology
4. Experiment and Result
5. Conclusion

Neural Combinatorial Optimization (NCO)

Method	Ideas
POMO [1]	A strong baseline NCO model based on Attention model with heavy encoder and light decoder.
LEHD[2]	An NCO model with light encoder and heavy decoder for robust large-scale generalization of VRPs.
ELG [3]	Effectively balances exploration and exploitation by combining local topological strategies based on expert knowledge with guidance from neural networks, making it adaptable to different scales of VRPs.
FunSearch[4], EoH [5]	Translating high-level heuristic ideas, referred to as “thoughts” into executable code via LLMs, effectively integrating the strengths of both LLMs and EC.

[1] Kwon et al. *POMO: Policy optimization with multiple optima for reinforcement learning*. In Advances in Neural Information Processing Systems 33 (**NeurIPS 2021)**

[2] Luo et al. *Neural Combinatorial Optimization with Heavy Decoder: Toward Large Scale Generalization.* In Advances in Neural Information Processing Systems 36 (**NeurIPS 2023**)

[3] Gao et al. *Towards Generalizable Neural Solvers for Vehicle Routing Problems via Ensemble with Transferrable Local Policy*. In Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (**IJCAI 2024)**

[4] Romera-Paredes, Bernardino, et al. *Mathematical discoveries from program search with large language models.* **Nature 2024**

[5] Liu, F. et al. *Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model*. In Forty-first International Conference on Machine Learning (**ICML 2024**).

Our contributions

- Propose a new **LLM-guided attention bias** for model fine-tuning to enhance the generalization capabilities of neural combinatorial optimization models. This attention bias is **automatically designed by LLMs**, which are used to **augment any attention-based NCO models** for better performance on large-scale problems.
- Develop an **efficient fine-tuning** process that leverages training on instances of varying sizes. This method improves the **model's flexibility** and **solution quality** by incorporating LLM-generated attention bias, allowing **effective generalization** to larger and more diverse problem instances without altering their architecture.
- Our proposed method can achieve **state-of-the-art performance** in solving the TSP and CVRP across various scales and also generalizes well to solve real-world TSPLib/CVRPLib problems.

Content

1. Background
2. Related work
3. Methodology
4. Experiment and Result
5. Conclusion

Attention Bias for Constructive Neural Solver

- Given a problem instance s with n node features $(s_1, \ldots, s_n)$ (e.g., the coordinates of n nodes for TSP and additional demands for CVRP)
- **Encoder** transforms each node feature s_i to its initial embedding $\mathbf{h}_i^{(0)}$ through a learned linear projection.
- Then the initial embeddings $\left\{\mathbf{h}_1^{(0)}, \ldots, \mathbf{h}_n^{(0)}\right\}$ are fed into L attention layers to get the final node embedding matrix $H^{(L)} = (\mathbf{h}_1^{(L)}, \ldots, \mathbf{h}_n^{(L)}$ with the embedding process: $H^{(l)} = AttentionLayer(H^{(l-1)})$
- **Decoder** computes ***attention score*** as compatibility of the query and keys with all nodes, i.e.,

$$\mathbf{u} = \frac{\boldsymbol{q}^{\mathrm{T}}\mathbf{K}}{\sqrt{d}}, \boldsymbol{q} \in \mathbb{R}^{d\times 1}, \mathbf{K} \in \mathbb{R}^{d\times n},$$

- where $\boldsymbol{q}$ represents the query vector, $\mathbf{K}$ represents the matrix of keys, and d is the embedding dimension. The query comes from the context embedding, and the keys come from node embeddings.

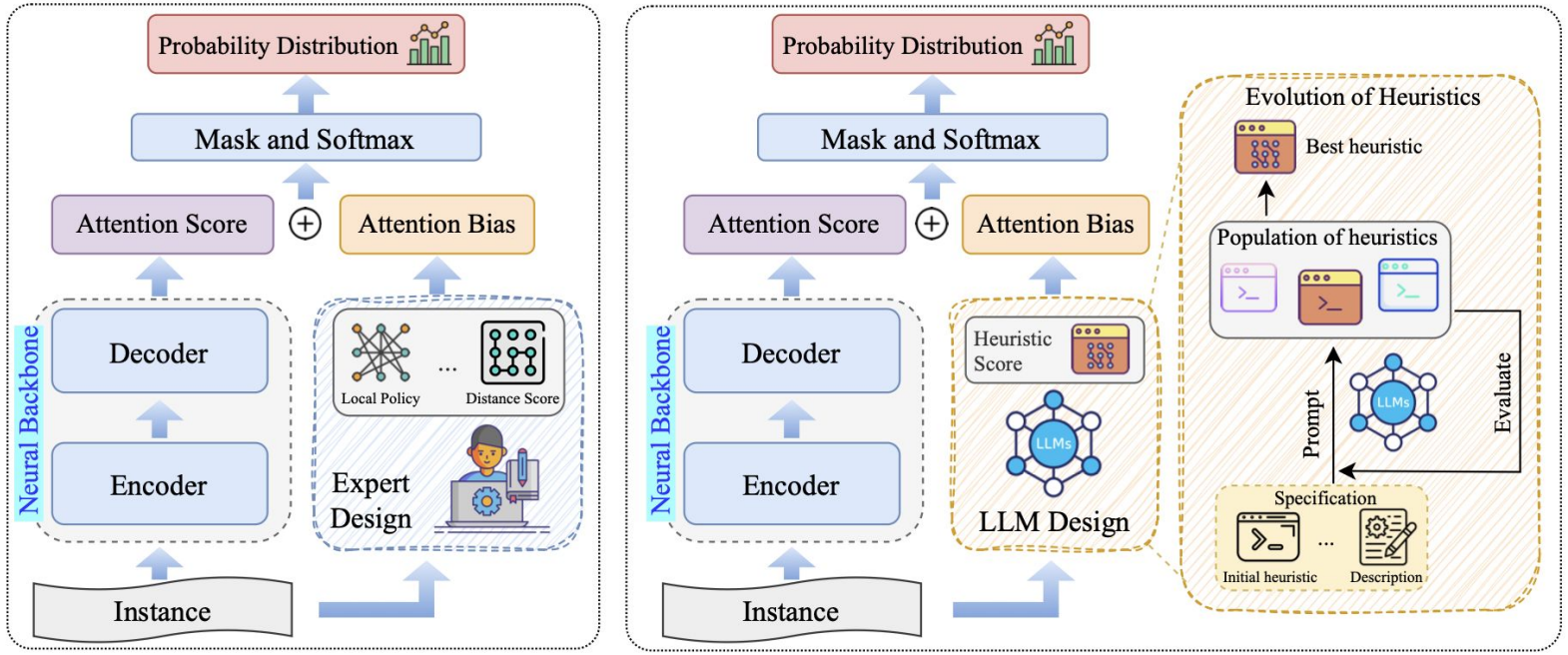
Attention Bias for Constructive Neural Solver

- We introduce a generic concept of ***attention bias***, which can be represented by vector $\hat{\mathbf{u}}$.
- At each decoding step t, the most suitable node is selected and added to the current solution based on the selection probability:

$$\mathbf{p}^t = \mathbf{p}(\pi_t | \pi_{1:t-1}, \theta) = \text{softmax}(\text{mask}(\mathbf{u} + \hat{\mathbf{u}})) \qquad (2)$$

- The ***attention bias*** can be **aggregated** with the model's ***attention score*** $\mathbf{u}$ to adjust the probability selection of nodes in constructing the solution.

Attention Bias via LLM design



(a) Attention bias via Human Design

(b) Attention bias via LLM Design (Ours)

Figure 1: The schematic diagrams of generalizable neural solvers adapted with attention bias designed by humans and LLMs.

- To effectively design attention bias, we harness the capabilities of LLMs with evolutionary search to **generate** and **refine** attention bias **automatically**.

Attention Bias via LLM design

- We initially examine a heuristic function $h(M, s)$ that takes the distance matrix M of n nodes of a problem instance s as input and outputs a matrix U representing the attention bias of n nodes, formally $U = h(M, s)$.
- **Heuristic representation:** each heuristic function h consists of three components:

1) *a natural language description,*
2) *a code block in a predefined format*
3) *a fitness value.*

Function description: The new algorithm computes the heuristics matrix by calculating a normalized version of the distances, where closer distances have higher positive values and farther distances yield negative values, thus assigning promising edges higher values and undesirable edges lower values.

Code:

```
def heuristic(distance_matrix):
    dist_tensor = torch.tensor(distance_matrix, dtype=torch.float32)
    # Calculate the maximum distance for each node (row-wise maximum)
    max_distances = dist_tensor.max(dim=1)[0]
    # Calculate contribution scores for each edge
    heuristics_matrix = max_distances.unsqueeze(1) - dist_tensor
    return heuristics_matrix
```

Fitness value: 10.74865

Figure 2: An example of a heuristic representation.

Attention Bias via LLM design

- **Population initialization:** we initialize a population P of N heuristics by prompting LLMs using *Initialization prompt*.
- **Evolution process: w**e perform the evolutionary process to find the best heuristic by repeating the following G iterations:
  - Select parent heuristics from the current population to create offspring heuristics.
  - Request LLM to produce a novel heuristic along with its related code implementation. The offspring heuristics are generated by using five evolution prompt strategies as exploration and modification operators.
  - The new heuristics are then evaluated on a set of evaluation instances to determine their fitness value.
  - Add the new heuristic to the current population if both the heuristic and its code are valid.
  - Select the top N heuristics from the current population to form the new population for the next generation.

Attention Bias via LLM design

Initialization Prompt for TSP

I need assistance in designing a new heuristic to improve the solution of the Traveling Salesman Problem (TSP) by incorporating insights from prior heuristics. The TSP requires finding the shortest path that visits all given nodes and returns to the starting node. The new algorithm computes a heuristic matrix where the distances between nodes are normalized. In this matrix, closer distances are assigned higher positive values, while farther distances are given negative values. This approach aims to prioritize promising edges by assigning them higher scores and to de-prioritize less desirable edges by giving them lower scores.
Please design a new heuristic.
First, describe your new heuristic and main steps in one sentence. The description must be inside a brace.
Next, implement it in Pytorch as a function named 'heuristic'. This function should accept one input: 'distance matrix'. The function should return one output: 'heuristic matrix'.
Note that 'distance matrix' is a tensor, and 'heuristic matrix' is a tensor with the same shape as 'distance matrix.' All are Torch tensors. Do not give additional explanations.

Initialization Prompt for CVRP

I need assistance in designing a new heuristic to improve the solution of the Capacitated Vehicle Routing Problem (CVRP) by incorporating insights from prior heuristics. The CVRP requires finding the shortest path that visits all given nodes and returns to the starting node, with the additional constraint that each node has a demand and each vehicle has a capacity. The total demand of the nodes visited by a vehicle cannot exceed its capacity. If the total demand exceeds the vehicle's capacity, the vehicle must return to the starting node before continuing.
Please design a new heuristic.
First, describe your new heuristic and main steps in one sentence. The description must be inside a brace.
Next, implement it in Pytorch as a function named 'heuristic'. This function should accept two inputs: 'distance matrix' and 'node_demands'. The function should return one output: 'heuristic matrix'.
Note that 'distance matrix' and 'node_demands' are tensors, and 'heuristic matrix' is a tensor with the same shape as 'distance matrix.' All are Torch tensors. Do not give additional explanations.

Initialization prompt

Evolution Prompt for E1:

I need assistance in designing a new heuristic to improve the solution of the Traveling Salesman Problem (TSP) by incorporating insights from prior heuristics. The TSP requires finding the shortest path that visits all given nodes and returns to the starting node. The new algorithm computes a heuristic matrix where the distances between nodes are normalized. In this matrix, closer distances are assigned higher positive values, while farther distances are given negative values. This approach aims to prioritize promising edges by assigning them higher scores and to de-prioritize less desirable edges by giving them lower scores.
I have five existing algorithms with their codes as follows:
No.1 Heuristic description:
and the corresponding code are:
...
No.5 Heuristic description:
and the corresponding code are:

Please help me create a new algorithm that has a totally different form from the given ones.
First, describe your new heuristic and main steps in one sentence. The description must be inside a brace.
Next, implement it in Pytorch as a function named 'heuristic'. This function should accept one input: 'distance matrix'. The function should return one output: 'heuristic matrix'.
Note that 'distance matrix' is a tensor, and 'heuristic matrix' is a tensor with the same shape as 'distance matrix.' All are Torch tensors. Do not give additional explanations.

Evolution Prompt for M1:

I need assistance in designing a new heuristic to improve the solution of the Traveling Salesman Problem (TSP) by incorporating insights from prior heuristics. The TSP requires finding the shortest path that visits all given nodes and returns to the starting node. The new algorithm computes a heuristic matrix where the distances between nodes are normalized. In this matrix, closer distances are assigned higher positive values, while farther distances are given negative values. This approach aims to prioritize promising edges by assigning them higher scores and to de-prioritize less desirable edges by giving them lower scores.
I have one algorithm with its code as follows:
Algorithm description:
Code:

Please assist me in creating a new algorithm that has a different form but can be a modified version of the algorithm provided.
First, describe your new algorithm and main steps in one sentence. The description must be inside a brace.
Next, implement it in Pytorch as a function named 'heuristic'. This function should accept one input: 'distance matrix'. The function should return one output: 'heuristic matrix'.
Note that 'distance matrix' is a tensor, and 'heuristic matrix' is a tensor with the same shape as 'distance matrix.' All are Torch tensors. Do not give additional explanations.

Evolution prompt

Model fine-tuning

- After generating attention bias from LLM, we update vector $\mathbf{p}$ based on Equation (2) in each feed-forward.
- We **modify loss function** of the neural models and then **fine-tune** the model by training on a small amount of data.
- The loss function is defined as the expectation of the cost $L(\pi)$ (total tour length) **normalized by the number of nodes**, i.e.,

$$\mathcal{L}(\theta|\mathbf{s}) = \mathbb{E}_{\mathbf{p}(\pi|\mathbf{s},\theta)}[\frac{L(\pi)}{n}] = \mathbb{E}_{\mathbf{p}(\pi|\mathbf{s},\theta)}[-R(\pi)], \quad (3)$$

where $R(\pi) = -\frac{L(\pi)}{n}$ is the total reward of solution π.

Algorithm 1 Fine-tuning NCO model

Input: Pre-trained policy network parameter θ on training set of 100 nodes, number of epochs E, steps per epoch T, batch size B, training set $\{S_n\}$, $(n \in \mathbb{N}, n \in [100, 200])$
Output: Updated policy network θ

for $epoch = 1, \ldots, E$ **do**
  $n \leftarrow \text{Random\_Integer}([100, 200])$
  // Training model on S_n with mini-batch
  for $step = 1, \ldots, T$ **do**
    $\nabla_\theta \mathcal{L}(\theta) \leftarrow \frac{1}{B} \sum_{i=1}^{B} \nabla_\theta \mathcal{L}(\theta|\mathbf{s}_i \in S_n)$
    $\theta \leftarrow \text{Adam}(\theta, \nabla_\theta \mathcal{L}(\theta))$
  end for
end for
return updated policy θ

Model fine-tuning

- We utilize the REINFORCE (Williams 1992) algorithm with shared baseline to estimate the gradient of the loss:

$$\nabla_\theta \mathcal{L}(\theta|\mathbf{s}) \approx -\frac{1}{N}\sum_{i=1}^{N}\left(R\left(\pi^i\right) - b(\mathbf{s})\right)\nabla_\theta \log \mathbf{p}\left(\pi^i|\mathbf{s},\theta\right), \tag{4}$$

where

$$\mathbf{p}\left(\pi^i|\mathbf{s},\theta\right) = \prod_{t=2}^{n}\mathbf{p}^t(\pi_t^i|\pi_{1:t-1}^i,\theta), \forall i \in [1,N]$$

$$b(\mathbf{s}) = \frac{1}{N}\sum_{j=1}^{N} R(\pi^j)$$

Content

1. Background
2. Related work
3. Methodology
4. Experiment and Result
5. Conclusion

Main results

Table 1: Experimental results on TSP and CVRP on the generated instances. The asterisk (*) denotes the results of methods directly obtained from the original paper. The best results obtained by neural solvers are in **bold**.

Method	TSP200	Time	TSP500	Time	TSP1000	Time	CVRP200	Time	CVRP500	Time	CVRP1000	Time
LKH/LKH3	0.000%	4m	0.000%	22m	0.000%	2.4h	0.000%	22m	0.000%	1.8h	0.000%	2.1h
Concorde/HGS	0.000%	2m	0.000%	22m	0.000%	2.3h	-1.126%	17m	-1.794%	1.3h	-2.162%	1.8h
OR-Tools	3.618%	10m	4.682%	38m	4.885%	2.9h	6.894%	15m	9.112%	38m	11.662%	50m
Att-GCN+MCTS*	0.884%	2m	2.536%	6m	3.223%	13m	-	-	-	-	-	-
MDAM bs50	1.996%	2m	10.065%	3m	20.375%	20m	4.304%	1m	10.498%	4m	27.814%	18m
POMO no-aug	2.394%	<1s	25.090%	2s	43.623%	6s	6.097%	<1s	30.164%	2.5s	144.664%	4s
POMO augx8	1.622%	2s	23.093%	10s	41.810%	0.6m	5.022%	2.5s	20.377%	10s	128.885%	0.8m
BQ greedy	0.895%	2s	1.834%	8s	3.965%	0.5m	3.527%	2s	5.121%	8s	9.812%	20s
BQ bs16	0.224%	1m	0.896%	3m	2.605%	12.8m	1.141%	0.6m	2.991%	2.8m	7.784%	13.7m
LEHD greedy	0.859%	1.7s	1.560%	0.2m	3.168%	1.3m	3.312%	2.4s	3.178%	0.23m	4.912%	1.34m
LEHD RRC50	0.123%	2.5m	0.482%	9.6m	1.416%	27m	0.515%	0.6m	0.930%	3.63m	2.814%	16.4m
ELG no-aug	2.722%	1.1s	8.390%	4.8s	12.413%	0.17m	2.605%	5.6s	7.232%	0.16m	14.401%	0.44m
ELG augx8	1.408%	4.2s	7.216%	0.4m	11.372%	1.2m	1.660%	9s	6.036%	0.5m	11.969%	1.8m
POMO+ReEvo*	4.02%	-	24.32%	-	29.08%	-	10.48%	-	25.7%	-	218.22%	-
LEHD+ReEvo*	0.74%	-	1.55%	-	2.97%	-	3.30%	-	2.94%	-	4.76%	-
POMO-LLM no-aug	1.364%	<1s	4.219%	3s	9.794%	7s	2.379%	1s	3.132%	2.7s	11.599%	4.2s
+ augx8	0.888%	2.4s	3.479%	0.3m	9.003%	0.8m	1.665%	2.6s	2.599%	10s	9.314%	0.8m
LEHD-LLM greedy	0.766%	1.8s	1.375%	0.2m	2.427%	1.3m	2.727%	2.4s	1.946%	0.24m	3.456%	1.33m
+ RRC50	**0.098%**	2.7m	**0.368%**	9.8m	**1.023%**	28m	**0.307%**	0.6m	**0.194%**	3.65m	**1.399%**	16.4m

Results on TSPLib and CVRPLib

Table 2: Experimental results on TSPLib.

Method	[100-200)	[200-500)	[500-1k]	[1k-5k)
POMO no-aug	3.74%	11.46%	22.57%	40.32%
POMO augx8	2.49%	9.57%	20.42%	36.85%
BQ greedy	2.68%	3.18%	8.31%	42.57%
BQ bs16	1.32%	2.18%	5.52%	36.73%
LEHD greedy	2.37%	2.64%	5.23%	11.55%
LEHD RRC50	0.48%	0.79%	2.17%	6.48%
ELG no-aug	2.54%	6.93%	9.24%	12.74%
ELG augx8	1.30%	3.88%	8.73%	11.33%
POMO-LLM no-aug	3.45%	6.33%	9.17%	13.12%
POMO-LLM augx8	2.12%	4.19%	8.02%	12.18%
LEHD-LLM greedy	2.33%	3.08%	3.41%	9.53%
LEHD-LLM RCC50	**0.32%**	**0.63%**	**1.83%**	**5.56%**

Table 3: Experimental results on CVRPLib on Set-X dataset.

Method	[100-200)	[200-500)	[500-1k]
POMO no-aug	9.743%	19.164%	57.227%
POMO augx8	6.889%	15.029%	40.904%
LEHD greedy	11.345%	9.452%	17.737%
LEHD RRC50	4.908%	4.732%	8.273%
ELG no-aug	6.253%	7.576%	10.024%
ELG augx8	**4.510%**	5.524%	7.805%
POMO-LLM no-aug	8.600%	10.604%	23.154%
POMO-LLM augx8	5.948%	8.507%	15.256%
LEHD-LLM greedy	10.922%	9.393%	16.439%
LEHD-LLM RCC50	4.839%	**4.624%**	**7.678%**

Table 4: Experimental results on CVRPLib on Set-XXL dataset.

Method	Leuven1	Leuven2	Antwerp1	Antwerp2
LEHD greedy	16.60%	34.85%	14.66%	22.77%
LEHD RRC50	11.98%	28.46%	10.78%	18.16%
GLOP	16.90%	21.80%	20.30%	19.40%
GLOP-LKH3	16.60%	**21.10%**	19.30%	19.40%
ELG no-aug	16.45%	23.26%	13.65%	26.13%
ELG augx8	12.26%	21.60%	12.26%	17.74%
LEHD-LLM greedy	15.12%	30.35%	13.61%	17.77%
LEHD-LLM RCC50	**11.68%**	25.97%	**9.64%**	**14.85%**

Ablation study

Effects of attention bias

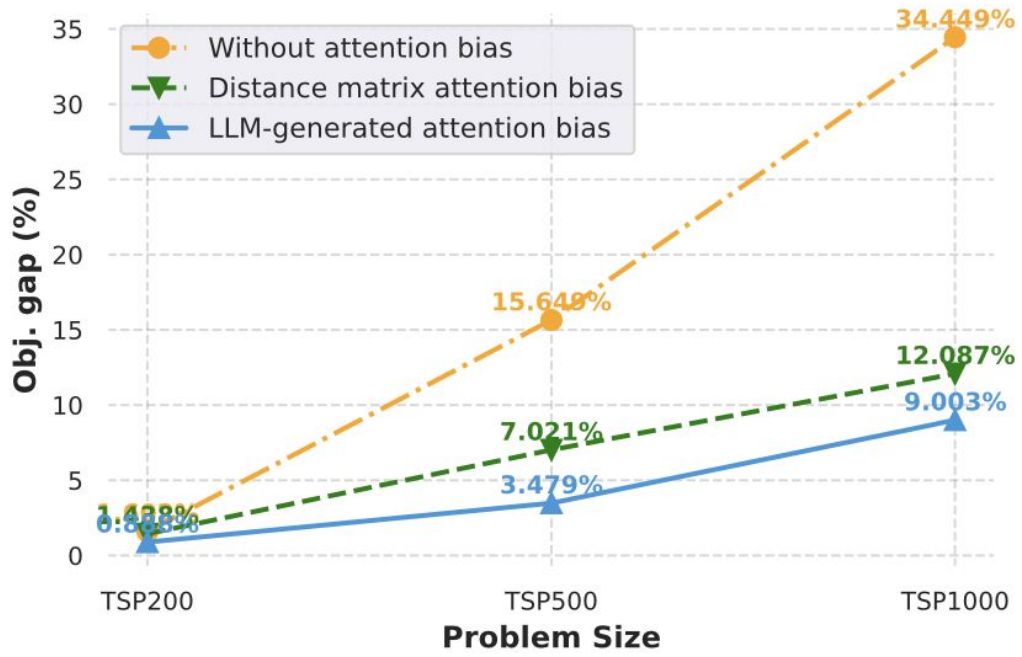


Figure 3: Performance of fine-tuning POMO with different attention biases and without any on the TSP test instances.

Effects of fine-tuning strategy

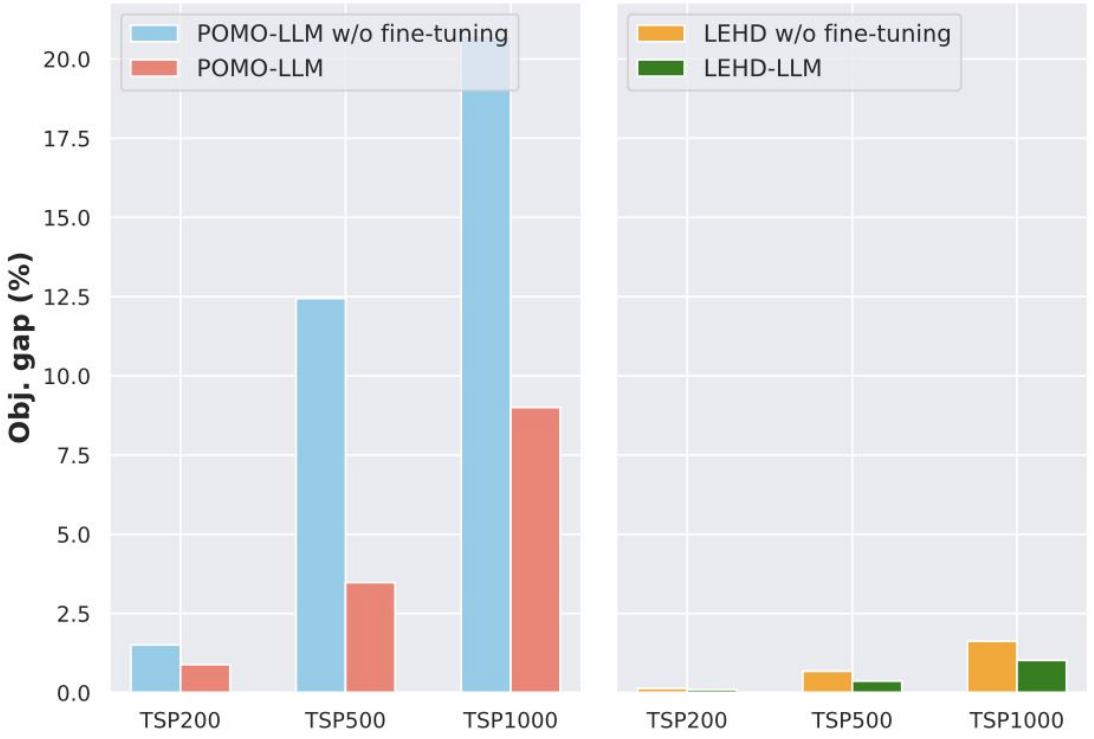


Figure 4: Performance of POMO-LLM and LEHD-LLM models with and without (w/o) fine-tuning strategy.

Ablation study

Effects of diverse scales for Fine-tuning

Table 5: Effect of problem size on fine-tuning POMO-LLM for TSP.

Method	TSP200	TSP500	TSP1000
POMO-LLM fixed size	0.972%	4.216%	11.782%
POMO-LLM varying size	**0.888%**	**3.479%**	**9.003%**

Effects of different LLMs

- We used GPT-4o mini, Llama, Mixtral, and Gemma. All experiments are conducted under identical settings on the TSP problem to ensure a fair comparison.
- Our experimental results indicated that integrating LLM-generated attention bias consistently improves the performance of NCO models.
- Among the evaluated models, GPT-4o mini achieves the best performance, demonstrating superior generalization and solution quality compared to the other LLMs.

Content

1. Background
2. Related work
3. Methodology
4. Experiment and Result
5. Conclusion

Conclusion

Our contributions:

- Propose a new **LLM-guided attention bias** for model fine-tuning to enhance the generalization capabilities of neural combinatorial optimization models. This attention bias is **automatically designed by LLMs**, which are used to **augment any attention-based NCO** models for better performance on large-scale problems.
- Develop an **efficient fine-tuning** process that leverages training on instances of varying sizes. This method improves the **model's flexibility** and **solution quality** by incorporating LLM-generated attention bias, allowing **effective generalization** to larger and more diverse problem instances without altering their architecture.
- Our proposed method can **achieve state-of-the-art performance** in solving the TSP and CVRP across various scales and also generalizes well to solve real-world TSPLib/CVRPLib problems.

Future directions:

- We could explore diverse LLM architectures, adapt to VRPs with **complex constraints** and **other combinatorial optimization problems** to improve scalability and efficiency, enhance interpretability for decision support, and benchmark against traditional heuristics to expand the applicability and effectiveness of LLM-powered neural solvers.

CONTACT US

FPT Software AI Center,
10 Pham Van Bach Street, Dich Vong Hau Ward,
Cau Giay District, Ha Noi City, Vietnam

Contact Person: Dr. Hoang Thanh-Tung
Email: htt210@gmail.com