8-2024

Hierarchical neural constructive solver for real-world TSP scenarios

Yong Liang GOH

Zhiguang CAO
Singapore Management University, zgcao@smu.edu.sg

Yining MA

Yanfei DONG

Mohammed Haroon DUPTY

See next page for additional authors

Citation

GOH, Yong Liang; CAO, Zhiguang; MA, Yining; DONG, Yanfei; DUPTY, Mohammed Haroon; and LEE, Wee Sun. Hierarchical neural constructive solver for real-world TSP scenarios. (2024). *KDD '24: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Barcelona, Spain, August 25-29*. 884-895.
Available at: https://ink.library.smu.edu.sg/sis_research/9334

Hierarchical Neural Constructive Solver for Real-world TSP Scenarios

Yong Liang Goh
Grabtaxi Holdings Pte Ltd &
National University of Singapore
Singapore
gyl@u.nus.edu

Zhiguang Cao
Singapore Management University
Singapore
zgcao@smu.edu.sg

Yining Ma
National University of Singapore
Singapore
yiningma@u.nus.edu

Yanfei Dong
National University of Singapore
Singapore
dyanfei@u.nus.edu

Mohammed Haroon Dupty
National University of Singapore
Singapore
haroon@nus.edu.sg

Wee Sun Lee
National University of Singapore
Singapore
leews@nus.edu.sg

ABSTRACT

Existing neural constructive solvers for routing problems have predominantly employed transformer architectures, conceptualizing the route construction as a set-to-sequence learning task. However, their efficacy has primarily been demonstrated on entirely random problem instances that inadequately capture real-world scenarios. In this paper, we introduce realistic Traveling Salesman Problem (TSP) scenarios relevant to industrial settings and derive the following insights: (1) The optimal next node (or city) to visit often lies within proximity to the current node, suggesting the potential benefits of biasing choices based on current locations. (2) Effectively solving the TSP requires robust tracking of unvisited nodes and warrants succinct grouping strategies. Building upon these insights, we propose integrating a learnable choice layer inspired by Hypernetworks to prioritize choices based on the current location, and a learnable approximate clustering algorithm inspired by the Expectation-Maximization algorithm to facilitate grouping the unvisited cities. Together, these two contributions form a hierarchical approach towards solving the realistic TSP by considering both immediate local neighbourhoods and learning an intermediate set of node representations. Our hierarchical approach yields superior performance compared to both classical and recent transformer models, showcasing the efficacy of the key designs.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; **Neural networks**; **Sequential decision making**; **Learning latent representations**.

KEYWORDS

neural constructive solver, traveling salesman problem, deep reinforcement learning

KDD '24, August 25–29, 2024, Barcelona, Spain
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3672053

ACM Reference Format:
Yong Liang Goh, Zhiguang Cao, Yining Ma, Yanfei Dong, Mohammed Haroon Dupty, and Wee Sun Lee. 2024. Hierarchical Neural Constructive Solver for Real-world TSP Scenarios. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3672053

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is a classical combinatorial optimization problem. Simply put, the TSP asks the following: given a set of cities, what is the shortest route where a salesman can visit every city only once and return back to his starting city? While it is simple to state, the TSP is a very difficult problem known to be NP-hard. Nevertheless, the TSP is a crucial problem to study, as many parallel problems can be reduced to solving the TSP, such as chip placement [20], the study of spin glass problems in physics [17], DNA sequencing [4], and many others.

Given its prevalence across a multitude of domains, the TSP has been extensively researched in the community. Particularly, the main approaches can be broken down into exact methods and approximate methods. Exact methods often materialize in the form of mathematical programming. Some popular exact solvers, such as Concorde [2], are developed based on linear programming and cutting planes. Approximate methods tend to be in the form of expert heuristics. An example would be the Lin-Kernighan-Helsgaun (LKH-3) algorithm, which utilizes heuristics and local search methods to update and improve initial solutions. As their names describe, exact methods return the true optimal routes while approximate ones return solutions often within some error bound of the optimal one. As the size of the problems grows, exact methods are intractable due to the NP-hard nature of the problem.

More recently, the deep learning community has put much effort into establishing practical neural solvers. These typically appear in the form of deep reinforcement learning [3, 10, 15, 19, 21], which presents a label-free approach to improve the models. This is preferred over supervised learning approaches (e.g, [14, 28]) since they require large amounts of labelled data, which is often challenging to obtain given the limited scalability of exact solvers.

It is important to note that learning-based solvers may perform well on specific target distributions they are trained on, but often suffer from poor generalization to other arbitrary instances. This

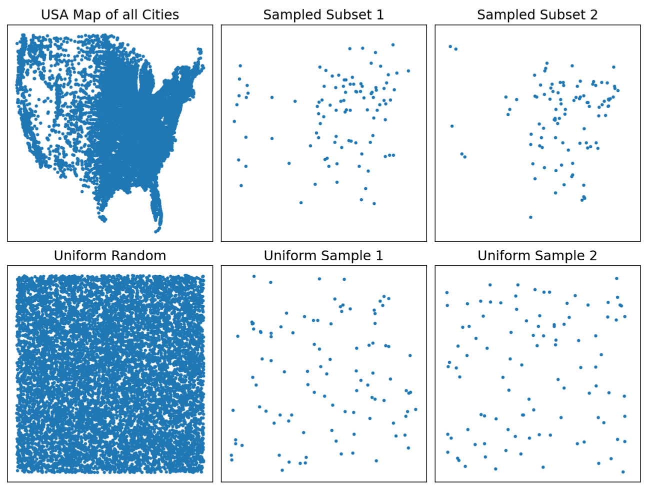

Figure 1: Comparing subset of TSP drawn from USA map (first row) against a uniform distribution (second row). Left-most plots show the base distribution. The subsets follow certain underlying structures for the USA case compared to completely random problems.

is acceptable if the target distributions reflect real-world cases. However, prior works have mostly only been trained and tested on problems derived from random synthetic distributions, which may not accurately represent real-world applications. While some learned solvers have been tested on realistic TSP instances, such as those from the TSPLIB [27], a collection of TSP instances observed in the real world, these realistic problem collections are typically too small to train on. Therefore, there is a gap in studying the performance of learning-based solvers in more realistic scenarios from the real world.

In this paper, we propose and study a setup that closely mirrors practical scenarios. Consider a logistic company with a large set of M fixed locations that it can deliver to. The company will do many delivery trips to these locations but for each trip, only a small subset V of the M locations, e.g., locations that placed orders for that day, need to be visited. This observation motivates us to generate realistic distributions as follows: select a set of M fixed locations from the real world to reflect real-world location distributions, and build each problem instance by randomly sampling a subset of V locations from the set of M locations.

We construct the locations using real-world datasets of cities from the USA, Japan, and Burma. When state-of-the-art neural solvers are trained in realistic settings, their performance may not be satisfactory. To improve the neural solvers, we focus on exploiting the nature of the node construction process. Existing neural solvers typically construct TSP tours autoregressively, solving the problem of visiting all unvisited cities starting from the current city and returning to the starting city. This suggests learning more effective and generalizable representations of the decision information regarding the current city and unvisited cities.

Specifically, we first observe that neural solvers often struggle to select a proper next city in the neighbourhood of the current city in our proposed realistic setup. This highlights the importance of exploiting more effective decision information about the current city. To this end, we propose to incorporate a customized hypernetwork layer [11] that leverages the embedding of the current city to modify the choice of the next city to visit.

Moreover, to obtain an improved representation of unvisited cities, we design a hierarchical representation that divides the cities into C partitions and use an embedding to represent the unvisited cities in each of the C partitions. We perform the partitioning using a soft clustering method, inspired by the EM algorithm. Given its differentiable property of EM, we can propagate the gradient through the clustering to learn the encoder parameters effectively.

We showcase the effectiveness of the hypernetwork and the hierarchical representation of unvisited cities in experiments with the proposed realistic setting. We highlight the following contributions:

- We introduce a more realistic TSP setting using real-world data to more convincingly demonstrate the effectiveness of neural TSP solvers.
- We make the key observation that neural solvers often struggle with node selection within a small locality, and we design a hypernetwork layer to emphasize local choices.
- We further exploit the nature of solving structured TSPs by representing the set of unvisited cities with multiple key embeddings using a differentiable soft clustering algorithm instead of conventional simplistic pooling methods.

2 RELATED WORK

2.1 Constructive Neural Solvers

Deep reinforcement learning forms the hallmark of training constructive neural solvers. Early works from [3] proposed to use the Pointer Network [32] based on the sequence-to-sequence architecture in [29] to solve TSP and Knapsack problems. They employ an actor-critic approach and achieved strong results on the TSP. Follow-up works from [26] further improve the performance of the Pointer Network.

Following on from this, the transformer network based on attention [31] was proposed by [19] to solve the TSP, Capacited Vehicle Routing Problem (CVRP) and the Orienteering Problem (OP). Primarily, the work showed that one can train a neural solver using the REINFORCE algorithm [33] and a simple greedy rollout of the network with a lagging baseline. Since then, multiple works based on the same architecture have been proposed to improve the predictive power of such solvers further [36]. POMO [21] was introduced and observed that constructive solvers were limited by their starting nodes. Hence, to effectively explore the search space of solutions, one should use all nodes as starting nodes, effectively constructing a simple beam search. Additionally, they showed that a stable baseline can be found in the average of all solutions. Sym-NCO [15] was proposed to exploit the symmetry of TSP by introducing symmetry losses to regularize the transformer network. Recently, ELG was introduced by [10] that defined a local learnable policy based on a k-Nearest Neighbor graph. They exemplified the generalizability of the network on large CVRP instances.

2.2 Improvement Neural Solvers

Apart from constructive methods, another approach to solving the TSP looks at improvement solvers. This methodology is inspired

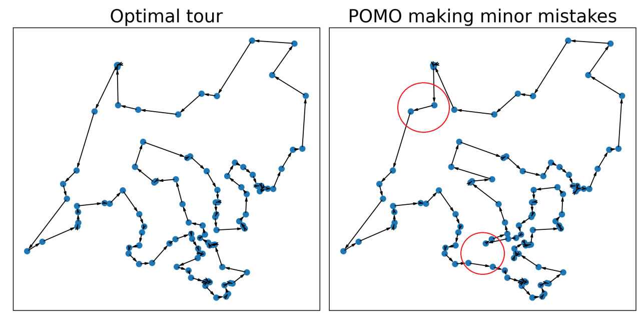

Figure 2: POMO model making a minor mistake with a poor selection of a local node

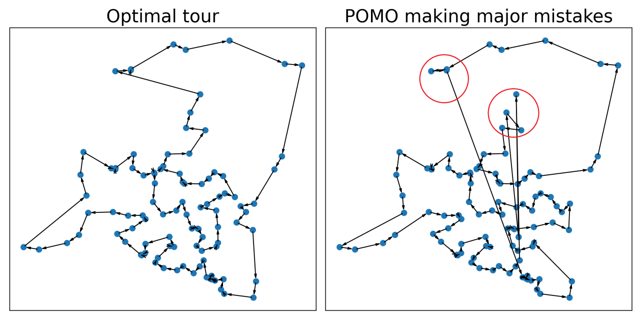

Figure 3: POMO model making a major mistake by not visiting nodes that are near it, causing cross-cluster routes that are inefficient

by algorithms such as 2-opt, whereby the solver first starts with a complete route, and a heuristic is then used to select edges to delete or add so as to edit the solution. Such methods are measured based on how quickly they can reach a strong solution. Work such as [7] uses neural networks to learn the 2-opt heuristic for improvement, while [34] uses the transformer to select node pairs for swaps for the TSP. Ma et al. [24] then extended the transformer network to learn node and edge embeddings separately, which is then upgraded for pickup and delivery problems in [23] and flexible k-opt in [22], pushing the iterative solver's performance further.

2.3 Search-based techniques

The previous two approaches are based on some form of learning-based search: constructive solvers try to perform a global search by learning heuristics entirely from data, whereas iterative solvers learn to guide local search techniques instead. Besides these, there is a class of search-based techniques that involve applying search during inference. Efficient Active Search (EAS) was proposed by [12] to introduce lightweight learnable layers at inference that could be tuned to improve the predictive power of a model on test samples. Other works such as [9] showcase that one can leverage a small pre-trained network and combine it with search techniques such as Monte-Carlo Tree Search (MCTS) [5] so as to solve large-scale TSPs. The work in [6] then combined both MCTS and EAS to improve the search capabilities further. Lastly, another work [18] showcased how one could combine dynamic programming with a pre-trained neural network to scale the TSP to 10,000 nodes.

Previous works attempt to regularize the networks via symmetry or scale the solver to larger problems. However, they essentially are still based on transformer models trained on arbitrary distributions. Apart from ELG, these works do not consider the impact of local choices nor explore further how to represent unvisited cities better, which are critical aspects of solving the TSP in our view.

3 OUR APPROACH

Generally, we find that neural solvers tend to make two classes of errors in route construction compared to the optimal solution for such practical scenarios. The first class often appears as a minor error, where a poor decision is made in a local neighbourhood, as shown in Figure 2. This results in a sub-optimal route because a local choice is not picked first. The second class tends to appear in problems with more structure, where the agent fails to visit all reasonable nodes within a local cluster and has to backtrack to the area, exemplified in Figure 3. This tends to give solutions that are significantly poorer than optimal.

Our approach seeks to tackle these two errors more effectively. We propose two main architectural improvements to the base transformer model to address these issues. Firstly, we propose a learnable local choice decoder that accentuates certain choices based on the agent's current location (and hence locality). Secondly, we propose a differentiable clustering algorithm to learn a set of representations to capture and summarize the set of remaining cities. Our full approach is illustrated in Figure 4.

3.1 Recap: Constructive Neural Solvers

In this subsection, we review previous works in well-known constructive neural solvers such as the attention model [19] and the POMO model [21]. The problem can be defined as an instance s on a fully connected graph of n nodes, where each node represents a city. Each node n_i where $i \in \{1, ..., n\}$ has features x_i, typically the 2D coordinate. A solution is defined as a TSP tour, and is stated as a permutation of the nodes given by $\tau = (\tau_1, ..., \tau_n)$, such that $\tau_t \in \{1, ..., n\}$. Note that since the tour does not allow backtracking, $\tau_t \neq \tau_{t'}, \forall t \neq t'$. The formulation yields the following policy:

$$p_\theta(\tau|s) = \prod_{t=1}^{n} p_\theta(\tau_t|s, \tau_{1:t-1}) \tag{1}$$

Since the model is trained via reinforcement learning, the policy's action thus refers to the selection of the next node τ_t given the current state. The entire policy p_θ is parameterized with a neural network which involves both an encoder and a decoder. The encoder is a standard transformer model represented as such

$$\tilde{\mathbf{h}}_i = \mathrm{LN}^l(\mathbf{h}_i^{l-1} + \mathrm{MHA}_i^l(h_1^{l-1}, ..., h_n^{l-1})) \tag{2}$$

$$\mathbf{h}_i^l = \mathrm{LN}^l(\tilde{\mathbf{h}}_\mathbf{i} + \mathrm{FF}(\tilde{\mathbf{h}}_\mathbf{i})) \tag{3}$$

where h_i^l is the embedding of the i-th node at the l-th layer, d the dimension size of the embeddings, MHA is the standard multi-headed attention layer [31], LN is a layer normalization function, and FF is a simple feed-forward multi-layer perceptron (MLP). Each node's embeddings go through a total of L-layers before being passed into a decoder.

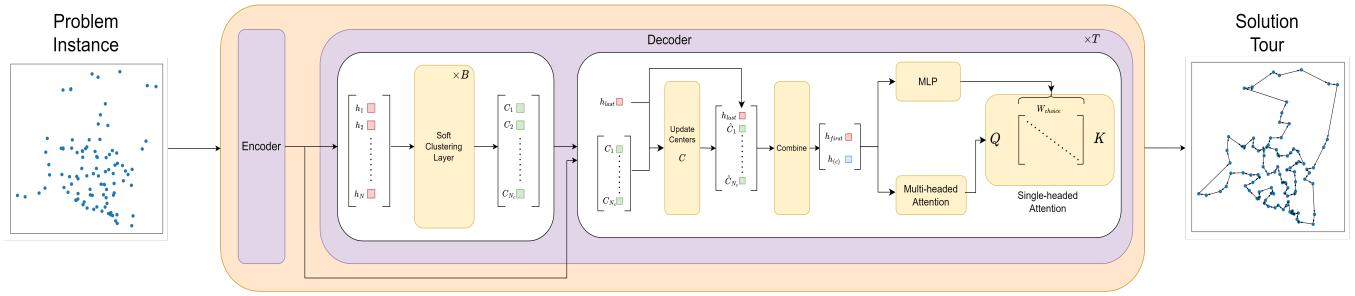

Figure 4: Overview of our proposed architecture. Given a TSP instance, we learn contextual embeddings of the cities in a set of cluster representations with an EM-inspired differentiable technique. In addition, our policy is dynamically adapted with a local hypernetwork which emphasises the completion of local cluster before moving on to new distant cities.

For the decoder, the solution is produced in an autoregressive fashion. In this case, at time step t, the decoder receives the following

$$\mathbf{h}_{(c)} = \mathbf{h}^L_{\text{LAST}} + \mathbf{h}^L_{\text{START}} \tag{4}$$

where $\mathbf{h}_{(c)}$ is known as a contextual embedding. In this instance, the context given is the sum of the L-th encoder layer's representation of the current node and the starting node. This is then first passed through a multi-headed attention layer where visited nodes are masked, followed by a single-headed attention layer for decision-making. In this decision layer, we obtain the following

$$Q = W_Q(H), K = W_K(H), V = W_V(H) \tag{5}$$

where H is the set of all node embedding after the decoder's multi-headed attention layer. With this, we compute the compatibility of the query with all nodes together with a mask, where the mask indicates previously visited nodes. This ensures that the decoder cannot pick an already visited node. Mathematically, we use the following attention mechanism

$$a_j = \begin{cases} U \cdot \text{TANH}(\frac{QK^\top}{\sqrt{d}}) & j \neq \tau_{t'}, \forall t' < t \\ -\infty & \text{otherwise} \end{cases} \tag{6}$$

where we clip the values between $[-U, U]$ using a TANH function as with works in [3, 19, 21]. These values are then normalized with a simple softmax function, giving us the following decision-making policy:

$$p_i = p_\theta(\tau_t = i | s, \tau_{1:t-1}) = \frac{e^{a_j}}{\sum_j e^{a_j}} \tag{7}$$

Finally, to train the model, the REINFORCE algorithm [33] is used, where works such as POMO [21] and Sym-NCO [15] use a shared baseline of all starting points. Concretely, the expected return J is maximized with gradient ascent and is approximated by

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} (R(\tau^i) - b^i(s)) \nabla_\theta \log p_\theta(\tau^i | s) \tag{8}$$

where $R(\tau^i)$ is the reward of permutation τ^i, the tour length. $p_\theta(\tau^i|s) = \prod_{t=1}^{T} p_\theta(a^i_t | s, a^i_{1:t-1})$ is the product of action probabilities for the trajectory, where actions refer to the selection of the next node to move to, s refers to the state of the problem, the set of all nodes, current node, and starting node. The baseline is calculated as the average return of all starting points, given by

$$b^i(s) = b_{\text{shared}}(s) = \frac{1}{N} \sum_{j=i}^{N} R(\tau^j) \forall i \tag{9}$$

3.2 Improving local decision making with the Choice Decoder

More often than not, good selections for the TSP tend to lie within the locality of the current position. In the standard transformer architecture, the decoder attempts to capture this by using the current node's representation in a single-headed attention layer, as shown in Equations 6 and 7. In fact, Equation 6 is essentially a kernel function between the current querying node and the candidate nodes, an observation made in [30]. Effectively, we can view the compatibility score as

$$\phi(Q, K) = U \cdot \text{TANH}(\frac{QK^\top}{\sqrt{d}}) \tag{10}$$

Given that we aim to focus more on the current node's vicinity and features, we propose to *generate a set of attention weights based on the current embeddings*. This aims to amplify or nullify the compatibility scores further by conditioning it on the current node. Hypernetworks [11] are small neural networks designed to generate a set of weights for a main network. Its goal is to serve as a form of relaxed weight sharing. This approach allows the hypernetwork to take as input some information about the problem, such as its structure, and adapt the main network's weights towards the problem. Inspired by this approach, we construct the set of attention weights using an MLP as a hypernetwork, with the input being the current node's embedding. Concretely, we modify the compatibility function as follows:

$$\hat{a}_j = U \cdot \text{TANH}(\frac{(QW_{\text{CHOICE}})K^\top}{\sqrt{d}}) \tag{11}$$

such that

$$W_{\text{CHOICE}} = \text{MLP}(Q), W_{\text{CHOICE}} \in \mathbb{R}^{dxd} \tag{12}$$

Effectively, W_{CHOICE} serves as a set of learnable conditional parameters that serve to alter the compatibility scores based on current embeddings. However, implementing this as a full matrix is extremely expensive. Instead, we realize W_{CHOICE} as a diagonal matrix, effectively reducing the compatibility score function to

$$\phi(Q, K|Q) = \tilde{a}_j = U \cdot \text{TANH}\left(\frac{(Q * W_{\text{CHOICE}})K^\top}{\sqrt{d}}\right) \tag{13}$$

where $*$ is the element-wise product. Inherently, W_{CHOICE} now reduces to a set of learnable scalars which serve to modify the compatibility scores between Q and K. Additionally, these learnable scalars are now conditioned upon Q, the current node, since it is generated via the MLP. The complexity of the MLP also now reduces from producing an output of $\mathrm{R}^{d \times d}$ to R^d.

3.3 Exploiting structure with Hierarchical Decoder

In its current state, the decoder models the TSP as a set-to-sequence function. A key aspect of the input is the contextual embedding, $\mathbf{h}_{(c)}$. This embedding serves to represent the current state the model is in and is often a combination of the starting node, current node, and some global representation of the problem. For the global representation, works such as [19] and [21] use an average of all node embeddings, while others such as [13], maintain an average of all visited nodes so far. Essentially, all of these approaches attempt to capture various nuances of the TSP.

However, for realistic problem settings, it is important to exploit the structure of the distribution of the cities. A single global representation would not be effective enough to capture the intricate correlations present between the cities. One notable and ubiquitous case is the presence of cluster patterns wherein certain cities are located near to each other while being distant to others. This clustering pattern, if captured within the global and contextual representation, can potentially provide the model with important clues to determine the next city to visit. Thus, for such problems, we propose to maintain a set of C representations that are able to summarize the set of unvisited cities left, instead of a simple single representation. We postulate that this is meaningful as structured problems have frequent cities in fixed areas of the map, and being able to identify if a node belongs to a certain area could be beneficial to the decision-making process.

Prior works in other domains have shown the efficacy of cluster construction in applications such as node classification [8]. In this work, we wish to group all cities into C representations. To this end, we design the following layer inspired by the Expectation-Maximization (EM) algorithm [25]. We first briefly review the EM algorithm for the Gaussian Mixture Model (GMM). Let $\theta = \{\pi_k, \mu_k\}$ denote the set of parameters, the coefficients of Gaussians π and its associated means μ (covariance Σ_k is assumed to be known), $\mathbf{X} = \{\mathbf{x}^{(i)}\}$ denote the set of data points, and $\mathbf{Z} = \{z^{(i)}\}$ denote the set of latent variables associated with the data. The maximum-likelihood objective is given by

$$\log p(\mathbf{X}|\pi, \mu) = \sum_{i=1}^{N} \log \sum_{z^{(i)}=1}^{K} p(\mathbf{x}^{(n)}|z^{(n)}; \mu) p(z^{(n)}|\pi) \tag{14}$$

In general, Equation 14 yields no closed-form solution. Additionally, it is non-convex, and its derivatives are expensive to compute. Since latent variable $z^{(i)}$ exists for every observation and we have a sum inside a log, we look at the EM algorithm to solve this. Typically, the EM algorithm involves two steps: the E-step computes the posterior probability over z given the current parameters, and the M-step, which assumes that given that the data was generated with $z^{(i)} = k$, finds the set of parameters that maximizes this. Effectively, for a standard GMM, this yields the following E-step, given an initial set of parameters θ:

$$\gamma_k = p(z^{(i)} = k|\mathbf{x}^{(i)}; \theta^{\text{old}}) = \frac{\pi_k \mathcal{N}(\mathbf{x}|\mu_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\mu_j)} \tag{15}$$

where γ_k can be viewed as the responsibility of cluster k towards data point $\mathbf{x}^{(i)}$. Then, for GMMs, the following update equations can be applied in the M-step:

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_k^{(i)} \mathbf{x}^{(i)} \tag{16}$$

$$\pi_k = \frac{N_k}{N}, \text{where} \sum_{i=i}^{N} \gamma_k^{(i)} \tag{17}$$

Essentially, Equation 15 estimates the contribution of each Gaussian model given the current set of parameters. While Equations 16 and 17, highlight closed-form update equations for the Gaussian parameters.

Now, suppose a TSP instance drawn from a fixed map can be represented efficiently with C latent representations. Since we are dealing with subsets of problems from the same space, these C representations can be fixed and learnable. Let $\mathbf{C} \in \mathrm{R}^{N_c \times d}$ denote this set of representations, where we have $C = \{c_1, c_2, ..., c_{N_c}\}$. We propose to learn and update these representations by considering a mixture model, where the latent variables are modelled by these latent embeddings. Similar to the EM algorithm, we produce a set of mixing coefficients using an attention layer and its attention weights. Concretely, our soft clustering algorithm estimates its mixing coefficients via the attention mechanism, and using these scores, the clusters are then updated with a weighted sum of the embeddings. This can be shown as

$$\hat{h}_i = W_H h_i \tag{18}$$

$$\hat{c_j} = W_C c_j \tag{19}$$

$$\pi_{i,j} = \text{SOFTMAX}\left(\frac{\hat{h}_i \hat{c_j}^\top}{\sqrt{d}}\right) \tag{20}$$

$$c_j = \sum_i \pi_{i,j} h_i \tag{21}$$

where Π is the matrix containing all coefficients $\pi_{i,j}$, W_H and W_C are parameters for the attentional scores, and C is the set of learnable embeddings for the distribution. Given a single set of parameters in the attention layer, the embeddings H and C are passed through this same layer a total of B times iteratively, mimicking a "rollout" of a soft clustering algorithm of E-steps and M-steps within each iteration. Loosely, we can see that Equation 20 resembles a similar calculation of the E-step, wherein we use a set of parameters to estimate the coefficients instead of minimizing for the Euclidean distance in the GMM case. Equation 21 is similar to the M-step of GMMs, where we update the centers (in our case the embeddings are the latent variables) with a weighted sum of the data.

Once we have the set of C embeddings, we update the representation at every step of decoding by subtracting a weighted sum of the current node's embedding; this computes the weighted sum of embeddings of unvisited cities, instead of all cities. Thus, at time step T, if the agent is current at node i, we update C via

$$c'_j = c_j - (\pi_{i,j} * h_i), \forall j \in N_c \tag{22}$$

Then, we now construct a new context embedding such that

$$\mathbf{h}_{(c)} = W_{\text{COMBINE}}[h^L_{\text{LAST}}, c_1, c_2, ..., c_{N_c}] + \mathbf{h}^L_{\text{FIRST}} \tag{23}$$

where $[\cdot]$ is the concatenation operation, and W_{COMBINE} is a simple linear layer to combine the embeddings. We keep $\mathbf{h}^L_{\text{FIRST}}$ separate so as to preserve the importance of the starting node. As the decoder constructs the solutions, the C embeddings get updated along the way, maintaining a small set of unvisited cities so as to keep track of the solution.

Algorithm 1 Psuedo code of soft clustering algorithm

```
1:  procedure CLUSTER(encoder embeddings $H$, number of centers $N_c$, number of iterations $B$, initial embeddings $C$, embedding size $d$)
2:      for $b \leftarrow 1$ to $B$ do
3:          $\hat{H} \leftarrow W_H(H)$
4:          $\hat{C} \leftarrow W_C(C)$
5:          $\pi = \text{SOFTMAX}(\frac{\hat{H}\hat{C}^\top}{\sqrt{d}})$          ▷ Compute attention scores
6:          $C = \sum_i \pi_i h_i$          ▷ Update the centers with data
7:          $C_{\text{OUT}} = \hat{C} + C$          ▷ Residual connection
8:          $C = \text{NORM}(C_{\text{OUT}})$          ▷ Layer normalization
9:      end for
10:     return $C$
11: end procedure
```

Algorithm 2 Psuedo code of one step of decoding in the hierarchical neural constructive solver

```
1:  function UPDATE(current node embedding $h_i$, cluster centers $C$, cluster weights $\pi$)
2:      $c'_j = c_j - (\pi_{i,j} * h_i), \forall c_j \in C$
3:      return $C'$
4:  end function
5:  procedure DECODE(encoder embeddings $H$, cluster centers $C$, cluster weights $\pi$, starting points $P$)
6:      $C' \leftarrow \text{UPDATE}(h_{\text{LAST}}, C, \pi)$          ▷ Remove visited embedding
7:      $\mathbf{h}_{(c)} \leftarrow W_{\text{COMBINE}}[h_{\text{LAST}}, c_1, c_2, ..., c_{N_c}] + \mathbf{h}^L_{\text{FIRST}}$
8:      $W_{choice} \leftarrow \text{MLP}(h_{\text{LAST}})$          ▷ Hypernetwork
9:      $\mathbf{h}'_{(c)} \leftarrow \text{MHA}(\mathbf{h}_{(c)}, K, V)$
10:     $\hat{a}_j = C \cdot \text{TANH}(\frac{(QW_{\text{CHOICE}})K^\top}{\sqrt{d}})$
11:     $u_j \leftarrow \text{SOFTMAX}(\hat{a}_j)$          ▷ Create action probabilities
12:     $p_i \leftarrow \text{SAMPLE}(u)$
13:     return $i, p_i$ ▷ Return the selected node and its probability
14: end procedure
```

In totality, our approach forms a hierarchy in the decision-making process; an intermediate level of C embeddings represent the set of unvisited cities and their groupings, and immediate local decision-making is learnt and skewed by W_{CHOICE} to favour decisions based on current positions. Algorithms 1 and 2 highlights the overall flow of our approach.

Table 1: List of augmentations suggested by [21]

$f(x,y)$	
(x,y)	(y,x)
$(x,1-y)$	$(y,1-x)$
$(1-x,y)$	$(1-y,x)$
$(1-x,1-y)$	$(1-y,1-x)$

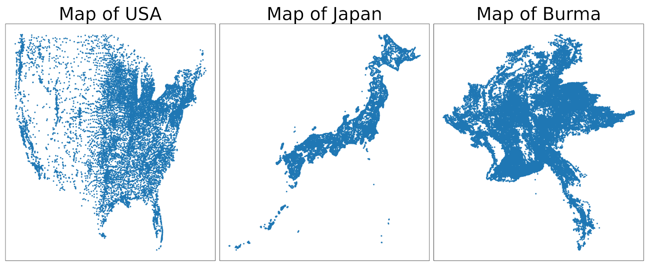

Figure 5: World maps used for experiments

4 EXPERIMENTAL SETUP

4.1 Data Generation

We present three different benchmarks for comparison. For all scenarios, we look at TSP-100 problems. Firstly, we generate random uniform data on a $[0,1]$ square and a fixed test set of $10,000$ instances. This is done to show if the addition of other layers interferes with the base performance of the transformer model.

To generate realistic data, we sample instances from 3 different countries, available at [1] and shown in Figure 5. Namely, they are

- USA13509 - 13,509 cities across the United States of America, each with a population >500
- BM33708 - 33,708 cities across the country of Burma
- JA9847 - 9,847 cities across the country of Japan

Each country is first normalized to a $[0,1]$ square. Then, at every training epoch, we randomly sample problems of size 100 from the map. Naturally, clusters that are denser across the country will be sampled more frequently. A test set size of $10,000$ samples is also drawn and held aside for evaluation. Each test set is fully solved via Concorde [2] to get the optimal length of each tour. We define 1 epoch to be $100,000$ samples, and the models are trained for 200 epochs to prevent overfitting. In totality, the model sees $20,000,000$ different samples. Thirdly, we define a limited data setting. This mimics a typical practical problem where the company does not have unlimited access to data. We first define a small dataset of $50,000$ samples for such a setting. Based on the experiment size, we sample the necessary amount of data. Likewise, the models are tested on the same test set of $10,000$ samples.

Additionally, to show the generality of our approach beyond the logistics domain, we include the PCB3038 dataset from TSPLib, where the goal is to find the shortest path across a circuit board layout. Here, we can view the problem as such: from all possible holes the board can have, and given a subset of these holes, what is the optimal path of traversal? Solving such problems with high degrees of accuracy leads to cost savings in manufacturing and improved yields.

4.2 Benchmark Models

We compare our approach to the following constructive neural solvers: POMO [21] the classical transformer that forms the basis for many follow-up works, Sym-NCO [15] a follow-up work from POMO that improves neural solvers by exploiting problem symmetry, and ELG [10] a recent work that also focuses on locality by defining a separate local policy based on its k-Nearest Neighbors. All models are trained based on the POMO shared baseline. It should be noted that in ELG, the authors introduced a different training algorithm. Since we wish to compare the efficacy of the architectural contributions, we train the ELG model using POMO. We reimplement all models and ran on the proposed dataset for our experimental results.

4.3 Hyperparameters

Since the neural models all share the same underlying backbone POMO transformer model, we retain the same set of hyperparameters across them to ensure the contributions are purely architectural. We utilize 6 layers of the transformer encoder and 2 layers of the transformer decoder. All models are trained for 200 epochs, with 100,000 episodes per epoch and a $1e^{-4}$ learning rate using the Adam optimizer [16]. Gradient clipping is set to $[-10, 10]$ for all models. For ELG, we used their recommended 50-nearest neighbours. As for our approach, we set $N_c = 5$ embeddings and $B = 5$ iterations for the hierarchical approach by using a validation set of 1,000 samples for verification. Additional details for hyperparameters can be found in Appendix A.1.

4.4 Performance Metric

All models are measured by the optimality gap, the percentage gap between the neural model's tour length and the optimal tour length. This is given by

$$O = \left(\frac{\frac{1}{N}\sum_i^N R_i}{\frac{1}{N}\sum_i^N L_i} - 1\right) * 100 \tag{24}$$

where L_i is the tour length of test instance i computed by Concorde. Also, we perform instance augmentation, just like in POMO, which involves various translations and reflections across the x and y axes [21], as shown in Table 1.

5 RESULTS

5.1 Performance on Uniform Random Distribution

Table 2 highlights the overall performance of the models on the classic uniform random sampling dataset. Most models have solid performance, with augmentation playing a prominent role in boosting the predictive power. Our model and ELG also show the importance of having some form of local feature to improve the decision-making process. The addition of our unvisited city tracking via soft clustering provides additional boost to the model's predictive power since we can identify locations on the square.

5.2 Performance on Structured Distributions

Table 3 showcases the different model's performance on TSP100 instances drawn from various countries. Our model has a clear advantage in the USA and Japan, with a narrow margin in Burma. Interestingly, we also see that augmentation has minimal effect on increasing the solver's performance, except for the case of Sym-NCO. It is likely because the maps are no longer symmetric in nature, and simple transformations do not improve the chances of finding a very different route. For Sym-NCO, we see that since it's specifically trained to exploit augmentations by considering all symmetries during training, its strong performance only appears when it can perform those transformations.

5.3 Performance on Varied Sizes

Figure 6 compares the model's performance between the complete set of training, 50,000 fixed samples, and 10,000 fixed samples. We can see that the model performance degrades as expected once data is limited. However, the ranking of the models is still similar at the 50,000 sample mark. Interestingly, reducing the dataset further to 10,000 samples sees ELG becoming the top neural constructive solver. We attribute ELG's strong performance on limited data to its local policy scheme. ELG uses well-crafted local features in the form of polar coordinates to create their local policy. The direct use of these features allows their local policy to learn valuable features to alter the action probabilities easily with less data. It should be noted that our choice hypernetwork is possibly a larger function class that also encompasses this approach. Hence, our model can learn a better overall function when more data is present.

5.4 Performance on PCB3038

Table 5 showcases the performance of the models on the PCB3038 TSPLib dataset. Evidently, the dataset is no longer as symmetric as before, since Sym-NCO struggles to greatly improve upon POMO. Instead, some local features are important to have, as displayed by ELG and our model's improvement over POMO. Since our approach is a superset function of ELG, it can learn a better-performing solver, significantly improving upon POMO.

5.5 Evolution of Embeddings

Soft clustering the nodes in the embedding space is a crucial part of our overall approach. Therefore, we evaluate if our algorithm indeed is able to cluster the nodes into a set of meaningful clusters as the training progresses. Specifically, we conduct a 2D t-SNE plot of the node and cluster embeddings during the training process. Figure 7 evidently shows that as the training progresses, the cluster centroid embeddings are better able to separate the node embeddings in the latent space. This plausibly leads to a better representation of the unvisited embeddings and hence the problem space, allowing the model to identify the groups of nodes so as to select similar ones first.

Table 2: Model performance on 10,000 generated samples from the uniform random distribution. Best model in bold.

Model	Tour Length	Opt. Gap (%)	Aug. Tour Length	Aug. Opt Gap (%)
Concorde	7.7649	-	7.7649	-
POMO	7.8824	1.5134%	7.8114	0.5997%
Sym-NCO	7.9106	1.8772%	7.8148	0.6425%
ELG	7.8223	0.7393%	7.7861	0.2734%
Ours	**7.8145**	**0.6397%**	**7.7809**	**0.2063%**

Table 3: Performance of various models on realistic TSP100 instances from 3 different countries. Best models are in bold.

Dataset	Model	Tour Length	Opt. Gap (%)	Aug. Tour Length	Aug. Opt. Gap (%)	Inference Time (10k samples)
USA13509	Concorde	5.6209	-	5.6209	-	56 min 37 sec
	POMO	5.6958	1.3334%	5.6922	1.2677%	2 min 46 sec
	Sym-NCO	5.7022	1.4650%	5.6604	0.7219%	2 min 49 sec
	ELG	5.6660	0.8022%	5.6641	0.7691%	2 min 58 sec
	Ours	**5.6548**	**0.6024%**	**5.6533**	**0.5762%**	3 min 03 sec
JA9857	Concorde	3.5341	-	3.5341	-	52 min 55 sec
	POMO	3.5621	0.7926%	3.5620	0.7893%	2 min 38 sec
	Sym-NCO	3.5670	0.9315%	3.5497	0.4421%	2 min 40 sec
	ELG	3.5576	0.6659%	3.5574	0.6596%	2 min 49 sec
	Ours	**3.5438**	**0.2741%**	**3.5435**	**0.2670%**	2 min 51 sec
BM33708	Concorde	5.0019	-	5.0019	-	59 min 22 sec
	POMO	5.0823	1.6060%	5.0746	1.4540%	2 min 55 sec
	Sym-NCO	5.0561	1.0828%	5.0354	0.6683%	2 min 58 sec
	ELG	5.0587	1.1343%	5.0528	1.0162%	3 min 02 sec
	Ours	**5.0383**	**0.7261%**	**5.0328**	**0.6166%**	3 min 06 sec

Table 4: Ablation study on the USA13509 map

Dataset	Model	Tour Length	Opt. Gap (%)	Aug. Tour Length	Aug. Opt. Gap (%)
USA13509	POMO only	5.6958	1.3334%	5.6922	1.2677%
	POMO + Choice	5.6676	0.8300%	5.6659	0.7997%
	POMO + Choice Free	5.7381	2.1038%	5.7311	1.9791%
	POMO + Choice + Average Tracking	5.6648	0.7807%	5.6638	0.7633%
	POMO + Choice + Soft Clustering Tracking	**5.6548**	**0.6024%**	**5.6533**	**0.5762%**

Table 5: Model performance on 10,000 on the PCB3038 dataset to show efficacy of models on problems from domains beyond logistics.

Dataset	Model	Tour Length	Opt. Gap (%)	Aug. Tour Length	Aug. Opt. Gap (%)
PCB3038	Concorde	7.5866	-	7.5866	-
	POMO	7.7952	2.7494%	7.6984	1.4743%
	Sym-NCO	7.7702	2.4201%	7.6776	1.1996%
	ELG	7.6882	1.3393%	7.6387	0.6876%
	Ours	**7.6417**	**0.7236%**	**7.6074**	**0.2746%**

5.6 Ablation Studies

As shown in Table 4, we perform ablation studies for our network and consider three different variations. Firstly, POMO + Choice adds only the local choice layer based on the hypernetwork. Secondly, POMO + Choice Free replaces the local choice hypernetwork layer with free parameters, meaning that the weights are no longer conditioned on the current node's embedding but rather allowed to

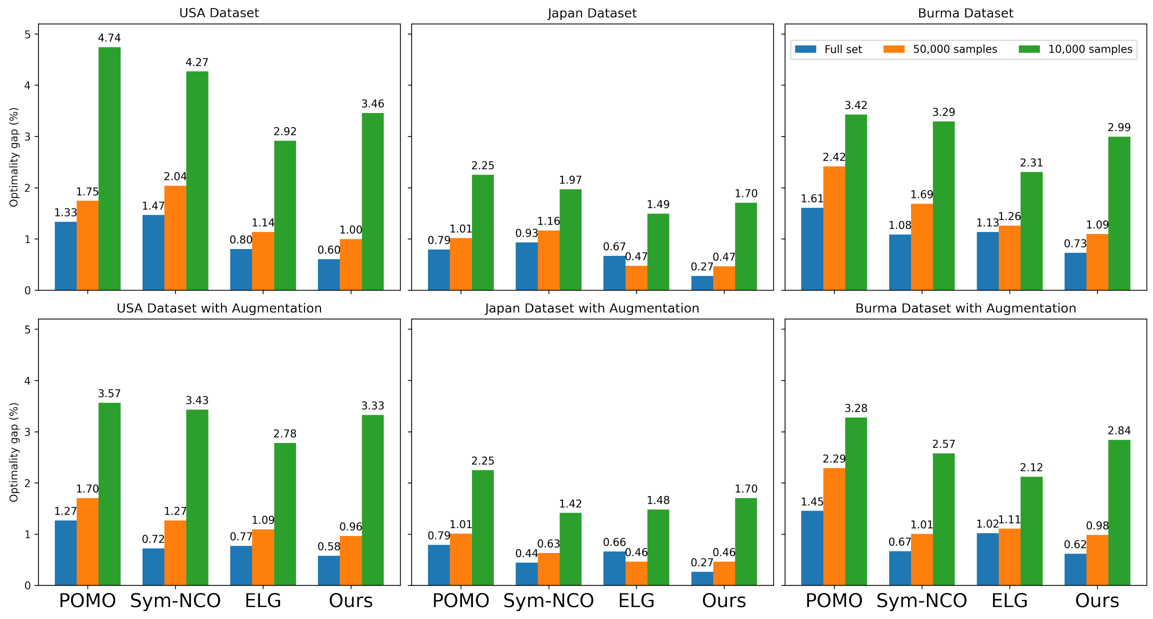

Figure 6: Optimality gaps across various dataset sizes (lower is better). The x-axis dictates the model types, and the y-axis denotes the optimality gap. The first row showcases standard data input, and the second shows data after augmentation. Countries are USA, Japan, and Burma, from left to right. Numerical details can be found in Appendix A.2.

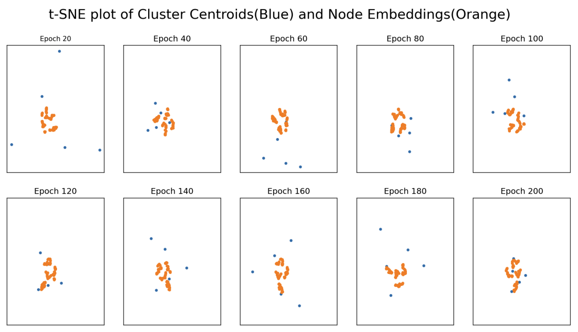

Figure 7: 2D t-SNE plot of cluster centroids (in blue) and a set of node embeddings (in orange) as the training progresses. Over time, the centroids surround and segregate the embeddings better.

learn freely. Thirdly, POMO + Choice + Average Tracking removes the soft clustering layer, instead averaging all unvisited nodes into a single embedding. From the table, we first see that the local choice layer is essential. Additionally, if we were to remove the ability to condition on the current node's position instead, the model would perform exceptionally poorly - even worse than the original transformer network. Finally, if we adopt the simplistic approach of averaging all unvisited cities, the model performance also suffers, showcasing the importance of our soft clustering layer.

6 CONCLUSION

In this work, we propose a more realistic approach to representing and generating Traveling Salesman Problems (TSPs) in real-world contexts. Our investigation reveals that previous state-of-the-art neural constructive solvers do not fully exploit the problem's intricacies to enhance predictive capability. To address this gap, we present a dual strategy to deal with the problem from two fronts. Firstly, we emphasize the importance of considering current agent positions, leading us to introduce a *hypernetwork*, which enables dynamic fine-tuning to the decision-making process based on the agent's current node position. Secondly, we recognize that realistic TSP scenarios are often structured, and therefore, improving solutions in such scenarios necessitates a deeper understanding of the structure of the set of unvisited nodes. Instead of treating all unvisited nodes uniformly, we propose a *soft clustering algorithm* inspired by the EM algorithm. This approach enhances the neural solver's performance by grouping nodes based on similarities, thereby increasing the likelihood of selecting nodes from the same cluster for early resolution. We illustrate the effectiveness of these methods across diverse geographical structures. Importantly, our methods are complementary and can be integrated with existing models like ELG or Sym-NCO to create more robust solutions.

ACKNOWLEDGMENTS

This work was funded by the Grab-NUS AI Lab, a joint collaboration between GrabTaxi Holdings Pte. Ltd. and National University of Singapore, and the Industrial Postgraduate Program (Grant: S18-1198-IPP-II) funded by the Economic Development Board of Singapore. This research is also supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031).

REFERENCES

[1] [n. d.]. https://www.math.uwaterloo.ca/tsp/world/countries.html

[2] David Applegate. 2003. Concorde: A code for solving traveling salesman problems. *http://www. tsp. gatech. edu/concorde. html* (2003).

[3] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* (2016).

[4] Marco Caserta and Stefan Voß. 2014. A hybrid algorithm for the DNA sequencing problem. *Discrete Applied Mathematics* 163 (2014), 87–99.

[5] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 4. 216–217.

[6] Jinho Choo, Yeong-Dae Kwon, Jihoon Kim, Jeongwoo Jae, André Hottung, Kevin Tierney, and Youngjune Gwon. 2022. Simulation-guided beam search for neural combinatorial optimization. *Advances in Neural Information Processing Systems* 35 (2022), 8760–8772.

[7] Paulo R d O Costa, Jason Rhuggenaath, Yingqian Zhang, and Alp Akcay. 2020. Learning 2-opt heuristics for the traveling salesman problem via deep reinforcement learning. In *Asian conference on machine learning*. PMLR, 465–480.

[8] Yanfei Dong, Mohammed Haroon Dupty, Lambert Deng, Zhuanghua Liu, Yong Liang Goh, and Wee Sun Lee. 2024. Differentiable Cluster Graph Neural Network. arXiv:2405.16185 [cs.LG]

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

[10] Chengrui Gao, Haopu Shang, Ke Xue, Dong Li, and Chao Qian. 2023. Towards generalizable neural solvers for vehicle routing problems via ensemble with transferrable local policy. *arXiv preprint arXiv:2308.14104* (2023).

[11] David Ha, Andrew Dai, and Quoc V. Le. 2016. HyperNetworks. arXiv:1609.09106 [cs.LG]

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

[13] Yan Jin, Yuandong Ding, Xuanhao Pan, Kun He, Li Zhao, Tao Qin, Lei Song, and Jiang Bian. 2023. Pointerformer: Deep Reinforced Multi-Pointer Transformer for the Traveling Salesman Problem. *arXiv preprint arXiv:2304.09407* (2023).

[14] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. 2019. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227* (2019).

[15] Minsu Kim, Junyoung Park, and Jinkyoo Park. 2022. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *Advances in Neural Information Processing Systems* 35 (2022), 1936–1949.

[16] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[17] Scott Kirkpatrick and Gérard Toulouse. 1985. Configuration space analysis of travelling salesman problems. *Journal de Physique* 46, 8 (1985), 1277–1292.

[18] Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. 2022. Deep policy dynamic programming for vehicle routing problems. In *International conference on integration of constraint programming, artificial intelligence, and operations research*. Springer, 190–213.

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

[20] Ratnesh Kumar and Zhonghui Luo. 2003. Optimizing the operation sequence of a chip placement machine using TSP model. *IEEE Transactions on Electronics Packaging Manufacturing* 26, 1 (2003), 14–21.

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

[22] Yining Ma, Zhiguang Cao, and Yeow Meng Chee. 2023. Learning to Search Feasible and Infeasible Regions of Routing Problems with Flexible Neural k-Opt. In *Thirty-seventh Conference on Neural Information Processing Systems*.

[23] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Hongliang Guo, Yuejiao Gong, and Yeow Meng Chee. 2022. Efficient Neural Neighborhood Search for Pickup and Delivery Problems. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*. 4776–4784.

[24] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. 2021. Learning to iteratively solve routing problems with dual-aspect collaborative transformer. *Advances in Neural Information Processing Systems* 34 (2021), 11096–11107.

[25] Todd K Moon. 1996. The expectation-maximization algorithm. *IEEE Signal processing magazine* 13, 6 (1996), 47–60.

[26] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. 2018. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems* 31 (2018).

[27] Gerhard Reinelt. [n. d.]. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[28] Zhiqing Sun and Yiming Yang. 2023. Difusco: Graph-based diffusion solvers for combinatorial optimization. *arXiv preprint arXiv:2302.08224* (2023).

[29] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[30] Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2019. Transformer dissection: a unified understanding of transformer's attention via the lens of kernel. *arXiv preprint arXiv:1908.11775* (2019).

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[32] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. *Advances in neural information processing systems* 28 (2015).

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

[34] Yaoxin Wu, Wen Song, Zhiguang Cao, Jie Zhang, and Andrew Lim. 2021. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems* 33, 9 (2021), 5057–5069.

[35] Haoran Ye, Jiarui Wang, Helan Liang, Zhiguang Cao, Yong Li, and Fanzhang Li. 2024. Glop: Learning global partition and local construction for solving large-scale routing problems in real-time. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 20284–20292.

[36] Cong Zhang, Yaoxin Wu, Yining Ma, Wen Song, Zhang Le, Zhiguang Cao, and Jie Zhang. 2023. A review on learning to solve combinatorial optimisation problems in manufacturing. *IET Collaborative Intelligent Manufacturing* 5, 1 (2023), e12072.

A APPENDIX

A.1 Hyperparameters

As described in the main body, we retain all original settings of the transformer. This yields the following:

- 6 encoder layers
- 2 decoder layers
- We removed global graph embedding as it was found to harm performance
- Clipping value U is set to 10.0
- Batch size for training is 128 samples
- A total of 200 epochs were run, where each epoch consisted of $100,000$ episodes
- Adam optimizer was used with a weight decay of $1e-6$
- Learning rate of 0.0001 was used
- Unique to our model:
  - The soft clustering layer is applied $B = 5$ times
  - We use a total of $N_c = 5$ embeddings for the clustering

A.2 Model performance across varying dataset size

Table 6 showcases in detail the bar plot results from Figure 6. Here, we compare the model's performance across three different sizes: a full training dataset of 20,000,000 samples, a small dataset of 50,000 samples repeated across 200 epochs, and an extremely small dataset of 10,000 samples repeated across 200 epochs. In totality, we can see that even though we reduced the dataset size significantly, the performance at 50,000 samples is still remarkable. The models retain the ranking in performance in this scenario. However, reducing this dataset further sees ELG become the top performing model. This is likely because ELG leverages distinct features in the form of polar coordinates in a small k-Nearest Neighborhood. By having these explicit features, the model requires less data to translate it to a meaningful representation. Whereas for our model, we learn the importance of the pairings entirely in the latent space, requiring more data. It should also be noted that our representation of locality is likely a superset of ELG's approach.

A.3 Comparison with General Solvers

While our approach biases the solver towards the distribution, recent works such as GLOP [35] proposed generic solvers that utilize the attention model as a Hamiltonian path solver. Table 7 compares our neural solver against GLOP. As GLOP's main premise is to break down a large problem and solve multiple sub-paths using local solvers trained on smaller sizes. Since our test results are on TSP100, we utilize the TSP25 and TSP50 solvers. Overall, we can see that our approach is stronger than a generically trained solver in both speed and performance. Nevertheless, since GLOP is based on the transformer model, our contributions can easily be integrated.

Table 6: Optimality gaps across various countries and dataset sizes. Best performing models in bold.

Dataset	# of Samples	Model	Tour Length	Opt. Gap (%)	Aug. Tour Length	Aug. Opt. Gap (%)
USA13509	50,000	Concorde	5.6209	-	5.6209	-
		POMO	5.7190	1.7458%	5.7166	1.7024%
		Sym-NCO	5.7345	2.0394%	5.6912	1.2691%
		ELG	5.6848	1.1361%	5.6824	1.0936%
		Ours	**5.6769**	**0.9954%**	**5.6750**	**0.9617%**
USA13509	10,000	Concorde	5.6209	-	5.6209	-
		POMO	5.8875	4.7433%	5.8848	4.6950%
		Sym-NCO	5.8598	4.2695%	5.8125	3.4282%
		ELG	**5.7849**	**2.9172%**	**5.7771**	**2.7782%**
		Ours	5.8152	3.4575%	5.8079	3.3265%
JA9847	50,000	Concorde	3.5341	-	3.5341	-
		POMO	3.5699	1.0141%	3.5697	1.0073%
		Sym-NCO	3.5724	1.0837%	3.5622	0.7960%
		ELG	3.5508	0.4723%	**3.5504**	**0.4627%**
		Ours	**3.5506**	**0.4667%**	3.5504	0.4629%
JA9847	10,000	Concorde	3.5341	-	3.5341	-
		POMO	3.6137	2.2514%	3.6136	2.2494%
		Sym-NCO	3.6037	1.9695%	**3.5841**	**1.4154%**
		ELG	**3.5868**	**1.4917%**	3.5863	1.4779%
		Ours	3.5943	1.7037%	3.5942	1.7016%
BM33708	50,000	Concorde	5.0019	-	5.0019	-
		POMO	5.1228	2.4158%	5.1164	2.2883%
		Sym-NCO	5.0878	1.7164%	5.0532	1.0242%
		ELG	5.0648	1.2570%	5.0572	1.1055%
		Ours	**5.0566**	**1.0925%**	**5.0520**	**0.9833%**
BM33708	10,000	Concorde	5.0019	-	5.0019	-
		POMO	5.1732	3.4248%	5.1659	3.2784%
		Sym-NCO	5.1666	3.2914%	5.1307	2.5740%
		ELG	**5.1173**	**2.3064%**	**5.1080**	**2.2109%**
		Ours	5.5157	2.9934%	5.1440	2.8393%

Table 7: Comparison of GLOP and our model on the various datasets.

Dataset	Model	Aug. Tour Length	Aug. Opt. Gap (%)	Run-time
USA13509	Concorde	5.6209	-	
	GLOP	5.6704	0.8799%	6 min 07 sec
	Ours	**5.6533**	**0.5762%**	**3 min 03 sec**
JA9847	Concorde	3.5341	-	
	GLOP	3.5839	1.4094%	6 min 04 sec
	Ours	**3.5435**	**0.2670%**	**3 min 01 sec**
BM33408	Concorde	5.0019	-	
	GLOP	5.0510	0.9822%	6 min 07 sec
	Ours	**5.0328**	**0.6166%**	**3 min 04 sec**