2-2021

Multi-decoder attention model with embedding glimpse for solving vehicle routing problems

Liang XIN

Wen SONG

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

Jie ZHANG

Multi-Decoder Attention Model with Embedding Glimpse for Solving Vehicle Routing Problems

Liang Xin,[1*] **Wen Song,**[2*] **Zhiguang Cao,**[3†] **Jie Zhang**[1]
[1]Nanyang Technological University, Singapore
[2]Shandong University, China
[3]National University of Singapore, Singapore
XINL0003@e.ntu.edu.sg, wensong@email.sdu.edu.cn, zhiguangcao@outlook.com, zhangj@ntu.edu.sg

Abstract

We present a novel deep reinforcement learning method to learn construction heuristics for vehicle routing problems. In specific, we propose a Multi-Decoder Attention Model (MDAM) to train multiple diverse policies, which effectively increases the chance of finding good solutions compared with existing methods that train only one policy. A customized beam search strategy is designed to fully exploit the diversity of MDAM. In addition, we propose an Embedding Glimpse layer in MDAM based on the recursive nature of construction, which can improve the quality of each policy by providing more informative embeddings. Extensive experiments on six different routing problems show that our method significantly outperforms the state-of-the-art deep learning based models.

Introduction

Routing problems, such as the Travelling Salesman Problem (TSP) and Capacitated Vehicle Routing Problem (CVRP), are a family of combinatorial optimization problems (COP) that have extensive real-world applications in many domains (Toth and Vigo 2014). Due to the combinatorial nature, routing problems are NP-hard in general (Applegate et al. 2006). Exact approaches, such as branch-and-bound algorithms (Fischetti, Toth, and Vigo 1994), have nice theoretical guarantee of optimality, but the (worst-case) computation complexity is exponential. In contrast, approximate algorithms guided by heuristics can find near-optimal solutions with polynomial computation complexity, therefore are often preferred, especially for large-scale problems.

Traditional approaches design hand-crafted rules as the heuristics. Instead, as modern approaches, deep learning models learn the heuristics from data samples (Bello et al. 2017; Dai, Dai, and Song 2016; Nazari et al. 2018; Kool, van Hoof, and Welling 2019a; Chen and Tian 2019). Most of these deep learning methods follow the encoder-decoder structure, and learn construction heuristics by repeatedly adding nodes (or locations) into an empty or partial solution until completion. Particularly, the encoder maps the information of nodes into feature embeddings, and the decoder predicts the probabilities of selecting each valid node at every construction step. To improve solution quality, different methods (e.g. sampling (Kool, van Hoof, and Welling 2019a) or beam searching (Nazari et al. 2018)) are used to generate a set of solutions from the trained construction policy to get the best one.

Though showing promising results, existing works suffer from two major limitations. First, the generated solutions are not *diverse* enough. Intuitively, a more diverse set of solutions could potentially lead to better ones. This is because for VRP and many other COPs, multiple optimal solutions exist and trying to find different ones will increase the chance of finding at least one. In addition, with the same number of solutions, generating less diverse and partially identical solutions will leave less space for the potentially better ones. Existing methods train only one constructive policy discriminatively, and the solutions are created using sampling or beam search from this same policy. The only source of diversity comes from the relatively deterministic probability distribution, which is far from enough. The second limitation, as pointed out in (Xin et al. 2020), is regarding the training of construction policy itself. The construction process can be viewed as a sequence of node selection sub-tasks, where the already visited node is *irrelevant* to the future decisions. However, most existing models for learning construction heuristic (Kool, van Hoof, and Welling 2019a; Bello et al. 2017) use the same node embeddings to decode at each step, without eliminating irrelevant nodes. Therefore, the node embeddings are based on the original graph for the whole task, not the graphs for each sub-task, and may deteriorate the quality of the trained policy.

In this paper, we address the above limitations simultaneously. First, to improve diversity, we propose the Multi-Decoder Attention Model (MDAM) to train multiple construction policies. It employs a Transformer (Vaswani et al. 2017) to encode the node information, and multiple identical attention decoders with unshared parameters to sample different trajectories. During training, each of the decoders learns distinct solution patterns, and is regularized by a Kullback-Leibler divergence loss to force the decoders to output dissimilar probability distribution of selecting nodes. Based on MDAM, we propose a novel beam search scheme where separate beams are maintained for each decoder. This

[*]Liang Xin and Wen Song contributed equally.
[†]Zhiguang Cao is the corresponding author.

enables full utilization of the distinct patterns learned by each decoder, and effectively keeps the diversity of solutions. Secondly, to increase the quality of trained construction policies, we propose an Embedding Glimpse layer in MDAM by exploiting the recursive nature of routing problems. As the visited nodes become unrelated to the future decisions, we explicitly remove them in the top attention layer of our encoder. Therefore the decoders will get more informative embeddings for selecting the next node, hence increase the quality of each single solution.

We would like to note that, rather than outperforming those highly optimized solvers in general, we are here to push the edge of deep learning model towards learning stronger heuristics for routing problems and potentially other combinatorial problems with weak hand-designed heuristics. Similar to (Kool, van Hoof, and Welling 2019a), while focusing on TSP and CVRP, our method is flexible and generally applicable to a wide range of routing problems with different constraints and even uncertainty. Extensive experiment results on six routing problems well confirm the effectiveness of increasing diversity and removing irrelevant nodes. More importantly, our model significantly outperforms state-of-the-art deep reinforcement learning based methods, and also demonstrates comparable or superior performance to the traditional non-learning based heuristics and sophisticated solvers in short inference time.

Related Works

Among existing models for learning construction heuristics, Pointer Network (PtrNet) in (Vinyals, Fortunato, and Jaitly 2015) uses Long Short-Term Memory (LSTM) Networks (Hochreiter and Schmidhuber 1997) as encoder and decoder to solve TSP with supervised learning, hence limited to small scale due to the expensive query for labels (optimal solutions). In contrast, Bello et al. (2017) use REINFORCE algorithm (Williams 1992) to train PtrNet. Without the need for true labels, the model can be trained on TSP with larger sizes. Instead of encoding nodes sequentially with LSTM in PtrNet, Nazari et al. (2018) use permutation invariant layers to encode nodes and train this model for CVRP. They also improve the solution quality using beam search to keep track of the most promising solutions and choose the best one. Kool, van Hoof, and Welling (2019a) adopt the Transformer model (Vaswani et al. 2017) to encode nodes and use a pointer-like attention mechanism to decode. By sampling 1,280 solutions from the trained policy, this elegant model achieves state-of-the-art results on several routing problems.

Some other works do not adopt the above encoder-decoder structure. Dai et al. (2017) use Deep Q-Learning algorithm to train a deep architecture over graph, i.e. Structure2Vec (Dai, Dai, and Song 2016), for several combinatorial optimization problems including TSP. However, the full connectivity makes the graph structure of TSP unimportant and results in unsatisfactory performance compared to that of other problems with crucial graph information, such as Minimum Vertex Cover and Maximum Cut. Instead of learning construction heuristics, Chen and Tian (2019) propose NeuRewriter to learn improvement heuristics, and train a region-picking policy and a rule-picking policy that recursively refine an initial solution for certain steps. For CVRP, NeuRewriter outperforms the sampling results in (Kool, van Hoof, and Welling 2019a). However, unlike our easily parallelized searching method, NeuRewriter is naturally not parallelizable when solving an instance since it uses sequential rewriting operations. Different from learning to pick nodes in construction or improvement heuristics, Lu, Zhang, and Yang (2020) design a network to learn which type of local move to pick and exhaustively search for the best possible greedy move of this type at each local improvement step. Despite the good solution quality, this model is not practical due to its prohibitively long computation time caused by the tens of thousands of exhaustive searching over the local move space at each step.

Model

Multi-Decoder Attention Model and Search

To output diverse solutions, we propose the Multi-Decoder Attention Model (MDAM) and design a customized beam search scheme utilizing the structure of multiple decoders to effectively keep the diversity of the solutions in the beam.

Multi-Decoder Attention Model The MDAM consists of an encoder and multiple decoders with identical structures but unshared parameters. The encoder takes the input instance x as the two-dimensional coordinates of nodes in TSP and embed them into feature vectors. For other routing problems, additional dimensions such as *demands* and *prizes* are included as input. During each step of the solution construction, each decoder takes node embeddings to produce probabilities of visiting each valid node. The MDAM architecture is shown in Figure 1, while EG layer will be detailed later.

The encoder of MDAM follows the Transformer Model (Vaswani et al. 2017). It consists of multiple self-attention blocks, the core of which is the multi-head attention layer and is defined formally as follows:

$$Q_i^h, K_i^h, V_i^h = W_Q^h X_i, W_K^h X_i, W_V^h X_i, \tag{1}$$

$$\begin{aligned} A^h &= \text{Attention}(Q^h, K^h, V^h) \\ &= \text{softmax}(Q^h {K^h}^T / \sqrt{d_k}) V^h, h = 1, 2, ..., H, \end{aligned} \tag{2}$$

$$\text{Multihead}(Q, K, V) = \text{Concat}(A^1, A^2, ..., A^H) W_O, \tag{3}$$

where Eqs. (1) and (2) are performed for each of the H attention heads; X_i is the d-dimensional embedding for the ith node; Q, K, V are ***Query***, ***Key***, ***Value*** vectors, respectively, $W_Q^h, W_K^h, W_V^h \in R^{d \times d_k}$ with $d_k = d/H$; the attention outputs A^h are concatenated and projected with $W_O \in R^{d \times d}$ to get this multi-head attention layer output.

Then skip-connection layers (He et al. 2016), Batch Normalization (BN) layers (Ioffe and Szegedy 2015) and two linear projection layers with ReLU activation in between (referred as FF) are used to get the output f of this self-attention block as follows:

$$\hat{f}_i = \text{BN}(X_i + \text{Multihead}_i(Q, K, V)), \tag{4}$$

$$f_i = \text{BN}(\hat{f}_i + \text{FF}(\hat{f}_i)). \tag{5}$$

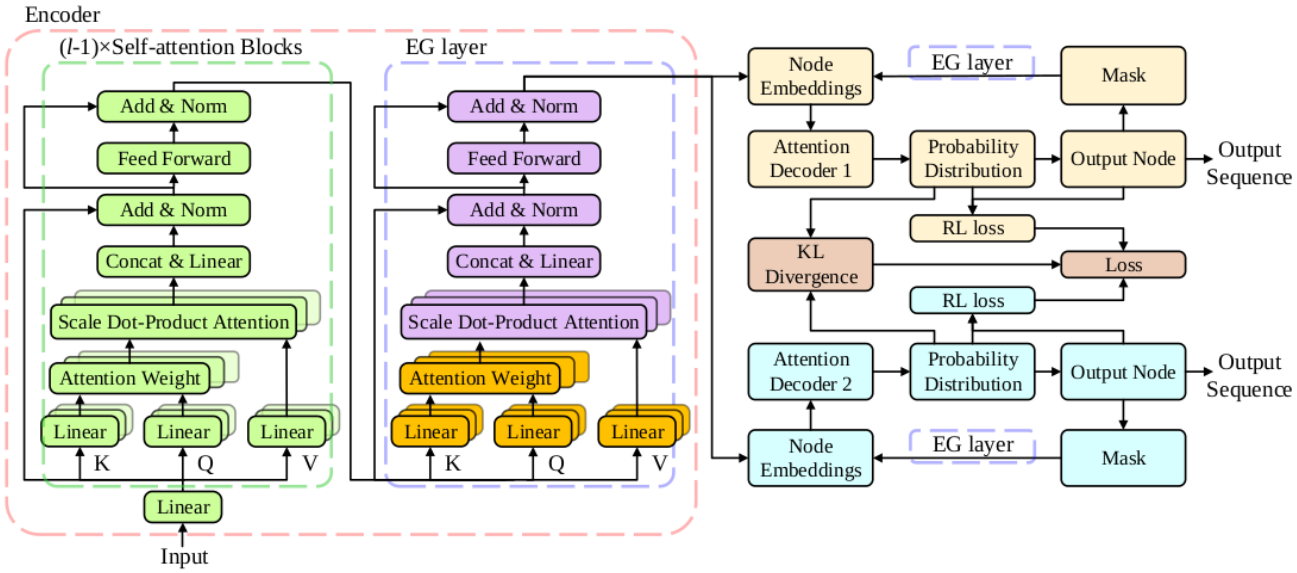

Figure 1: The Multi-Decoder Attention Model (we use two decoders for illustration)

Let M be the number of decoders with identical structure. Each decoder, indexed by m, is an attention mechanism that models the probability of selecting next node to visit at each step t, $P^m(y_t|x, y_1, ..., y_{t-1})$, following (Kool, van Hoof, and Welling 2019a). The decoder indexed by m is defined formally as follows:

$$f_c = \text{Concat}(\bar{f}, f_{C_0}, f_{C_{t-1}}), \tag{6}$$

$$g_c^m = \text{Multihead}(W_{gQ}^m f_c, W_{gK}^m f, W_{gV}^m f), \tag{7}$$

$$q^m, k_i^m = W_Q^m g_c^m, W_K^m f_i, \tag{8}$$

$$u_i^m = D \tanh((q^m)^T k_i^m / \sqrt{d}), \tag{9}$$

$$P^m(y_t|x, y_1, ..., y_{t-1}) = \text{softmax}(u^m), \tag{10}$$

where f_c is the context embedding; $\bar{f}$ is the mean of the nodes' embeddings f; f_{C_0} and $f_{C_{t-1}}$ are the embeddings of the starting node and the current node, respectively, which are replaced by trainable parameters for the first step; Eq. (7) is a multi-head attention over the valid nodes at step t to get a new context, similar to the glimpse in (Bello et al. 2017); g_c^m, q^m, k_i^m are of dimension d; Eq. (9) uses $D = 10$ to clip the result for better exploration following (Bello et al. 2017); the softmax function in Eq. (10) is over the set of valid nodes.

While greedily decoding, each decoder independently outputs a trajectory by selecting the node with maximum probability at each step. We impose a regularization to encourage the decoders to learn distinct construction patterns and output diverse solutions. A Kullback-Leibler (KL) divergence between each pair of the output probability distributions from the multiple decoders of MDAM is maximized as the regularization during training:

$$D_{KL} = \sum_s \sum_{i=1}^{M} \sum_{j=1}^{M} \sum_y P^i(y|x, s) \log \frac{P^i(y|x, s)}{P^j(y|x, s)}, \tag{11}$$

where $P^i(y|x, s)$ is the probability that decoder i selects node y in state s for instance x.

Customized Beam Search While the solution space for combinatorial optimization problems is exponentially large, evaluating a small set of solutions is computationally feasible with the known deterministic metric (e.g. tour length for routing problems). Therefore, it is desirable to output a set of solutions and retrieve the best one. However, with learned construction heuristics, existing methods like the typical sampling and beam search fail to maintain a set of diverse solutions. This issue comes from the fact that randomly sampling (with replacements) from the same distribution (policy) frequently outputs partially or completely repeated solutions, and beam search on a tree finds a set of unique solutions but with low variability and being deterministic (Kool, Van Hoof, and Welling 2019b).

To achieve better performance, we propose a novel beam search scheme customized for MDAM where each decoder performs search independently. Given a required beam size $\mathcal{B}$, we maintain a separate beam for each decoder to keep the diversity of the whole beam, and the size of a beam is $B = \lceil \mathcal{B}/M \rceil$. The solutions in the same beam will utilize a consistent construction pattern since they are constructed by the same decoder.

During searching of the same decoder, we can usually conclude the *inferiority* of a partial solution to another, without the need of full construction. Taking TSP for example, some partial solutions of a decoder may collapse into having the same starting node, set of visited nodes, and current node, i.e. the same remaining sub-task. Figure 2 depicts an example, where the second (1→2→4→5) and third (1→4→2→5) partial solutions in step 4 of decoder 1 collapse. One of these two will have a partial tour length longer than or equal to the other. Hence with all possible future decisions of the same decoder, it can never achieve shorter complete tour length, i.e. inferior to the other one. For CVRP, similar analysis also applies. But to ensure inferiority, in addition to the collapse condition, the partial so-

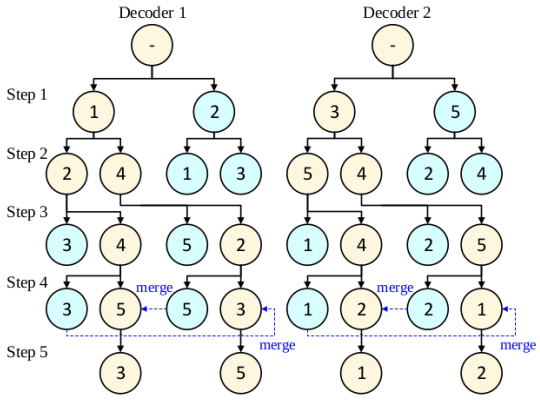

Figure 2: The Multi-Decoder Attention Model (we use two decoders for illustration)

lution with the tour length longer than or equal to the other should have less or same vehicle capacity left.

Here we design a merging technique based on the inferiority conditions above to avoid unnecessary solution constructions and save space in the beam for potentially better ones. Specifically, in each beam we evaluate partial solutions on the fly, and merge them whenever one is inferior to another one. After merging, the inferior one will be deleted, and the probability of the superior one will take the larger value of the merged two. Due to pairwise comparisons, the computation overhead for merging grows quadratically with the increase of beam size. Nevertheless, this is not a critical issue for our method since we maintain separate beams for each decoder, the sizes of which (B) are often small.

Embedding Glimpse Layer

For construction heuristics, the model needs to output a series of node selections in an auto-regressive way. The already visited nodes are no longer relevant to the future decisions, and decoding based on the same embeddings for all the construction steps may lead to poor performance. To further improve the solution quality, an ideal way is to re-embed only the unvisited nodes after visiting one node at each tour construction step. However, this requires complete re-embedding of nodes in each step, and is extremely expensive. To address it, we propose the Embedding Glimpse (EG) layer to approximate the re-embedding process. This is based on the intuition that higher layer of a neural network extracts more task-related features. Taking the Convolutional Neural Network (CNN) for Computer Vision tasks as an example, the lower layers usually extract general features like pixel and edge patterns, while the features from the upper layers are more related to solving the specific tasks (Schroff, Kalenichenko, and Philbin 2015). We hypothesize that the attention model shares a similar hierarchical feature extraction paradigm, and design the EG layer as follows.

In the encoder with l self-attention blocks, we perform the re-embedding process approximately by fixing the lowest $l-1$ attention layers and masking out the attention weights of the irrelevant nodes in the top attention layer to get the new node embeddings. Termed as EG layer, this top attention layer is part of the encoder and keeps the same for each of the multiple decoders. Part of the EG layer can be precomputed for only one time (the orange boxes in Figure 1):

$$Q_i^h, K_i^h, V_i^h = W_Q^h X_i, W_K^h X_i, W_V^h X_i, \tag{12}$$

$$w^h(Q^h, K^h) = Q^h {K^h}^T / \sqrt{d_k}. \tag{13}$$

At step t, we mask out the attention weights to the visited nodes C_{t-1} by setting $w^h_{:,C_{t-1}} = -\infty$ and do the following (the purple boxes in Figure 1):

$$A^h = \text{Attention}(Q^h, K^h, V^h) = \text{softmax}(w^h)V^h, \tag{14}$$

$$\text{Multihead}(Q, K, V) = \text{Concat}(A^1, A^2, ..., A^H)W_O. \tag{15}$$

Then Eqs. (4) and (5) engender the node embeddings, based on which each decoder selects a node to visit and gets a new mask. After that, while making a new decision about which node to visit, new node embeddings can be achieved by performing Eqs. (14), (15), (4) and (5) with the new mask.

The EG layer can be considered as correcting the node embeddings with the information that the visited nodes should be no longer relevant. However, running EG layer every step incurs additional computational overhead. To alleviate this, we perform EG layer every p steps. This is reasonable because the node embeddings change gradually and only one node is removed in a step. We found this technique consistently boosts the performance with little inference time. The EG layer can be viewed as a generalization of the step-wise idea in (Xin et al. 2020) with better computation efficiency. By choosing the hyper-parameter p, we can keep the number of times to re-embed with EG layer approximately constant for problems with different sizes.

Training

The MDAM structure and the training process is shown in Figure 1. The l-layer encoder consists of l-1 attention blocks and one EG layer which we have introduced before. For input instance x, each of the multiple decoders individually samples a trajectory π_m to get separate REINFORCE loss with the same greedy roll-out baseline from MDAM:

$$\nabla \mathcal{L}_{RL}(\theta|x) = \sum_m \underset{P_\theta^m(\pi_m|x)}{E} (L(\pi_m) - b(x)) \nabla \log P_\theta^m(\pi_m|x), \tag{16}$$

where $L(\pi_m)$ is the tour length. The baseline we adopt is similar to the one with the best performance in (Kool, van Hoof, and Welling 2019a). The model with the best set of parameters among previous epochs is used as the baseline model to greedily decode the result as the baseline $b(x)$.

$$b(x) = \min_m L(\pi'_m = \{y'_m(1), ..., y'_m(T)\}), \tag{17}$$

$$y'_m(t) = \arg\max_{y_t}(P_{\theta'}^m(y_t|x, y'_m(1), ..., y'_m(t-1))), \tag{18}$$

where θ is the current parameters of MDAM, θ' is the fixed parameters of baseline model from previous epoch, and m is the decoder index. We optimize the model by gradient descent:

$$\nabla \mathcal{L}(\theta) = \nabla \mathcal{L}_{RL}(\theta|x) - k_{KL} \nabla D_{KL}, \tag{19}$$

where k_{KL} is the coefficient of KL loss. Ideally, for each state encountered by each decoder, a KL loss should be computed during training to encourage diversity. To avoid expensive computation, we only impose KL loss on the first step, motivated by the following reasons. First, the initial state of all decoders is the same, i.e. an empty solution, where KL loss is meaningful and easy to compute since all decoders start from this state. Second, for the same instance, different optimal solutions usually have different options at the first step, which have strong impact on the construction patterns.

Experiments

In this section, we conduct experiments on six routing problems to verify the effectiveness of our method. Among them, TSP and CVRP are the most widely studied ones. TSP is defined as finding the shortest tour which visits each of the cities once and returns to the starting city, given the distances between each pair of the cities. CVRP generalizes TSP, where the starting city must be a depot and every other city has a demand to be served by the vehicles. Multiple routes could be planned in CVRP, each for a vehicle, which visits a subset of cities with total demands not exceeding the capacity of the vehicle. All the cities need to be covered by the routes. We follow existing works (Kool, van Hoof, and Welling 2019a; Nazari et al. 2018) to generate instances with 20, 50 and 100 nodes (cities), which use two-dimensional Euclidean distance to calculate the distance between two cities, and the objective is to minimize the total travel distance. The coordinates of the city locations are sampled from the uniform distribution ranging from 0 to 1 for both dimensions independently. For CVRP, the vehicle capacities are fixed as 30, 40, 50 for problems with 20, 50, 100 nodes (cities), respectively. And the demands of each non-depot city are sampled from integers $\{1...9\}$. Regarding the remaining four routing problems, i.e., Split Delivery Routing Problems (SDVRP), Orienteering Problem (OP) (Golden, Levy, and Vohra 1987), Prize Collecting TSP (PCTSP) (Balas 1989) and Stochastic PCTSP (SPCTSP), the settings follow the existing work (Kool, van Hoof, and Welling 2019a) and are introduced in the Supplementary Material. Note that these problems have their specific constraints and even random elements (SPCTSP). Nevertheless, MDAM is flexible enough to handle these properties by masking out the invalid nodes to visit at each step.

Hyperparameters. We embed the nodes with element-wise projection to 128-dimensional vectors. The Transformer encoder has 3 layers with 128 dimension features and 8 attention heads where the top one serves as the EG layer, and the hidden dimension of the fully connected layer (FF in Eq. (5)) is 512. We choose the number of decoders in MDAM to be 5, and each of the 5 decoders takes 128 dimension vectors and 8 heads attention. For EG layer, we set the number of steps between re-embeddings to be 2, 4, 8 for TSP20, 50, 100 and 2, 6, 8 for CVRP20, 50, 100 for faster evaluation. Following (Kool, van Hoof, and Welling 2019a), we train the model with 2,500 iterations per epoch and batch size 512 (except 256 for CVRP100 to fit GPU memory constraint) for 100 epochs. We use Adam Optimizer (Kingma and Ba 2015) with learning rate 10^{-4} for optimization. The coefficient of KL loss k_{KL} needs to be large enough to keep the diversity between different decoders but not too large to deteriorate the performance of each decoder. We set k_{KL} to 0.01 based on experiments on TSP20. Our code has been released [1].

Comparative Study

Here we compare our method (MDAM) with existing strong deep learning based models. For testing, we sample 10,000 instances from the same distributions used for training. To compute the optimality gap, we use the exact solver Concorde [2] to get the objective values of the optimal solutions for TSP. And for CVRP which is much harder to be solved exactly, we use the state-of-the-art heuristic solver LKH3 (Helsgaun 2017) to get the benchmark solutions by following (Kool, van Hoof, and Welling 2019a). For the beam search version of MDAM, we use 5 decoders each with beam size B=30 and 50 (denoted as bs30 and bs50), i.e. the whole beam size $\mathcal{B}$=150, and 250 respectively. Note that some methods (e.g. (Vinyals, Fortunato, and Jaitly 2015), (Dai et al. 2017), (Tsiligirides 1984), and (Deudon et al. 2018)) do not serve as baselines due to the reported inferior performance in (Kool, van Hoof, and Welling 2019a). We do not compare with L2I (Lu, Zhang, and Yang 2020), due to its prohibitively long computation time. Without any instance parallelization, the average inference time of MDAM with 50-width beam search on one CVRP100 instance is 6.7s, while L2I needs 24 minutes (Tesla T4). For the other four problems, we compare MDAM with AM and other strong baselines as in (Kool, van Hoof, and Welling 2019a), with details introduced in Supplementary Material.

The results are summarized in Table 1. We can observe that for almost all the testing sets of the six routing problems with 20, 50 and 100 nodes, MDAM outperforms AM (Kool, van Hoof, and Welling 2019a) significantly. To avoid prohibitively long computation time for large problems, exact solver Gurobi [3] is used as heuristics with time limits. On CVRP and SDVRP, our MDAM with greedy decoding strategy significantly outperforms existing greedy decoding models in (Nazari et al. 2018) and (Kool, van Hoof, and Welling 2019a), and the (standard) beam search version of RL (Nazari et al. 2018). With our beam search scheme, MDAM outperforms not only the sampling version of AM (Kool, van Hoof, and Welling 2019a), but also the improvement heuristic NeuRewriter (Chen and Tian 2019) on CVRP. For the comparison with traditional non-learning based methods, it is worth noting that MDAM outperforms sophisticated general purpose solvers OR Tools [4] and Gurobi (with time limits) on large instances (OP and PCTSP with 100 nodes), and shows relatively good scalability. For some problems such as OP with 50 nodes, MDAM outperforms highly specialized heuristic solvers.

For SPCTSP, optimization based methods (e.g. Gurobi, OR Tools and improvement heuristics) require some forms of online re-optimization (REOPT) (Kool, van Hoof, and

[1] https://github.com/liangxinedu/MDAM

[2] https://www.math.uwaterloo.ca/tsp/concorde

[3] https://www.gurobi.com

[4] https://developers.google.com/optimization/routing

	Method	n=20			n=50			n=100		
		Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
TSP	Concorde	3.84*	0.00%	1m	5.70*	0.00%	2m	7.76*	0.00%	3m
	AM greedy	3.85	0.34%	0s	5.80	1.76%	2s	8.12	4.53%	6s
	AM sampling	3.84	0.08%	5m	5.73	0.52%	24m	7.94	2.26%	1h
	MDAM greedy	3.84	0.05%	5s	5.73	0.62%	15s	7.93	2.19%	36s
	MDAM bs30	**3.84**	**0.00%**	2m	**5.70**	**0.04%**	7m	**7.80**	**0.48%**	20m
	MDAM bs50	**3.84**	**0.00%**	3m	**5.70**	**0.03%**	14m	**7.79**	**0.38%**	44m
CVRP	LKH	6.14*	0.00%	2h	10.38*	0.00%	7h	15.65*	0.00%	13h
	RL (beam 10)	6.40	4.39%	27m	11.15	7.46%	39m	16.96	8.39%	74m
	AM greedy	6.40	4.43%	1s	10.98	5.86%	3s	16.80	7.34%	8s
	AM sampling	6.25	1.91%	6m	10.62	2.40%	28m	16.23	3.72%	2h
	NeuRewriter	6.16	0.48%	22m	10.51	1.25%	35m	16.10	2.88%	66m
	MDAM greedy	6.24	1.79%	7s	10.74	3.47%	16s	16.40	4.86%	45s
	MDAM bs30	**6.14**	**0.26%**	3m	**10.50**	**1.18%**	9m	**16.03**	**2.49%**	31m
	MDAM bs50	**6.14**	**0.18%**	5m	**10.48**	**0.98%**	15m	**15.99**	**2.23%**	53m
SDVRP	RL (greedy)	6.51	5.77%	-	11.32	8.07%	-	17.12	7.17%	-
	RL (beam 10)	6.34	3.01%	-	11.08	5.78%	-	16.86	5.54%	-
	AM greedy	6.39	3.82%	1s	10.92	4.25%	4s	16.83	5.36%	11s
	AM sampling	6.25	1.55%	9m	10.59	1.10%	42m	16.27	1.85%	3h
	MDAM greedy	6.25	1.49%	13s	10.72	2.31%	28s	16.39	2.62%	1m
	MDAM bs30	**6.16**	**0.08%**	4m	**10.49**	**0.18%**	11m	**16.01**	**0.24%**	28m
	MDAM bs50	**6.15***	**0.00%**	6m	**10.47***	**0.00%**	19m	**15.97***	**0.00%**	1h
OP	Gurobi	5.39*	0.00%	16m		-			-	
	Gurobi (30s)	5.38	0.05%	14m	13.57	16.29%	2h	3.23	90.28%	3h
	Compass	5.37	0.36%	2m	16.17	0.25%	5m	33.19*	0.00%	15m
	AM greedy	5.19	3.64%	0s	15.64	3.52%	1s	31.62	4.75%	5s
	AM sampling	5.30	1.56%	4m	16.07	0.87%	16m	32.68	1.55%	53m
	MDAM greedy	5.32	1.32%	7s	15.92	1.80%	14s	32.32	2.61%	32s
	MDAM bs30	**5.38**	**0.15%**	1m	**16.19**	**0.10%**	6m	**32.91**	**0.84%**	14m
	MDAM bs50	**5.38**	**0.13%**	3m	**16.21***	**0.00%**	15m	**32.96**	**0.69%**	32m
PCTSP	Gurobi	3.13*	0.00%	2m		-			-	
	Gurobi (30s)	3.13*	0.00%	2m	4.48	0.03%	54m		-	
	OR Tools (60s)	3.13	0.01%	5h	4.48*	0.00%	5h	6.07	1.56%	5h
	ILS C++	3.16	0.77%	16m	4.50	0.36%	2h	5.98*	0.00%	12h
	AM greedy	3.18	1.62%	0s	4.60	2.66%	2s	6.25	4.46%	5s
	AM sampling	3.15	0.45%	5m	4.52	0.74%	19m	6.08	1.67%	1h
	MDAM greedy	3.16	0.82%	7s	4.56	1.73%	18s	6.17	3.13%	34s
	MDAM bs30	**3.14**	**0.21%**	2m	**4.50**	**0.55%**	9m	**6.07**	**1.46%**	16m
	MDAM bs50	**3.14**	**0.19%**	4m	**4.50**	**0.47%**	23m	**6.06**	**1.31%**	35m
SPCTSP	REOPT all	3.34	2.74%	17m	4.68	2.22%	2h	6.22	1.72%	12h
	REOPT half	3.31	1.82%	25m	4.64	1.35%	3h	6.16	0.74%	16h
	AM greedy	3.26	0.17%	0s	4.65	1.66%	2s	6.32	3.36%	5s
	AM sampling (half)	**3.25***	**0.00%**	15m	4.59	0.16%	37m	6.15	0.52%	92m
	MDAM greedy (half)	3.26	0.14%	24s	4.61	0.77%	53s	6.19	1.29%	102s
	MDAM bs10 (half)	3.25	0.11%	3m	**4.58***	**0.00%**	8m	**6.11***	**0.00%**	17m

Table 1: Multi-Decoder Attention Model (MDAM) vs Baselines (AM (Kool, van Hoof, and Welling 2019a), RL (Nazari et al. 2018), NeuRewriter (Chen and Tian 2019), Compass (Kobeaga, Merino, and Lozano 2018), ILS (Lourenço, Martin, and Stützle 2003)). Note: We evaluate our model using a single RTX-2080Ti GPU. AM sampling samples 1,280 solutions. We run Concorde and LKH in parallel for 32 instances on a 32 virtual CPU system (2Xeon E5-2620). Other results come from the original papers. The runtimes are reported for solving 10,000 test instances following (Kool, van Hoof, and Welling 2019a). All problems aim to minimize the objectives except OP, which aims to maximize the prizes collected along the tour. Bold is the best among learning based methods, while '*' is the best in all methods.

	Method	n=20 Obj	n=20 Gap	n=20 Time	n=50 Obj	n=50 Gap	n=50 Time	n=100 Obj	n=100 Gap	n=100 Time
TSP	MDAM (no MD) greedy	3.85	0.27%	1s	5.78	1.52%	3s	8.06	3.76%	7s
	MDAM (no EG) greedy	3.84	0.06%	3s	5.74	0.71%	8s	7.98	2.78%	23s
	MDAM greedy	3.84	0.05%	5s	5.73	0.62%	15s	7.93	2.19%	36s
CVRP	MDAM (no MD) greedy	6.39	4.27%	1s	10.93	5.32%	3s	16.57	5.96%	9s
	MDAM (no EG) greedy	6.25	1.95%	4s	10.79	3.95%	11s	16.46	5.26%	26s
	MDAM greedy	6.24	1.79%	7s	10.74	3.47%	16s	16.40	4.86%	45s

Table 2: MDAM Structure Ablation Results

Welling 2019a) to adapt to the dynamic changes (e.g. constraint violation). Similarly, for AM sampling and MDAM, we iteratively plan the tour based on the expected prize of the unvisited nodes and execute half of it. We use beam size 10 for MDAM searching as increasing the beam size does not significantly improve the performance and fast inference time is important for online settings.

In terms of efficiency, though the baselines were executed on different machines, the computation time of MDAM is well accepted compared with existing deep learning based methods, especially considering the significant performance boost. Though MDAM (and all deep models) could be slower than the highly specialized solver on some problems (e.g. Concorde on TSP), it is generally much faster than the traditional algorithms with comparable solution quality.

Ablation Study

We further evaluate the effectiveness of different components in our model taking TSP and CVRP as the testbed. We assess the contribution of the Multi-Decoder structure (MD) and Embedding Glimpse layer (EG) to the quality of greedy construction policy, based on ablation study. We omit beam search here since it is an independent technique applicable to a given greedy decoding policy. The results are summarized in Table 2. We can observe that both MD and EG consistently improve the quality of learned construction policy for all instance sets, which well verifies the rationale of our design. While MD boosts the performance significantly with relatively longer inference time, improvement by EG is relatively small but with little additional computation overhead.

Analysis of MDAM

To demonstrate the usefulness of the distinct construction patterns learned by each decoder, we evaluate the performance of each decoder on CVRP20 with greedy decoding and beam search (B=50). Figure 3 reports the number of times that each decoder finds the best (winning) and strictly best (solely winning) solution among all the 5 decoders in both the greedy decoding and search mode. We can see that it is common that more than one decoders find the best solution, since the winning times are higher than the solely winning times in both greedy decoding and beam search modes. More importantly, all decoders perform similarly since no one is dominant or not contributing at all, showing that each of them is indeed effective in the solving process.

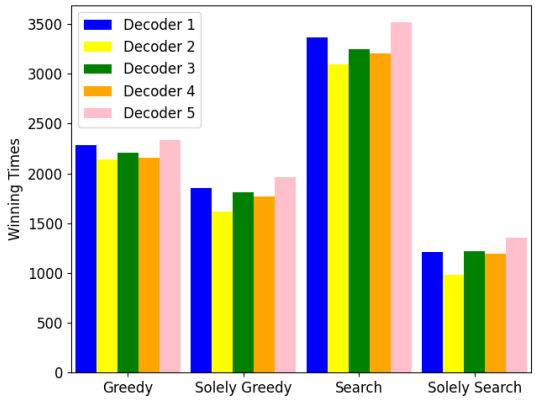

Figure 3: Greedy Decoding and Beam Search (B=50) Results of Each Decoder on CVRP20

In the Supplementary Material, we present more analysis of MDAM, including the effectiveness of the merging technique we designed for the customized beam search, and the impact of different number of decoders in the beam search mode. We also compare with a stronger version of AM to further prove the effectiveness of MDAM. More specifically, we tune the temperature hyperparameter of the softmax output function of AM sampling to provide more diverse solutions. Finally, we show that comparing with AM, our MDAM has better generalization ability on larger problems such as CVRP with 150 and 200 nodes.

Conclusions and Future Work

In this paper, we propose a novel model to learn construction heuristics for routing problems, which is trained by reinforcement learning. It employs a multi-decoder structure to learn distinct construction patterns, which are further exploited by a customized beam search scheme. An Embedding Glimpse layer is incorporated in the model, which empowers the decoders with more informative embeddings. Our method outperforms state-of-the-art deep learning based methods on six routing problems, and engenders solutions close to traditional highly optimized solvers with reasonable time. In the future, we plan to improve our model by allowing flexibility in the number of decoders, and enabling collaborations among the decoders to make decisions jointly instead of individually.

Acknowledgments

The research was supported by the ST Engineering-NTU Corporate Lab through the NRF corporate lab@university scheme. Part of this research was conducted at Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU), which is a collaboration between Singapore Telecommunications Limited (Singtel) and Nanyang Technological University (NTU) that is funded by the Singapore Government through the Industry Alignment Fund - Industry Collaboration Projects Grant. Wen Song was partially supported by the Young Scholar Future Plan of Shandong University (Grant No. 62420089964188). Zhiguang Cao was partially supported by the National Natural Science Foundation of China (61803104).

References

Applegate, D. L.; Bixby, R. E.; Chvatal, V.; and Cook, W. J. 2006. *The traveling salesman problem: a computational study*. Princeton university press.

Balas, E. 1989. The prize collecting traveling salesman problem. *Networks* 19(6): 621–636.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2017. Neural combinatorial optimization with reinforcement learning. In *Proceedings of International Conference on Learning Representations (ICLR) Workshop.*

Chen, X.; and Tian, Y. 2019. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, 6278–6289.

Dai, H.; Dai, B.; and Song, L. 2016. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, 2702–2711.

Dai, H.; Khalil, E.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, 6348–6358.

Deudon, M.; Cournut, P.; Lacoste, A.; Adulyasak, Y.; and Rousseau, L.-M. 2018. Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, 170–181. Springer.

Fischetti, M.; Toth, P.; and Vigo, D. 1994. A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research* 42(5): 846–859.

Golden, B. L.; Levy, L.; and Vohra, R. 1987. The orienteering problem. *Naval Research Logistics (NRL)* 34(3): 307–318.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

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

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8): 1735–1780.

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of International Conference on Machine Learning*, 448–456.

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations (ICLR).*

Kobeaga, G.; Merino, M.; and Lozano, J. A. 2018. An efficient evolutionary algorithm for the orienteering problem. *Computers & Operations Research* 90: 42–59.

Kool, W.; van Hoof, H.; and Welling, M. 2019a. Attention, Learn to Solve Routing Problems! In *Proceedings of International Conference on Learning Representations (ICLR).*

Kool, W.; Van Hoof, H.; and Welling, M. 2019b. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International Conference on Machine Learning*, 3499–3508. PMLR.

Lourenço, H. R.; Martin, O. C.; and Stützle, T. 2003. Iterated local search. In *Handbook of metaheuristics*, 320–353. Springer.

Lu, H.; Zhang, X.; and Yang, S. 2020. A Learning-based Iterative Method for Solving Vehicle Routing Problems. In *Proceedings of International Conference on Learning Representations (ICLR).*

Nazari, M.; Oroojlooy, A.; Snyder, L.; and Takác, M. 2018. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, 9839–9849.

Schroff, F.; Kalenichenko, D.; and Philbin, J. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 815–823.

Toth, P.; and Vigo, D. 2014. *Vehicle routing: problems, methods, and applications*. SIAM.

Tsiligirides, T. 1984. Heuristic methods applied to orienteering. *Journal of the Operational Research Society* 35(9): 797–809.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, 2692–2700.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4): 229–256.

Xin, L.; Song, W.; Cao, Z.; and Zhang, J. 2020. Step-wise Deep Learning Models for Solving Routing Problems. *IEEE Transactions on Industrial Informatics* 1–1. doi:10.1109/TII.2020.3031409.