AN ATTENTION-LSTM HYBRID MODEL FOR THE COORDINATED ROUTING OF MULTIPLE VEHICLES

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning has recently shown promise in learning quality solutions in a number of combinatorial optimization problems. In particular, the attention-based encoder-decoder models show high effectiveness on various routing problems, including the Traveling Salesman Problem (TSP). Unfortunately, they perform poorly for the TSP with Drones (TSP-D), requiring routing a heterogeneous fleet of vehicles in coordination. In TSP-D, two different types of vehicles are moving in tandem and may need to wait at a node for the other vehicle to join. Stateless attention-based decoder fails to make such coordination between vehicles. We propose an attention encoder-LSTM decoder hybrid model, in which the decoder's hidden state can represent the sequence of actions made. We empirically demonstrate that such a hybrid model improves upon a purely attention-based model for both solution quality and computational efficiency. Our experiments on the min-max Capacitated Vehicle Routing Problem (mmCVRP) also confirm that the hybrid model is more suitable for coordinated routing of multiple vehicles than the attention-based model.

1 INTRODUCTION

Recently, reinforcement learning has yielded promising results on various combinatorial optimization problems (Bengio et al., 2020). *Routing problems* are important combinatorial optimization problems with diverse practical applications, including delivery and courier services, ride-sharing, and logistics. Deep reinforcement learning-based approaches have also been shown to work well on routing problems (Vinyals et al., 2015; Bello et al., 2016; Khalil et al., 2017; Nazari et al., 2018; Kool et al., 2018) such as the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP).

These deep reinforcement learning-based approaches are attractive not only because they show good performance on various benchmarks, but also because they can be easily adapted for new problems. As their name suggests, deep reinforcement learning (Mnih et al., 2013) methods are based on deep neural networks (LeCun et al., 2015). Deep neural networks tend to require less problem-specific engineering, and thus the same neural network architecture is often observed to perform well across many tasks. In contrast, traditionally best-performing routing algorithms are highly specific to a problem they are designed to solve, and it is nontrivial to extend them for an application to a new problem.

Indeed, Kool et al. (2018) have shown that the same attention-based encoder-decoder model performs well across six routing problems, including TSP and CVRP with little problem-specific modifications on the model. Their Attention Model (AM) is a variant of the Transformer model (Vaswani et al., 2017), which has been particularly successful across domains, demonstrating high performance across many natural language processing (Devlin et al., 2018; Raffel et al., 2019) as well as computer vision (Dosovitskiy et al., 2020) tasks.

In this paper, we demonstrate that AM loses its strong competency in routing *multiple vehicles in coordination*, although AM is the state-of-the-art *end-to-end* learning method for single-vehicle routing problems. While AM is shown to be effective for CVRP, which involves multiple *routes*, it still only considers a single vehicle; the objective of CVRP—to minimize the total distance traveled—enables AM to route a single vehicle multiple times to obtain multiple routes. Hence, no coordination among multiple vehicles is considered in AM. We consider routing problems to

minimize the *makespan*, the time when the last vehicle returns to the depot. Such problems require coordination among vehicles. Our aim is to devise an *end-to-end* reinforcement learning approach that can learn heuristic solutions by considering interdependence among vehicles.

1.1 The Traveling Salesman Problem with Drone (TSP-D)

In particular, we consider the Traveling Salesman Problem with Drone, or TSP-D (Agatz et al., 2018), where a drone and a truck are routed together to deliver customer orders. While the truck visits customers, the drone can also deliver a customer, flying off from and back to the truck. The recent progress in drone technology has made the routing of heterogeneous vehicles a problem of practical importance. A tandem of the drone and truck allows reducing transportation costs, where the high speed of drones combined with the large storage capacity of trucks enables to efficiently serve customers (Macrina et al., 2020). Similarly, the combination of drones and ships/planes are used for surveillance and rescue operations to detect targets (Otto et al., 2018). TSP-D is not just limited to the truck-drone combination, but also applicable or extensible to coordinated routing of other heterogeneous vehicles, such as a truck and any mobile robot for urban delivery and a human order-picker and a mobile robot assistant in warehouses. See Chung et al. (2020) for other related drone routing problems, such as the flying sidekick TSP (Murray & Chu, 2015).

The development of efficient computational methods for TSP-D (Agatz et al., 2018; Poikonen et al., 2019; Roberti & Ruthmair, 2021; Vásquez et al., 2021) is still in its infancy, while highly efficient exact and heuristic optimization methods exist for TSP. Exact optimization approaches for TSP-D suffer from long computational time and are typically limited to smaller than 40-node problems. Some heuristic optimization approaches can also take several minutes to a few hours to produce quality solutions for large-scale problems. In this paper, we aim to develop a reinforcement learning approach for TSP-D, which is as competitive as fast heuristic optimization methods.

The main challenge of TSP-D stems from the *simultaneous* decision-making for a heterogeneous fleet of vehicles and strong *interdependency* among them. Unlike TSP or CVRP, where the decision is made for only one vehicle for a tour or multiple sequential tours, TSP-D requires route decisions for two different vehicles in coordination. The existing literature in reinforcement learning to route a single agent or a fleet of homogeneous vehicles in a multi-agent setting does not fit the nature of such heterogeneous vehicle routing problems. For instance, models to route a single truck or drone only handle cases when there is a single agent which interacts with an environment and does not generalize into the presence of several vehicles in the same environment. In contrast, in routing a heterogeneous fleet of vehicles, the interdependency among vehicles is a critical consideration since the capacity of one vehicle to fulfill customer orders is dependent on the interaction with the other vehicle. This, in effect, has a strong impact on the routing decisions of both vehicles.

1.2 Contributions

In addressing the above-outlined challenges, our main contributions are two-fold. First, we provide TSP-D as a Markov Decision Process (MDP) so that reinforcement learning can be developed and applied. While MDP formulations for TSP only involve customer nodes, an MDP formulation for TSP-D should be able to capture the in-transit status of vehicles and remaining times to reach the next node, to express the state spaces of coordinated drone-truck routing adequately. Our MDP formulation is comprehensive and distinct from the dynamic programming (DP) approach of Bouman et al. (2018), which combines three DP problems sequentially to obtain the final formulation. This makes their DP formulation hard to use in end-to-end reinforcement learning approaches.

Second, we present an *Attention-LSTM Hybrid Model* (HM) for efficient routing of multiple vehicles taking into account required interactions between vehicles. Our HM consists of an attention-based encoder to encode a highly connected graph and an LSTM-based decoder that stores the routing history of all vehicles. Using a single decoder to route all vehicles allows passing information about the routing decisions of vehicles to each other, thus promoting efficient interaction. In training the hybrid model, we rely on a central controller, which observes an entire graph to route all vehicles.

We show that HM outperforms the AM consistently and provides a flexible end-to-end framework, as competitive as existing heuristic methods for TSP-D. We demonstrate the effectiveness of the proposed approach through computational experiments using both synthetic and real data. We also

test HM and compare it with AM in solving the min-max CVRP (mmCVRP), whose objective is to minimize the makespan by routing multiple capacitated vehicles to serve all customers. TSP-D and mmCVRP share a similar objective function form, and simultaneous routing becomes critical for vehicle coordination. We show that HM has the potential to be effective in coordinated routing problems in various applications.

2 The Problem Definition and Formulation

We define TSP-D formally and propose a Markov Decision Process (MDP) formulation.

2.1 TSP-D Defined

In TSP-D, we consider a complete graph $\mathcal{G}$ consisting of the set of nodes $\mathcal{N} = \{1, 2, ..., N\}$. Node 1 is a depot representing a warehouse, where a single truck equipped with a single drone is loaded with customer orders. Then truck and drone must serve $N-1$ customers whose locations are known. When the distance from node i to node j is $d_{i,j}$, the traverse time from node i to node j for truck and drone is denoted by $\tau^{\text{tr}}_{i,j}$ and $\tau^{\text{dr}}_{i,j}$, respectively, where we assume $\tau^{\text{dr}}_{i,j} \leq \tau^{\text{tr}}_{i,j}$ for all $i, j \in \mathcal{N}$. Without loss of generality, we let the speed of truck and drone be 1 and $\alpha \geq 1$, respectively, so that $d_{i,j} = \tau^{\text{tr}}_{i,j} = \alpha\tau^{\text{dr}}_{i,j}$. A typical value of α used in the literature is 2.

The objective of TSP-D is to complete serving all customers and return to the depot in a minimal time, while the truck and the drone are subjected to joint routing constraints. The cost of each TSP-D solution is the makespan, not the total travel time. Each customer can be served either by a truck or a drone. We allow the drone to serve only a single customer per launch due to the limited delivery capacity. For simplicity, we allow the drone to fly for an unlimited distance, which is a standard assumption. Only customer nodes and the depot can be used to launch, recharge, and load the drone. Figures E.6 and E.7 show examples of TSP-D routes. Figure 1 shows a simple example and explains the MDP formulation proposed in the next section.

2.2 An MDP Formulation of TSP-D

To route both the drone and the truck in a shared urban environment, we assume that a central controller observes the entire graph and makes routing decisions for both the drone and the truck. We formalize this problem as a Markov Decision Process (MDP) with discrete steps.

The state of the MDP is represented by the following variables. First, $\mathcal{V}_t \subseteq \mathcal{N}$ represents the set of nodes visited by any vehicle up to step t. c^{tr}_t and c^{dr}_t denote the current destination node of the truck and the drone respectively. If the truck (or the drone) is in transit between two nodes at step t, then c^{tr}_t (or c^{dr}_t) is set as the destination node of the vehicle. Or, if the truck (or the drone) is exactly located at a node, c^{tr}_t (or c^{dr}_t) is set as the node index. By design, as we explain below, at least one vehicle will be located at a node at each step. r^{tr}_t and r^{dr}_t represent the remaining times to arrive at destinations c^{tr}_t and c^{dr}_t respectively. When the truck (or the drone) is located at a node, $r^{\text{tr}}_t = 0$ (or $r^{\text{dr}}_t = 0$). Then, $\mathbf{s}_t := (\mathcal{V}_t, c^{\text{tr}}_t, c^{\text{dr}}_t, r^{\text{tr}}_t, r^{\text{dr}}_t)$ represents the state. At step 0, vehicles are located at node 1. Therefore, $\mathcal{V}_0 = \{1\}$, $c^{\text{tr}}_0 = 1$, $c^{\text{dr}}_0 = 1$, $r^{\text{tr}}_t = 0$, $r^{\text{dr}}_t = 0$ and $\mathbf{s}_0 = (\{1\}, 1, 1, 0, 0)$.

At each step, the central controller determines the next destination of the truck $a^{\text{tr}}_t \in \mathcal{N}$ and that of the drone $a^{\text{dr}}_t \in \mathcal{N}$. We cannot change the destination of a vehicle which is in transit from one node to another. When the drone is in transit, i.e. $r^{\text{dr}}_t > 0$, we restrict $a^{\text{dr}}_t = c^{\text{dr}}_t$. Analogously, $a^{\text{tr}}_t = c^{\text{tr}}_t$ if the truck is in transit. For convenience, we refer to $\mathbf{a}_t := (a^{\text{tr}}_t, a^{\text{dr}}_t)$ as the action at step t.

Now let us discuss the state dynamics. The next step occurs when either of the vehicles arrives at a node. Let $\widehat{r}^{\text{tr}}_t = r^{\text{tr}}_t + \tau_{c^{\text{tr}}_t, a^{\text{tr}}_t}$ and $\widehat{r}^{\text{dr}}_t = r^{\text{dr}}_t + \tau_{c^{\text{dr}}_t, a^{\text{dr}}_t}$ be "updated" versions of r^{tr}_t and r^{dr}_t right after the action is selected, with the convention $\tau_{i,i} = 0$ for any $i \in \mathcal{N}$. Then, the time taken until the next step, which happens when at least one vehicle arrives at the destination, will be the minimum of the two. We denote this as $C_t := \min(\widehat{r}^{\text{tr}}_t, \widehat{r}^{\text{dr}}_t)$. The remaining time is updated as $r^{\text{tr}}_{t+1} = \widehat{r}^{\text{tr}}_t - C_t$, $r^{\text{dr}}_{t+1} = \widehat{r}^{\text{dr}}_t - C_t$. The set of visited nodes is updated as $\mathcal{V}_{t+1} = \mathcal{V}_t \cup \{a^v_t \mid r^v_{t+1} = 0, v \in \{\text{tr}, \text{dr}\}\}$. Also, $c^v_{t+1} = a^v_t$ for $v \in \{\text{tr}, \text{dr}\}$.

Let T be the index of the step when both the drone and the truck return back to the depot after serving all customers. Technically, we can define the value of T as infinity if the controller fails to serve all

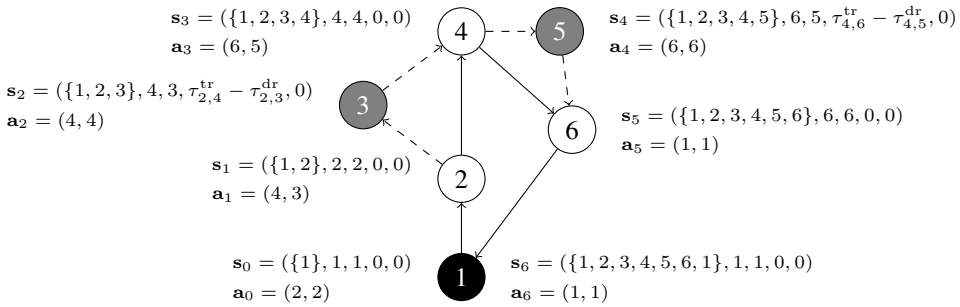


Figure 1: An example of TSP-D with state $\mathbf{s}_t := (\mathcal{V}_t, c_t^{\text{tr}}, c_t^{\text{dr}}, r_t^{\text{tr}}, r_t^{\text{dr}})$ and action $\mathbf{a}_t = (a_t^{\text{tr}}, a_t^{\text{dr}})$ on six nodes: black, grey and white nodes represent a depot, customers served by drone and customers served by truck respectively. Solid and dashed lines correspond to drive and fly arcs respectively. The drone and the truck traverse together from the depot to the first customer. Then drone is launched to serve the second customer, while the truck traverses to serve the third customer. At this node, the drone reunites with the truck to be launched to serve the next customer. Drone and truck reunite after at the last customer location and traverse back together to the depot.

customers and return vehicles. Then, our cost function (negative reward) is the total time spent in the system, or the makespan:

$$C = \sum_{t=0}^{T} C_t. \tag{1}$$

3 The Attention-LSTM Hybrid Model

Following Vinyals et al. (2015), we aim to learn a probabilistic routing policy π_θ as a product of conditional probabilities

$$\pi_\theta(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_T | \mathcal{G}) = \prod_{t=1}^{T} p_\theta(\mathbf{a}_t | \mathbf{s}_t), \tag{2}$$

where p_θ is a function parameterized by θ. Then, we employ policy gradient methods (Williams, 1992) to learn θ.

A series of research (Vinyals et al., 2015; Khalil et al., 2017; Nazari et al., 2018; Kool et al., 2018) has shown that using neural networks for modeling p_θ is a viable approach. In particular, Kool et al. (2018) demonstrated that AM, which leverages several layers of multi-head attention to learn representations of nodes, outperforms previous models, which rely on simpler representation models, for example, the model from Nazari et al. (2018) which uses a single layer of single-head attention. This observation was consistent across several classes of routing problems and also with findings in natural language processing (Vaswani et al., 2017; Devlin et al., 2018).

At each step of prediction, however, the decoder of AM conditions only on the current location of the vehicle and the current state of nodes and ignores the sequence of actions it has made in the past (see Appendix B for details). Such conditional independence is sensible for single-vehicle routing problems in which the ordering of past actions is irrelevant for future actions. However, we argue it is not well-suited for problems like TSP-D requiring coordination among multiple vehicles. For such coordination, we propose a hybrid model (HM) that also uses multi-head attention for encoding as in AM but leverages LSTM hidden states for decoding in order to address dependencies between subsequent actions. Since in TSP-D, the relative locations of the drone and the truck is a key piece of information for coordinated routing, LSTM's ability to store past decisions in its hidden states will be more effective than stateless attention-based decoders. The model by Nazari et al. (2018) also utilizes an LSTM-based decoder for single-vehicle routing problems such as CVRP, but AM outperforms the Nazari et al. Model (NM). We argue that our HM is suitable for coordinated routing by combining AM's encoder and NM's decoder with additional features.

The overall structure of HM is shown in Figure 2. We formally describe HM below.

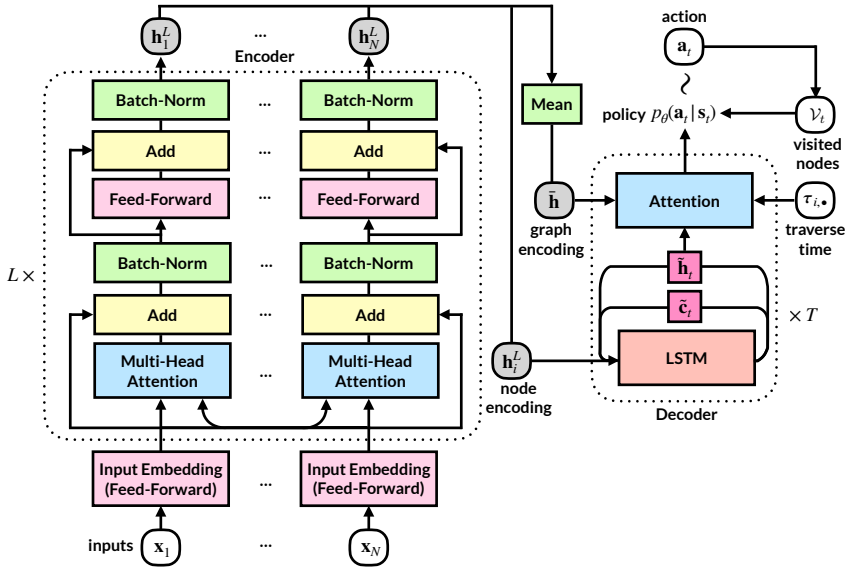


Figure 2: An Encoder-Decoder structure of Attention-LSTM Hybrid model (HM).

Encoder Given a graph with a set of nodes $\mathcal{N}$, we have coordinates of each node in $\mathbb{R}^2$. Let $\mathbf{x}_n = (x_n, y_n) \in \mathbb{R}^2$ represent coordinates for node $n \in \mathcal{N}$. To embed such a fully connected graph we start from the initial embeddings of the nodes using linear transformation:

$$\mathbf{h}_n^0 = \mathbf{W}^0 \mathbf{x}_n + \mathbf{b}^0 \quad \forall n \in \mathcal{N}, \tag{3}$$

where $\mathbf{W}^0$ and $\mathbf{b}^0$ are learnable parameters. These initial embeddings of nodes $\{\mathbf{h}_n^0 : n \in \mathcal{N}\}$ are then passed through L number of attention layers. Each attention layer, l, consists of two sublayers: a multi-head attention layer and a fully connected feed-forward layer, described in Appendix A.

Decoder Given the encoder outputs as embeddings of each node in the graph, we denote by $\bar{\mathbf{h}}$ the graph encoding computed as the mean of the embeddings of all the nodes in a graph. Then to select an action for a decision taker (either the drone or truck), we pass to the LSTM the embedding of the last node selected by the decision taker denoted as $\mathbf{h}_i^L$ where i represents the current location:

$$\widetilde{\mathbf{h}}'_{t+1}, \widetilde{\mathbf{c}}'_{t+1} = \text{LSTM}(\mathbf{h}_i^L, (\widetilde{\mathbf{h}}_t, \widetilde{\mathbf{c}}_t)). \tag{4}$$

Here $\widetilde{\mathbf{h}}_t$ and $\widetilde{\mathbf{c}}_t$ correspond to the hidden and cell states at time t. We apply dropout to the output of LSTM with probability p:

$$\widetilde{\mathbf{h}}_{t+1} = \text{Dropout}(\widetilde{\mathbf{h}}'_{t+1}, p), \quad \widetilde{\mathbf{c}}_{t+1} = \text{Dropout}(\widetilde{\mathbf{c}}'_{t+1}, p). \tag{5}$$

Then we pass the hidden state of the LSTM to attention along with the graph embedding and the traveling time from the current location of a decision taker, similar to Bogyrbayeva et al. (2021), to speed up training and improve the performance. The attention vector is calculated as follows:

$$\mathbf{a}_{i,\cdot} = \mathbf{v}_a^\top \tanh\left(\mathbf{W}^a[\bar{\mathbf{h}}; \widetilde{\mathbf{h}}_{t+1}; \mathbf{W}^d \boldsymbol{\tau}_{i,\cdot}]\right), \tag{6}$$

where i represents the current location of a decision taker, $\boldsymbol{\tau}_{i,\cdot}$ is the vector of the traverse times from the current node to all other nodes in a graph, and $\mathbf{v}_a, \mathbf{W}^a$, and $\mathbf{W}^d$ are trainable parameters with dimensions d_h. The resulting attention vector is $\mathbf{a}_{i,\cdot}$ and $a_{i,j}$ denotes its element. Then the attention is passed through softmax to produce the probabilities of visiting the next node as follows:

$$p_\theta(a_t^v = j|\mathbf{s}_t) = \begin{cases} \dfrac{\exp(a_{i,j})}{\sum_{j' \in \mathcal{N} \setminus \mathcal{V}_t} \exp(a_{i,j'})} & : j \in \mathcal{N} \setminus \mathcal{V}_t, \\ 0 & : j \in \mathcal{V}_t, \end{cases} \tag{7}$$

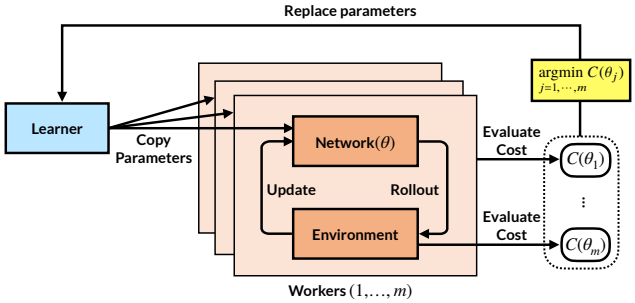


Figure 3: Overview of distributed RL training for HM: each of multiple workers executes rollout with different training instances, updates its parameters, and evaluates costs. The learner replaces its network parameters with the ones from the best worker.

so that we do not allow nodes to be revisited by either vehicle. Conceptually, RL policies shall learn the constraint if it is needed. However, we empirically found from preliminary experiments that preventing vehicles from revisiting nodes improves the efficiency of training and does not degrade the quality of final solutions. Hence, we disallow revisiting in all our experiments. See Appendix E.2 for further discussion on revisiting.

Distributed RL Training The proposed model determines the probability distribution, $\pi_\theta(\mathcal{V}_T|\mathcal{G})$, which, given a graph, produces the sequence of nodes to be visited by the drone and truck. In TSP-D, we aim to minimize the makespan C in (1). We define the training objective function as follows:

$$J(\theta|\mathcal{G}) = \mathbb{E}_{\pi_\theta(\mathcal{V}_T|\mathcal{G})}[(C - b(\mathcal{G})) \log \pi_\theta(\mathcal{V}_T|\mathcal{G})], \tag{8}$$

where $b(\mathcal{G})$ is a baseline for variance reduction. We use a critic network to estimate $b(\mathcal{G})$, while an actor network is used to learn the policy π_θ; hence we can compute the gradients $\nabla_\theta J(\theta|\mathcal{G})$ using the REINFORCE algorithm (Williams, 1992).

To accelerate the training of HM, we can take the advantage of parallelization. We modified the A2C algorithm to obtain *Distribute-and-Follow Policy Gradient* (DFPG) to stabilize the final performance of the proposed method at the convergence point (see Figure C.5). To achieve scalable efficiency and reduced variance, DFPG selects gradients of the best performing worker when updating the central learner's weights as shown in Algorithm C.1. We generate multiple parallel workers of an RL agent with the same network parameters copied from the central learner. Each worker performs a rollout with a batch of episodes of different data instances. Then the network parameters of the workers are updated by gradient using the advantage function as a baseline (see Algorithm C.2). Next, we evaluate the updated network parameters of each worker with the shared validation instances. Finally, we select the worker whose performance is the best among the other workers in a greedy manner where the central learner copies the network parameters from the best worker as shown in Figure 3.

We observed that DFPG accelerates the training of AM, but only a slight improvement over AM trained with REINFORCE is shown at the final epoch. Hence, we apply the REINFORCE algorithm for training AM as suggested in Kool et al. (2018) for a fair comparison.

4 Computational Experiments

We empirically demonstrate the strength of HM against AM and strong heuristics for TSP-D.

4.1 Training and Evaluation Configurations

Data Generation. We generate two sets of datasets for TSP-D from different types of locations. In *random locations* dataset, we randomly sample x and y coordinates of each node from a uniform distribution over $[1, 100] \times [1, 100]$, with the exception of the depot node which is distributed over $[0, 1] \times [0, 1]$. This makes the depot always located at the corner. While such a uniform sampling

is standard in the TSP-D literature, we create a new dataset named *Amsterdam* for more realistic experiments, from the dataset originally used in Haider et al. (2019). See Appendix D for the details.[1]

Baselines. Even though there is a surge of studies proposing the combination of learning methods with optimization heuristics to solve routing problems outlined in Mazyavkina et al. (2021), to analyze our neural architecture choice for HM, we compare against AM (Kool et al., 2018), which is the state-of-the-art *end-to-end* learning algorithm for routing problems. Also, since our HM is built upon AM's encoder, AM is a natural choice for comparison. We also compare against heuristic optimization algorithms from the operation research literature, namely 'TSP-ep-all' (Agatz et al., 2018) and 'divide-and-conquer heuristic' (DCH) (Poikonen et al., 2019). The TSP-ep-all heuristic starts with an initial TSP tour, then partitions the TSP tour into nodes to be served by drone and truck, followed by local search and exchange algorithms for the TSP tour and subsequent partitionings. On the other hand, the DCH of Poikonen et al. (2019) splits the TSP tour into several subgroups and partition nodes in each subgroup using a branch-and-bound algorithm. When we use DCH, we apply TSP-ep-all to partition each subgroup instead of branch-and-bound. In particular, we use DCH/g, which divides all nodes into subgroups so that each subgroup has g nodes. While $g = 10$ was originally used in Poikonen et al. (2019), we also test with $g = 25$. As g increases the solution time and quality increase. Note that, when $g = N$, the DCH/N is identical to TSP-ep-all. We implemented TSP-ep-all and DCH/g in Julia 1.5 (Bezanson et al., 2017), while the initial TSP tours are obtained by the Concorde TSP Solver (Applegate et al., 2001).

Decoding Strategies. There are two methods by which we sample solutions from the trained hybrid model. In the first method called *greedy*, we always select nodes with the highest probabilities to visit at each time step. In the second method, called *sampling*, we sample multiple solutions independently from the trained model according to (2). Then, we select the minimal-cost sample as the solution.

Solution times. We measure the inference time of the benchmark dataset using an Nvidia A100 GPU and AMD EPYC 7452 32-Core Processor for HM and AM. For heuristic methods, Intel Xeon E5-2630 2.2 GHz CPU is used. Each TSP-D benchmark dataset for each size consists of 100 instances. For HM greedy and AM greedy, instances are evaluated not in a batch but one by one in a sequence on the GPU. Heuristic methods use the same strategy but using a single thread of the CPU instead of a GPU. For HM sampling, denoted by HM (s), we make s copies of each instance and sample solutions for all s copies in a batch on the GPU. AM sampling is executed in the same way as HM sampling. AM (4800), however, could not run in a full batch for $N = 100$ in Table 1 on a single GPU with the memory of 40GB. Therefore, we split the sample size of 4800 into 4600 and 200 to obtain AM (4800) results. For $N = 100$, because TSP-ep-all takes more than 3 hours for each instance, we do not report the results. Note that we do not intend to compare the solutions times of the heuristics and the learning methods directly; rather, we aim to understand the trends. We report the average time for a single instance.

Other experiment details, including hyperparameters, are provided in Section D.

4.2 Results on TSP-D

We evaluate HM on 10 instances of 11-node graphs from Agatz et al. (2018), for which optimal solutions are already known. Although HM generates solutions sequentially without backtracking, it can find routes that are close to optimal routes. The greedy decoding from HM shows a 0.93% optimality gap, while the sampling strategy reduces the optimality gap to 0.69%. Figures E.6 and E.7 visualize the optimal solutions reported in the literature and the solution produced by HM with greedy decoding.

HM outperforms AM consistently by a large margin, and this trend signifies as the problem size increases. For example, for $N = 100$ in the random location datasets shown in Table 1, HM (greedy) has the gap of 3.64%, when AM (greedy) has the gap of 18.03% when compared to the best performing method. Figures 4(a)–(c) show the increasing advantage of HM (greedy) over AM (greedy) as N grows. This demonstrates that recurrent hidden states we introduce in HM are crucial for complex coordination between vehicles that larger problem instances require.

HM provides superb scalability as well as solid performance, comparable to or exceeding the best heuristic method. While the performance and speed of HM (greedy) lie between DCH/10 and

[1]The code will be shared with the public after the review process.

Table 1: TSP-D results on Random locations dataset. Averages and standard deviation of 100 problem instances. 'Cost' refers to the average cost value (1), where the small numbers represent the standard deviation. 'Gap' is the relative difference to the average cost of the best algorithm for the setting (23). 'Time' is the average solution time of the algorithm for a single instance.

	$N = 20$			$N = 50$			$N = 100$		
Method	Cost	Gap	Time	Cost	Gap	Time	Cost	Gap	Time
TSP-ep-all	281.62±18.05	0.02%	(0.8s)	397.21±20.19	0.43%	(4m)	-	-	(>3h)
DCH/10	292.23±19.00	3.79%	(0.1s)	420.51±23.98	6.32%	(0.2s)	570.74±20.61	4.80%	(0.3s)
DCH/25	-	-		404.78±22.03	3.57%	(6.0s)	548.23±22.36	0.66%	(11.0s)
AM (greedy)	294.88±24.44	4.74%	(0.1s)	439.21±28.09	11.01%	(0.2s)	642.82±25.12	18.03%	(0.5s)
HM (greedy)	285.59±18.24	1.44%	(0.1s)	408.51±18.78	3.29%	(0.3s)	564.42±22.03	3.64%	(0.5s)
AM (4800)	285.06±18.14	1.25%	(1.0s)	411.50±20.02	4.01%	(4.0s)	603.46±19.52	10.82%	(11.2s)
HM (100)	282.65±17.71	0.40%	(0.1s)	399.11±17.59	0.87%	(0.4s)	550.01±19.49	1.00%	(0.9s)
HM (1200)	281.90±17.45	0.13%	(0.3s)	396.80±17.10	0.29%	(1.9s)	546.28±18.99	0.32%	(5.4s)
HM (2400)	281.75±17.47	0.08%	(0.7s)	396.16±17.08	0.13%	(3.0s)	545.32±19.02	0.14%	(10.0s)
HM (4800)	**281.53**±17.35	**0.00%**	(1.1s)	**395.65**±17.00	**0.00%**	(5.0s)	**544.55**±19.01	**0.00%**	(18.2s)

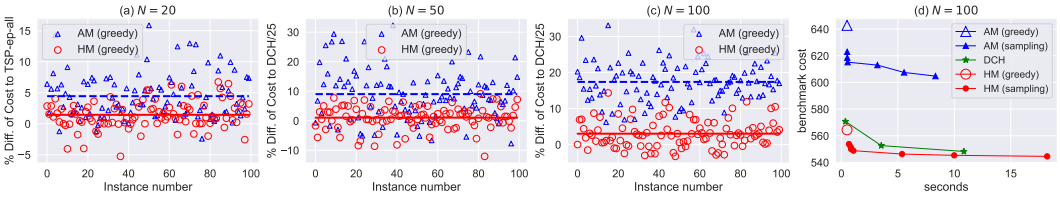


Figure 4: [(a), (b), (c)] AM (greedy) vs. HM (greedy). The percentage difference to TSP-ep-all ($N = 20$) or DCH/25 ($N = 50, 100$) is reported for each instance. The dashed and solid lines represent the mean values of AM and HM, respectively; [(d)] DCH/g vs. AM (s) vs. HM (s) for various g and s values. The average cost to the solution time is reported.

Table 2: TSP-D results on the Amsterdam dataset. Averages of 100 problem instances.

	$N = 10$			$N = 20$			$N = 50$		
Method	Cost	Gap	Time	Cost	Gap	Time	Cost	Gap	Time
TSP-ep-all	**2.02**±0.23	**0.00 %**	(0.0s)	**2.35**±0.21	**0.00 %**	(0.7s)	**3.26**±0.22	**0.00 %**	(4m)
DCH/10	-	-		2.49±0.24	5.70 %	(0.1s)	3.55±0.25	8.88 %	(0.2s)
DCH/25	-	-		-	-		3.37±0.22	3.24 %	(4.9s)
AM (greedy)	2.16±0.29	6.69 %	(0.1s)	2.61±0.32	10.94 %	(0.1s)	4.06±0.43	24.30 %	(0.2s)
HM (greedy)	2.08±0.26	2.84 %	(0.1s)	2.50±0.26	5.96 %	(0.1s)	3.50±0.28	7.31 %	(0.2s)
AM (4800)	2.08±0.25	2.92 %	(0.4s)	2.44±0.23	3.59 %	(0.8s)	3.72±0.34	13.95 %	(3.0s)
HM (100)	2.05±0.25	1.47 %	(0.1s)	2.41±0.24	2.31 %	(0.1s)	3.36±0.25	3.01 %	(0.3s)
HM (1200)	2.05±0.25	1.09 %	(0.2s)	2.39±0.23	1.57 %	(0.2s)	3.33±0.24	1.99 %	(1.4s)
HM (2400)	2.04±0.24	0.89 %	(0.2s)	2.39±0.23	1.39 %	(0.6s)	3.32±0.24	1.73 %	(2.2s)
HM (4800)	2.04±0.24	0.92 %	(0.4s)	2.38±0.23	1.27 %	(0.8s)	3.31±0.24	1.58 %	(3.8s)

Table 3: mmCVRP results with three vehicles. Averages of 128 problem instances.

	$N = 20$			$N = 30$		
Method	Cost	Gap	Time	Cost	Gap	Time
OR-Tools	**4,086.99**±471.78	**0.00 %**	(1.0s)	4,871.06±453.74	7.26 %	(1.0s)
AM (greedy)	4,396.11±535.81	7.56 %	(0.1s)	4,939.37±473.15	8.76 %	(0.2s)
HM (greedy)	4,298.83±523.19	5.18 %	(0.1s)	4,779.27±488.44	5.24 %	(0.2s)
AM (1200)	4,142.74±478.88	1.36 %	(14.3s)	4,574.05±420.01	0.01 %	(27.6s)
HM (1200)	4,129.41±496.54	1.04 %	(11.4s)	**4,541.45**±429.89	**0.00 %**	(25.2s)

DCH/25, the batch sampling approach using HM produces the best performance, as shown in Table 1, without increasing the solution time as much as TSP-ep-all. HM's scalability is comparable to DCH/25, but HM outperformed DCH/25 in terms of the average cost. Figure 4(d) also illustrates HM (sampling) is more effective than DCH and AM (sampling). No direct comparison of solution time between DCH and HM (sampling) is possible, although HM (sampling) shows high efficiency in our experiments. As DCH can be also run in parallel on multiple CPU threads, the speed of DCH can be further boosted. However, such parallelization for DCH is not as straightforward as in batch sampling on GPU.

The above observations are consistent in the real Amsterdam dataset in Table 2: HM provides better solutions than AM. For $N = 50$, AM (greedy) and HM (greedy) had gap of 24.30% and 7.31 %, respectively, compared to the TSP-ep-all solutions. AM (sampling) and HM (sampling) show the same pattern. However, the best solutions are found by TSP-ep-all in the Amsterdam dataset, while HM (sampling) still outperforms DCH. When customer locations follow non-uniform distributions, the performances of AM and HM seem to deteriorate, which is consistent with the finding of Bogyrbayeva et al. (2021).

4.3 Additional Results on mmCVRP

While our main focus is TSP-D, we also test performances of HM and AM in mmCVRP to demonstrate that HM can generalize to other multi-vehicle routing problems. TSP-D concerns the routing of heterogeneous vehicles (the drone is faster than the truck), whereas mmCVRP is about the routing of homogeneous vehicles (all vehicles are at the same speed and of the same capacity). In both cases, we aim to minimize the makespan to finish all jobs. The results in Table 3 demonstrate that HM provides competitive results with OR-Tools to solve the mmCVRP, while AM was not as competitive. The details of the experiments are provided in Appendix D.

4.4 Performances on TSP

While HM outperforms AM for TSP-D, the opposite is true for TSP. Our experiments on TSP in Appendix E.4 compare AM, HM, and NM. Note that NM also uses an LSTM decoder for routing problems. For TSP, the performances of the two LSTM-decoder models tend to behave similarly. The LSTM decoder seems not as effective as the attention-based decoder of AM for routing a single vehicle in TSP. This may be because the LSTM decoder puts unnecessary focus on the past decisions, while the current vehicle's location and the depot location are the only relevant information in TSP. AM uses only that information as an input to the decoder. However, the LSTM decoder is more effective in the coordinated routing of multiple vehicles as in TSP-D and mmCVRP since it can "memorize" the past decisions made for other vehicles.

5 Discussion

This study proposed a new end-to-end learning model that can learn coordinated routing of multiple vehicles. Our method applies to both heterogeneous and homogeneous fleet cases, as shown in TSP-D and mmCVRP, respectively. Compared to other learning methods to solve the routing problems, the proposed model efficiently deals with the coordination of multiple vehicles and achieves comparable or better performances as strong optimization heuristics methods. This is because the LSTM-based decoder of the proposed model stores the decisions of other vehicles to make better-informed decisions compared to stateless attention-based models. Moreover, in contrast to optimization heuristics that solve TSP-D and mmCVRP tailored to the specifics of each problem, the proposed model solves both problems with the same hyperparameters and input structures. The proposed model, however, does not perform as strongly in TSP.

Searching for a universal architecture that efficiently solves both TSP and TSP-D will be an important direction for future research. Such a universal architecture will solve a broad class of routing problems arising in various logistics services including heterogeneous fleets and multiple echelons of service vehicles. Identifying architectures and training algorithms robust to variations in distributions of customer locations will be another important direction. It is unclear how such variations would impact the performances of end-to-end learning methods. Understanding its impact will be critical for real-world applications.

REPRODUCIBILITY STATEMENT

We provided the details of our implementations in the appendix. After the review process, we will also archive our code for learning and heuristic algorithms, benchmark instance datasets, and the results we obtained and share them openly with the public.

REFERENCES

Niels Agatz, Paul Bouman, and Marie Schmidt. Optimization approaches for the traveling salesman problem with drone. *Transportation Science*, 52(4):965–981, 2018.

David Applegate, Robert Bixby, Vašek Chvátal, and William Cook. TSP cuts which do not conform to the template paradigm. In *Computational Combinatorial Optimization*, pp. 261–303. Springer, 2001.

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

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

Aigerim Bogyrbayeva, Sungwook Jang, Ankit Shah, Young Jae Jang, and Changhyun Kwon. A reinforcement learning approach for rebalancing electric vehicle sharing systems. *IEEE Transactions on Intelligent Transportation Systems*, Accepted:1–11, 2021.

Paul Bouman, Niels Agatz, and Marie Schmidt. Dynamic programming approaches for the traveling salesman problem with drone. *Networks*, 72(4):528–542, 2018.

Sung Hoon Chung, Bhawesh Sah, and Jinkun Lee. Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions. *Computers & Operations Research*, pp. 105004, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Zulqarnain Haider, Hadi Charkhgard, Sang Won Kim, and Changhyun Kwon. Optimizing the relocation operations of free-floating electric vehicle sharing systems. *Available at SSRN*, 2019.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 30: 6348–6358, 2017.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Giusy Macrina, Luigi Di Puglia Pugliese, Francesca Guerriero, and Gilbert Laporte. Drone-aided routing: A literature review. *Transportation Research Part C: Emerging Technologies*, 120:102762, 2020.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, pp. 105400, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Chase C Murray and Amanda G Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54: 86–109, 2015.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pp. 9839–9849, 2018.

Alena Otto, Niels Agatz, James Campbell, Bruce Golden, and Erwin Pesch. Optimization approaches for civil applications of unmanned aerial vehicles (UAVs) or aerial drones: A survey. *Networks*, 72 (4):411–458, 2018.

Laurent Perron and Vincent Furnon. OR-Tools, 2019. URL https://developers.google.com/optimization/.

Stefan Poikonen, Bruce Golden, and Edward A Wasil. A branch-and-bound approach to the traveling salesman problem with a drone. *INFORMS Journal on Computing*, 31(2):335–346, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Roberto Roberti and Mario Ruthmair. Exact methods for the traveling salesman problem with drone. *Transportation Science*, 55(2):315–335, 2021.

David W Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons, 2015.

Sebastián A Vásquez, Gustavo Angulo, and Mathias A Klapp. An exact solution method for the tsp with drone based on decomposition. *Computers & Operations Research*, 127:105127, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *arXiv preprint arXiv:1506.03134*, 2015.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

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

A THE HYBRID MODEL

This section describes the encoder presented in Figure 2 in detail. *A multi-head attention (MHA) layer* takes as an input the output of the previous layer (either output from the initial embedding or the output of the previous attention layer) and passes messages between nodes. In particular, for each input to MHA we compute the values of $\mathbf{q} \in \mathbb{R}^{d_q}, \mathbf{k} \in \mathbb{R}^{d_q}, \mathbf{v} \in \mathbb{R}^{d_v}$ or queries, keys and values respectively by projecting the input $\mathbf{h_n}$:

$$\mathbf{q}_n = \mathbf{W}^Q \mathbf{h}_n, \quad \mathbf{k}_n = \mathbf{W}^K \mathbf{k}_n, \quad \mathbf{v}_n = \mathbf{W}^V \mathbf{h}_n \quad \forall n \in \mathcal{N}, \tag{9}$$

where $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$ are trainable parameters with sizes $(d_k \times d_h)$, $(d_k \times d_h)$, and $(d_v \times d_h)$, respectively. From keys and queries we compute compatibility between nodes i and j as follows since we consider the fully connected graph:

$$u_{i,j} = \frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d_k}} \quad \forall i, j \in \mathcal{N}. \tag{10}$$

We use compatibility $u_{i,j}$ to compute the attention weights, $a_{i,j} \in [0, 1]$ using softmax:

$$a_{i,j} = \frac{e^{u_{ij}}}{\sum_{j'} e^{u_{i,j'}}}. \tag{11}$$

Then a message received by node n is a convex combination of messages received from all nodes:

$$\mathbf{h}_n^{'} = \sum_j a_{n,j} \mathbf{v}_j. \tag{12}$$

Instead of using a single head, we use a multi-head attention with head size M, which allows passing different messages from other nodes. Then after computing messages from each head using (11), we can combine all massages coming to node n as follows:

$$\mathbf{MHA}_n(\mathbf{h}_1, \ldots, \mathbf{h}_N) = \sum_{m=1}^{M} \mathbf{W}_m^O \mathbf{h}_{nm}^{'}. \tag{13}$$

The output of the MHA sublayer along with skip connection is passed through Batch Normalization:

$$\widehat{\mathbf{h}}_n^l = \mathbf{BN}^l(\mathbf{h}_n^{l-1} + \mathbf{MHA}_n(\mathbf{h}_1^{l-1}, \ldots, \mathbf{h}_N^{l-1})). \tag{14}$$

The output of Batch Normalization then passed through a fully connected feed-forward (FF) network with ReLU activation function. We again apply skip connection and batch normalization to the output of FF.

$$\mathbf{h}_n^l = \mathbf{BN}^l(\widehat{\mathbf{h}}_n^l + \mathbf{FF}^l(\widehat{\mathbf{h}}_n^l)), \tag{15}$$

where

$$\mathbf{FF}^l(\widehat{\mathbf{h}}_n^l)) = \mathbf{W}^{\text{ff},1} \cdot \text{ReLU}(W^{\text{ff},0} \widehat{\mathbf{h}}_n^l + \mathbf{b}^{\text{ff},0}) + \mathbf{b}^{\text{ff},1}. \tag{16}$$

B THE ATTENTION MODEL

The attention model presented in Kool et al. (2018) consists of the multi-head attention encoder and decoder. While both HM and AM use multi-head attention to encode a graph structure, they differ in the decoder. AM's decoder takes as an input the embedding of the graph and the embeddings of the first (depot) and the last node of a decision taker at time t.

$$\mathbf{h}_c = [\bar{\mathbf{h}}, \mathbf{h}_f^L, \mathbf{h}_i^L], \tag{17}$$

where $\bar{\mathbf{h}}$ is a graph embedding defined as the mean of the embeddings of all the nodes, $\mathbf{h}_f^L$ and $\mathbf{h}_i^L$ are the embedding of the first and last nodes at time t. Initially, when $t = 0$, the last and first nodes refer to the depot.

Later the context node $\mathbf{h}_c$ is used to compute a single query while embeddings of all nodes are used to compute keys and values:

$$\mathbf{q_c} = W^Q \mathbf{h}_c, \quad \mathbf{k_n} = W^K \mathbf{h}_n^L, \quad \mathbf{v_n} = W^V \mathbf{h}_n^L \quad \forall n \in N, \tag{18}$$

where W^Q, W^K, W^V are trainable parameters and from keys and the query we compute compatibility between nodes i and c using the following formula and masking nodes that cannot be visited at the current time step:

$$u_{c,n} = \frac{\mathbf{q}_c^T \mathbf{k}_n}{\sqrt{d_k}} \quad \forall n \quad \text{if } n \text{ is not masked.} \tag{19}$$

We again use a multi-head attention mechanism as in Encoder and follow steps shown in equations 10-12. We do not use skip connection and batch normalization for the output of MHA.

To compute logits, the final output of MHA, $\mathbf{q}_c$ is passed through a single-head attention, where the compatibility with only a context node is calculated:

$$u_{c,n} = C \tanh \left(\frac{\mathbf{q}_c^T \mathbf{k}_j}{\sqrt{d_k}} \right) \quad \forall n \quad \text{if } n \text{ is not masked.} \tag{20}$$

The final probabilities of visiting the next nodes are computed through softmax:

$$p_n = \frac{e^{u_{c,n}}}{\sum_{n'} e^{u_{c,n'}}} \quad \forall n \in \mathcal{N} \text{if } n \text{ is not masked.} \tag{21}$$

For masked nodes, the probabilities are set to zero.

C DISTRIBUTED TRAINING ALGORITHMS

Algorithm C.1: Distribute-and-Follow Policy Gradient

Generate a set of K multiple parallel workers, $\Omega = \{\omega_1, \ldots, \omega_K\}$.
Initialize actor and critic networks parameters $\{\theta_1^a, \ldots, \theta_K^a\}$, $\{\theta_1^c, \ldots, \theta_K^c\}$ of A.
Set the maximum number of epochs, M_{epochs}
for epochs $= 1$ **to** M_{epochs} **do**
  Initialize an empty list of validation rewards **R**
  for k = *1* **to** K **do** ▷ in parallel
    Initialize network gradients $d\theta_k^a \leftarrow 0, d\theta_k^c \leftarrow 0$
    $B_k \sim$ DataGenerator(ρ)
    Call Rollout(ω_k, B_k)
    $R_k \leftarrow$ Evaluate(ω_k)
    Append R_k to **R**
  $j \leftarrow \arg\min_{1 \le j \le K}(\mathbf{R})$
  for k = *1* **to** K **do** ▷ in parallel
    $\theta_k^a \leftarrow \theta_j^a$
    $\theta_k^c \leftarrow \theta_j^c$

Algorithm C.2: Rollout

Input: Batch of data B with a number of episodes denoted M_{epis}, a worker id k. Set the maximum number of steps denoted, T

Initialize reward R
Initialize LSTM initial state $(\widetilde{h}_0, \widetilde{c}_0)$
$x_0, \text{mask}_0 \leftarrow$ ENV.RESET(B)
for *t=0* **to** T **do**
  $a_t^{\text{tr}}, (\widetilde{h}_{t'}, \widetilde{c}_{t'}) \leftarrow \pi_{\theta_k^a}(x_t^{\text{tr}}, \text{mask}_t^{\text{tr}}, (\widetilde{h}_t, \widetilde{c}_t))$
  $a_t^{\text{dr}}, (\widetilde{h}_t, \widetilde{c}_t) \leftarrow \pi_{\theta_k^a}(x_t^{\text{dr}}, \text{mask}_t^{\text{dr}}, (\widetilde{h}_{t'}, \widetilde{c}_{t'}))$
  $x_{t+1}, \text{mask}_{t+1}, C_t \leftarrow$ ENV.STEP$(a_t^{\text{tr}}, a_t^{\text{dr}})$
  $R \leftarrow R - C_t$
Calculate $b(x_0; \theta_k^c)$ using critic
$d\theta_k^a \leftarrow \frac{1}{M_{\text{epis}}} \sum_{m=1}^{M_{\text{epis}}} (R^m - b^m(x_0^m; \theta_k^c)) \nabla_{\theta_k^a} \log \pi_{\theta_k^a}$
$d\theta_k^c \leftarrow \frac{1}{M_{\text{epis}}} \sum_{m=1}^{M_{\text{epis}}} \nabla_{\theta_k^c} (R^m - b^m(x_0^m; \theta_k^c))^2$

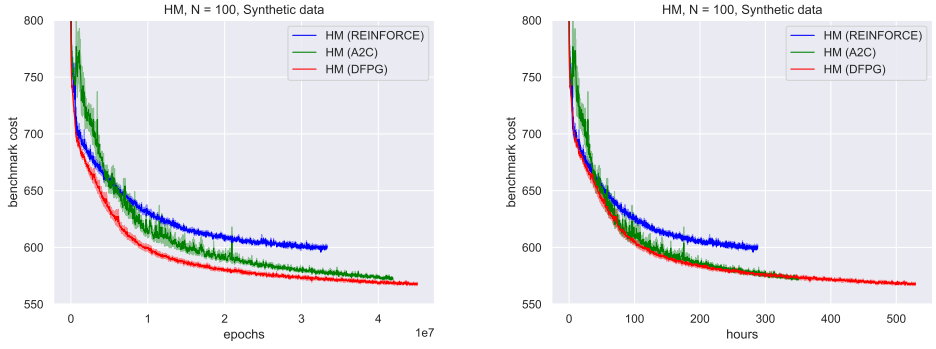


Figure C.5: Benchmark cost curve comparison between A2C and DFPG on 100-node graphs from the environment without revisiting. Synthetic data refers to 100 benchmark instances used in Table 1.

D IMPLEMENTATION DETAILS FOR TSP-D

A critic network for HM. The architecture of the critic network has similarities to the actor network, except we do not use recurrent neural networks. The goal of the critic network is to estimate the total time needed to serve all the customers and return back to the depot, which is achieved through embedding the initial state of the graph. In particular, we embed x and y coordinates of the nodes through element-wise projections with 1D convolution networks whose outputs are passed through attention followed by feed-forward networks. Python 3.8 and PyTorch 1.5 are used.

Data generation for the Amsterdam dataset. To identify potential demand locations for urban mobility and logistics services, we used an electric-vehicle (EV) parking location dataset in Amsterdam, used in Haider et al. (2019). As EVs are usually parked on the urban streets, next to the curbside chargers, these locations reflect where potential customers are located. For a detailed description of the dataset, refer to Haider et al. (2019). We randomly pick depot and customer nodes from the entire Amsterdam dataset to create various problem instances. We use the Amsterdam dataset for evaluation only and generate the training data on the fly using kernel density estimation. In particular, we estimate probability density functions of x and y coordinates of the customer nodes using Gaussian kernels. We deploy the Gaussian-KDE library of SciPy (Virtanen et al., 2020) that selects bandwidth using the Scott's Rule (Scott, 2015).

AM Implementation for TSP-D. To have a fair comparison between AM of Kool et al. (2018) and our HM, we tried several state representations of TSP-D for AM. The state representation of TSP used in AM is inappropriate to solve TSP-D because the routing decisions of the drone and the truck are strongly interconnected in TSP-D. There are mainly two challenges in the state representation of TSP-D: (i) we need to pass information about previously selected nodes either by drone or truck; and (ii) we also need to promote coordination between the drone and truck routing through passing information about their current transitions in a graph.

The first challenge also exists with TSP, which AM has resolved through the masking scheme. Instead of passing information about already selected nodes as a state, AM prohibits selecting such nodes and only uses the current location as part of the state representation. To address the second challenge, we tried to pass the current transitions of the drone and truck in a context node of a decoder (discussed in Appendix B). In particular, we passed the encoded current location information of a decision taker along with the encoded version of the selected node of the counterpart to compute the context node as follows:

$$\mathbf{h}_c = [\bar{\mathbf{h}}, \mathbf{h}_i^L, \mathbf{h}_j^L] \tag{22}$$

where $\bar{\mathbf{h}}$ is a graph embedding defined as the mean of the embeddings of all the nodes, $\mathbf{h}_i^L$ and $\mathbf{h}_j^L$ are the embedding of the last action of a decision taker and the last action of the counterpart at time t. Initially, when $t = 0$, the last actions of both vehicles refer to the depot.

We also tried passing the convex combinations of h_j^L and h_i^L. Also, we tried to inject the distances between nodes in the encoder through additive and linear attention. However, from the experimental studies, we observed that passing the encoded nodes information of the current location of the decision taker and the last selected node of a counterpart in a context node produced the best training rewards. Therefore, we selected such state representation and used the masking scheme similar to Kool et al. (2018), to prevent visiting already selected nodes.

Hyperparameters. We use 128 instances of data generated on the fly per epoch to train HM. We use 3 layers in the encoder and 8 attention heads. We initialize all the encoder parameters using Uniform($1/\sqrt{d}$, $1/\sqrt{d}$), with d the input dimension. In the decoder, we initialize the LSTM hidden and cell states with zeros and apply dropout with $p = 0.1$. Embedding dimensions are set to 256 for TSP-D with uniformly distributed graphs and 128 for other experiments. We use $d_h = 128$ to compute attention in the decoder. We use the constant learning rate set to 10^{-4}. For AM we used the same hyperparameters as in Kool et al. (2018).

Gap. In all the experiments, we use the following formula to report the relative gap between two solution methods:

$$\text{Gap} = \frac{Z - Z^*}{Z^*} \times 100\%, \tag{23}$$

where Z is the average cost of a solution method and Z^* is the average cost of the best-performing solution method. The best-performing solution method is selected based on the average cost.

E ADDITIONAL EXPERIMENTS

E.1 PERFORMANCES ON THE TSP-D EXACT SOLUTION BENCHMARK INSTANCES

Agatz et al. (2018) provide 10 instances of size $N = 11$ with exact optimal solutions, available at https://github.com/pcbouman-eur/TSP-D-Instances. We provide the performance of HM greedy and HM sampling on these benchmark instances in Table E.4. On average, the optimality gap of HM greedy and HM sampling is 0.93% and 0.69%, respectively.

The detailed routes are provided in Figures E.6 and E.7. Note that in Instances 6 and 10, the two routes are not identical, but both are optimal. This is because the objective in TSP-D is to minimize the makespan, not the total distance traveled.

Table E.4: The objective values of the optimal solutions and Hybrid Model solutions on TSP-D with 11 nodes.

Instance	Optimal	HM (greedy) (Gap)	HM (1200) (Gap)
1	221.19	223.41 (1.00%)	223.41 (1.00%)
2	205.76	205.76 (0.00%)	205.76 (0.00%)
3	192.96	193.99 (0.53%)	193.99 (0.53%)
4	241.26	241.26 (0.00%)	241.26 (0.00%)
5	248.14	249.85 (0.69%)	248.82 (0.67%)
6	217.69	217.69 (0.00%)	217.69 (0.00%)
7	237.34	240.47 (1.32%)	237.34 (0.00%)
8	214.77	226.64 (5.53%)	225.36 (4.93%)
9	256.34	256.83 (0.19%)	256.83 (0.19%)
10	227.90	227.90 (0.00%)	227.84 (0.00%)
mean	**226.33**	**228.38 (0.93%)**	**227.84 (0.69%)**

E.2 DISTRIBUTED TRAINING ALGORITHMS FOR TSP-D WHEN REVISITING IS ALLOWED

In Instance 9 of Figure E.7, the exact optimal solution revisits node 9. This is an important observation made by Agatz et al. (2018). To learn the true optimal solutions, we need to allow revisiting nodes in the simulator.

Allowing revisiting nodes makes training of both AM and HM significantly slower, as it induces much larger action spaces. AM even did not train smoothly in our experiments. When revisiting nodes is allowed, as in Instance 9 in Figure E.7, we compared the performances of training algorithms in Figure E.8. We observe that there are no significant gains from using A2C in the environment with revisiting allowed. The proposed DFPG method shows more stable learning curves with the best performance. This result demonstrates that DFPG has the potential to be useful in various routing problems with very large action spaces.

However, in this paper, we did not allow revisiting nodes, because our experiments showed not much gain in the performance on average when compared to the results with no revisiting allowed. For example, the HM solution for Instance 9 without revisiting has a gap of 0.19% only.

E.3 DETAILS OF THE MMCVRP EXPERIMENTS

We have a set of nodes representing customer locations and a depot. Each customer has known demand, which a fleet of vehicles must satisfy and return back to a depot in a minimal time. Each customer can be served by exactly one vehicle. The routing decisions of vehicles are restricted by their capacity to carry the load. We compare HM against OR Tools (Perron & Furnon, 2019) to solve mmCVRP. We implemented the mmCVRP simulator using Julia 1.5 and interfaced it with Python using pyjulia available at https://github.com/JuliaPy/pyjulia. For both $N = 20$ and $N = 30$, we used three vehicles. We use the same set of hyperparameters for both TSP-D and mmCVRP.

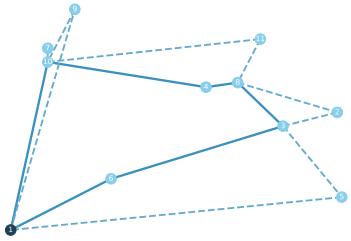
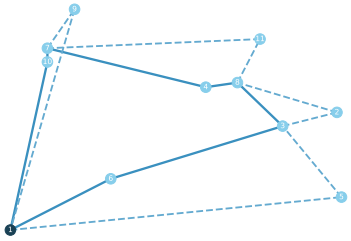
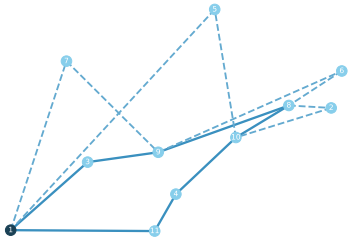
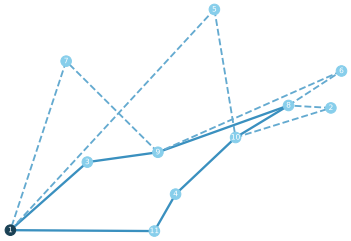
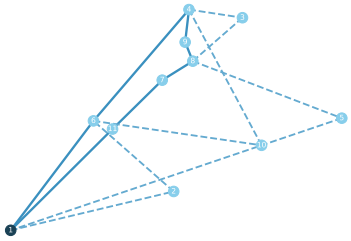
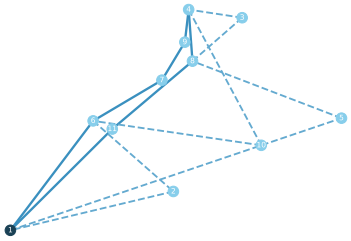
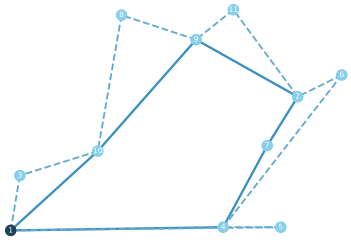
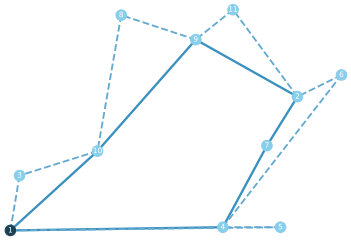
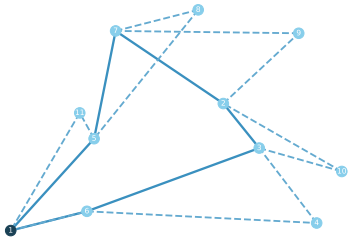
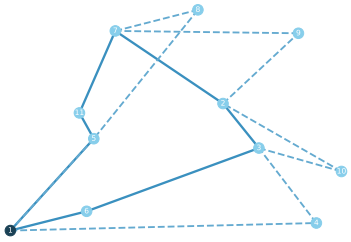
Instance	Optimal Solution	HM Greedy
1	 Cost = 221.19	 Cost = 223.41 (Gap = 1.00%)
2	 Cost = 205.76	 Cost = 205.76 (Gap = 0.00%)
3	 Cost = 192.96	 Cost = 193.99 (Gap = 0.53%)
4	 Cost = 241.26	 Cost = 241.26 (Gap = 0.00%)
5	 Cost = 248.14	 Cost = 249.85 (Gap = 0.69%)

Figure E.6: Routing examples: optimal vs. HM greedy (Instances 1 to 5). Node 1 is the depot, the solid line is the truck route, and the dashed line is the drone route.

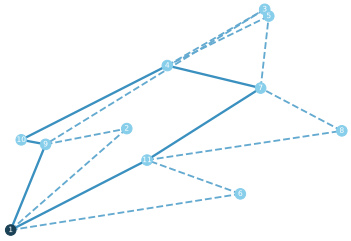
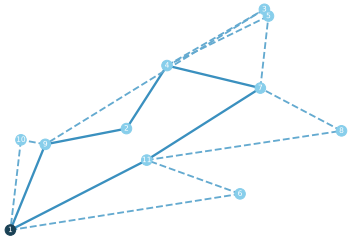
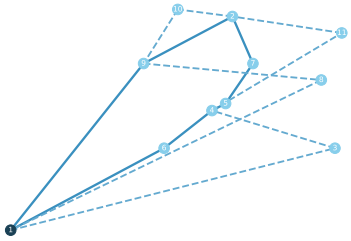
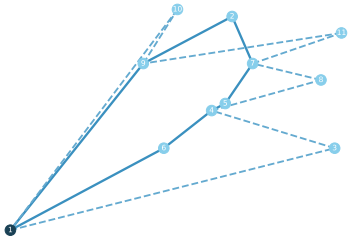
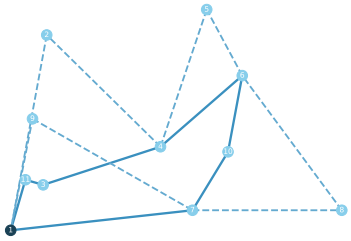
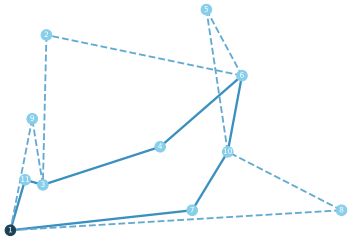
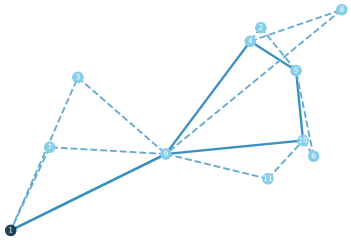
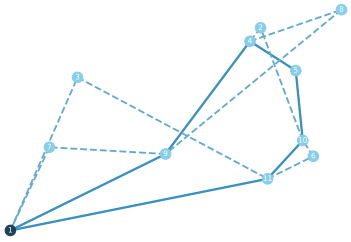
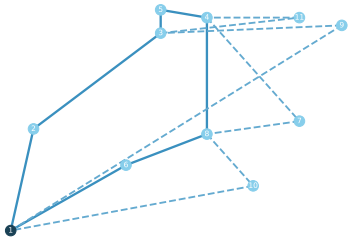
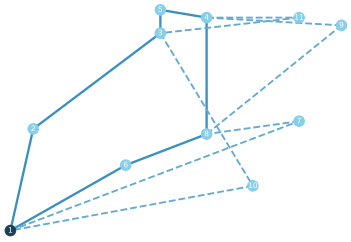
Instance	Optimal Solution	HM Greedy
6	 Cost = 217.69	 Cost = 217.69 (Gap = 0.00%)
7	 Cost = 237.34	 Cost = 240.47 (Gap = 1.32%)
8	 Cost = 214.77	 Cost = 226.64 (Gap = 5.53%)
9	 Cost = 256.34	 Cost = 256.83 (Gap = 0.19%)
10	 Cost = 227.90	 Cost = 227.90 (Gap = 0.00%)

Figure E.7: Routing examples: optimal vs. HM greedy (Instances 6 to 10). Node 1 is the depot, the solid line is the truck route, and the dashed line is the drone route.

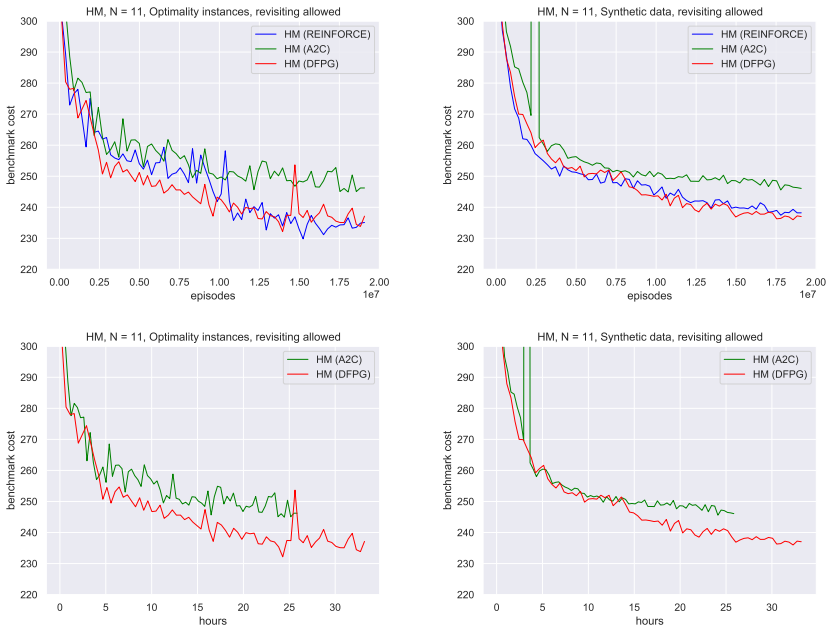


Figure E.8: Benchmark cost curves of HM with REINFORCE, A2C, and DFPG on 11-node graphs from the environment that *allows revisiting*. Optimality instances refer to 11-node instances from Agatz et al. (2018). Synthetic data refers to 100 benchmark instances used in Table 1. Top figures represent benchmark costs over the number of episodes. Bottom figures compare benchmark costs over wall-clock time.

Data generation. We generate the random data for mmCVRP by sampling x, y coordinates of the customer locations using Uniform(0, 1). We randomly assign customer demand at each node with integer values ranging between 1 and 9. We set the capacity of each vehicle as follows:

$$Q = \left\lceil 1.2/m \sum_{n \in \mathcal{N}} D_n \right\rceil, \tag{24}$$

where D_i is demand at node i, and m is the number of vehicles. We computed the traveling time between nodes as follows:

$$\tau_{i,j} = \left\lfloor 2000 d_{ij} \right\rfloor \quad \forall i \in \mathcal{N}, j \in \mathcal{N}, \tag{25}$$

where $d_{i,j}$ is the Euclidean distance between nodes i and j.

HM implementation for mmCVRP. We embed a graph exactly as in TSP-D. However, in the decoder, we also pass information about the remaining capacity of a vehicle. In particular, for each node i, we compute expected capacity if it is visited by a decision taker at time t: $Q'_t = Q_t - D_i^t$ for all $i \in N$, where Q_t is a capacity of a decision taker at time t and D_i^t is demand at node i at time t. Then we use a vector of expected remaining capacities $\mathbf{Q}'_t$ to compute attention in decoder:

$$\mathbf{a}_{i,\cdot} = \mathbf{v}_a^\top \tanh\left(\mathbf{W}^a[\bar{\mathbf{h}}; \widetilde{\mathbf{h}}_{t+1}; \mathbf{W}^d \mathbf{Q'}_t]\right), \tag{26}$$

AM implementation for mmCVRP. To compute the context node for mmCVRP, we use the embeddings of the current node of a decision taker and embeddings of the nodes selected by other nodes. In particular, in the presence of m vehicles in mmCVRP, we will have:

$$\mathbf{h}_c = [\bar{\mathbf{h}}, \mathbf{h}_i^L, \mathbf{h}_{j_1}^L ... \mathbf{h}_{j_{m-1}}^L, \mathbf{W}^d \mathbf{Q}_t] \tag{27}$$

where $\mathbf{h}_i^L$ is embedding of a node i selected by a decision taker and $\mathbf{h}_{j_{m-1}}^L$ is a embedding of a node j selected by a vehicle $m-1$. In computing the context node, we also use the remaining capacity of a decision taker $\mathbf{Q}_t$ at time t.

E.4 Performances on TSP Instances

We compare the performance of AM, HM, and the RL model by Nazari et al. (2018) for TSP. We call the Nazari et al. model NM. We generate random location instances with the same method as Kool et al. (2018) and Nazari et al. (2018). While we implemented AM and HM for TSP by ourselves, we use the result reported by Nazari et al. (2018) for NM. Random location instances are generated using the same rule. The result is presented in Table E.5.

HM for TSP is similar to TSP-D, except we pass a distance vector, $\mathbf{d}_{i,.}$, from the current location of an agent i to other nodes as follows in computing attention in decoder:

$$\mathbf{a}_{i,.} = \mathbf{v}_a^\top \tanh\left(\mathbf{W}^a[\bar{\mathbf{h}}; \widetilde{\mathbf{h}}_{t+1}; \mathbf{W}^d\mathbf{d}_{i,.}]\right). \tag{28}$$

Table E.5: TSP results on random locations. Average over 10,000 problem instances are reported. Cost refers to the cost function (1). Gap is the relative difference to the cost of the best algorithm for the setting.

	$N = 20$		$N = 50$		$N = 100$	
Method	Cost	Gap	Cost	Gap	Cost	Gap
Concorde	**3.83**	**0.00 %**	**5.69**	**0.00 %**	**7.76**	**0.00 %**
AM (greedy)	3.84	0.29 %	5.79	1.67 %	8.10	4.38 %
AM (sampling)	3.83	0.07 %	5.72	0.48 %	7.95	2.34 %
HM (greedy)	3.88	1.38 %	6.00	5.39 %	8.54	9.93 %
HM (sampling)	3.84	0.39 %	5.86	2.92 %	8.11	4.45 %
NM (greedy)	3.97		6.08		8.44	