Transportation Research Part C 148 (2023) 103981

Contents lists available at ScienceDirect

Transportation Research Part C

journal homepage: www.elsevier.com/locate/trc

Overview Paper

# A deep reinforcement learning approach for solving the Traveling Salesman Problem with Drone

Aigerim Bogyrbayeva [a], Taehyun Yoon [b], Hanbum Ko [b], Sungbin Lim [c], Hyokun Yun [d,1], Changhyun Kwon [e,*,2]

[a] *Suleyman Demirel University, Kazakhstan*
[b] *UNIST, South Korea*
[c] *Korea University, South Korea*
[d] *Amazon, USA*
[e] *University of South Florida, USA*

ARTICLE INFO

*Keywords:*
Vehicle routing
Traveling salesman problem
Drones
Reinforcement learning
Neural networks

ABSTRACT

Reinforcement learning has recently shown promise in learning quality solutions in many 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 Drone (TSP-D), requiring routing a heterogeneous fleet of vehicles in coordination—a truck and a drone. In TSP-D, the two vehicles are moving in tandem and may need to wait at a node for the other vehicle to join. State-less attention-based decoder fails to make such coordination between vehicles. We propose a hybrid model that uses an attention encoder and a Long Short-Term Memory (LSTM) network decoder, 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 the coordinated routing of multiple vehicles than the attention-based model. The proposed model demonstrates comparable results as the operations research baseline methods.

## 1. Introduction

Last-mile delivery, which refers to the transportation of products from a distribution center to the doorstep of a customer, is an integral part of the supply chain. However, last-mile delivery is often not cost-effective due to transportation costs associated with individualized shipments, complex routes, and various destinations. For instance, last-mile delivery costs comprise 50% of the total delivery costs (Joerss et al., 2016). Therefore, emerging technologies such as unarmed aerial vehicles (drones) are viewed as a potential solution in reducing inefficiencies in last-mile delivery. For instance, Amazon launched the Prime Air program that aims to deliver parcels using drones (Amazon Prime Air, 2022). Similarly, Wing, a subsidiary of Alphabet, delivers books, meals, and medicine using drones (Wing Aviation, 2021). The steady development of drone technology, including its capacity and flying range,

[*] Corresponding author.
*E-mail addresses:* aigerim.bogyrbayeva@sdu.edu.kz (A. Bogyrbayeva), thyoon@unist.ac.kr (T. Yoon), hanbum.ko95@unist.ac.kr (H. Ko), sungbin@korea.ac.kr (S. Lim), yunhyoku@amazon.com (H. Yun), chkwon@usf.edu (C. Kwon).
[1] This work does not relate to his position at Amazon.
[2] Part of this research was done while visiting KAIST, Daejeon, South Korea.

https://doi.org/10.1016/j.trc.2022.103981
Received 21 December 2021; Received in revised form 3 December 2022; Accepted 4 December 2022
Available online 7 January 2023

enables the transport of various products. For instance, the HorseFly drone, used by UPS, can carry packages up to 10 pounds and has a flight time of 30 min (Business Insider, 2017).

While drones certainly have high speed and do not require any infrastructure such as roads, bridges, etc., to fly, they also possess some limitations. Drones have a limited flying range, which depends on battery life. Also, drones cannot carry parcels of many customers; thus, they must return every time to a distribution center to pick up customer orders. On the other hand, trucks, which are traditionally used for last-mile delivery, have a large capacity to store all customer orders and can transport goods long distances. However, trucks usually have slow speeds due to congestion and require built infrastructure to reach their destinations. Therefore, combining the drone and truck is a promising tandem to improve the efficiency of last-mile delivery. Indeed, UPS has started testing combining the drone and the truck to deliver goods (Business Insider, 2017). In this setting, a truck driver loads a package into the drone and sends the drone to an autonomous route to an address. Meanwhile, the truck can serve other customers. When the drone returns to the truck, the driver swaps the battery of the drone and launches the next delivery.

To effectively deploy trucks and drones together for last-mile delivery, we must answer several challenging questions such as which customers should be served by drone, which customers should be served by truck, where to recharge the drone, how to route both drone and truck, etc. In the literature, the problem of routing the truck and drone is known as the Traveling Salesman Problem with Drone (TSP-D) (Agatz et al., 2018). The objective of TSP-D is to serve customers in a minimal time, while the truck and the drone are subjected to routing constraints. TSP-D is an NP-hard problem (Agatz et al., 2018), highlighting the need to develop heuristic methods that can scale to large urban networks. The development of efficient computational methods for TSP-D (Agatz et al., 2018; Poikonen et al., 2019; Roberti and Ruthmair, 2021; Vásquez et al., 2021) is still in its infancy. For instance, 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.

Our goal is to devise an end-to-end learning model that can address the above-outlined challenges. By end-to-end, we mean a learning-based model that takes problem instances as input and produces solutions as its direct output by itself without any additional algorithmic components.

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 a hybrid model (HM), based on an attention encoder and a Long Short-Term Memory (LSTM) decoder, for efficient routing of multiple vehicles taking into account required interactions between vehicles, along with a novel distributed training algorithm. 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. As our numerical results show, the proposed model performs comparably with operations research methods.

The remainder of the paper will proceed as follows: in Section 2 we present the literature review related to TSP-D and learning methods to solve routing problems; in Section 3 we formally define TSP-D and present its Markov Decision Process formulation; in Section 4 we present the Hybrid Model and a training algorithm to solve the problem; in Section 5 we present the main computational results for TSP-D; in Section 6 we provide additional computational experiments in other routing problems; and finally in Section 7 we conclude with final remarks and discussions.

## 2. Literature review

In this section, we present recent studies aimed at solving TSP-D. We limit our literature review to studies that focus on routing a single drone and a single truck where both vehicles actively participate in serving customers. The detailed reviews of combined drone-truck routing and comparison among problem variants can be found in Macrina et al. (2020) and Chung and Sah (2020). Also, we present the recent advancements in learning methods to solve routing problems in general.

### 2.1. Recent studies to solve TSP-D

Murray and Chu (2015) first introduced the notion of *sortie* for combined operations of the drone and truck, which they named the flying sidekick TSP (FSTSP). In a drone sortie, there are start and end nodes, which are defined as the nodes where the drone is launched and reunited with the truck. Between start and end nodes, the drone serves a single customer, while the truck may visit several customers. However, the start and end nodes of the drone cannot be the same, and they are limited to customer locations or a depot. The synchronization constraints are added to ensure that the drone and truck arrive at an end node at the same time. The drone routing is subjected to its battery life. At end nodes, the drone's battery is swapped with some service time for recharging. The objective of FSTSP is to minimize the total time spent in the system starting from when both the drone and truck leave a depot and return back after serving all customers. Murray and Chu (2015) presented the exact and heuristic methods to solve FSTSP. Later, Ha et al. (2018) extended the MIP of Murray and Chu (2015) to minimize the total transportation costs and Yurek and Ozmutlu (2018) introduced an iterative algorithm based on decomposition to solve FSTSP to minimize completion time. Dell'Amico et al. (2021c) and Dell'Amico et al. (2019) present enhanced set of formulations of FSTSP, for which Dell'Amico et al. (2021a) proposed a random restart local search heuristic to solve the large instances of the problem. Dell'Amico et al. (2021b) proposed a branch and bound algorithm to solve FSTPS up to 15 customers and devised an efficient heuristic to solve large instances. Liu et al. (2022) used a reinforcement learning approach to consider stochastic traveling time in FSTSP. Boccia et al. (2021) develops an exact method relying on a column-and-row generation approach to solve FSTPS.

In contrast to drone sortie assumptions, Agatz et al. (2018) presented the idea of *operations*, which is similar to drone sortie except it allows the drone to be launched and reunited with the truck at the same nodes. In fact, in their formulation, the truck and the drone are allowed to revisit nodes, which substantially increases the number of possible operations. Also, they considered different speeds for the drone and truck and allow the truck to wait for the drone at a rendezvous node. With these problem settings, Agatz et al. (2018) define TSP-D. Carlsson and Song (2018) investigated the benefits of combined drone and truck operations in general, where drone launch and pick-up nodes are not limited to customer locations. Gonzalez-R et al. (2020) extended TSP-D by allowing drones to visit multiple customers per launch and present an iterative greedy search heuristic.

For solving TSP-D, Agatz et al. (2018) devised a heuristic method that starts with an optimal TSP tour and partitions customer nodes into truck nodes and drone nodes using a dynamic programming approach. Later Poikonen et al. (2019) introduced a branch-and-bound method, where the drone and truck can wait for each other at rendezvous nodes, and presented four heuristics to solve large instances of the problem. de Freitas and Penna (2020) solved both TSP-D and FSTSP using a variable neighborhood search-based heuristic.

Exact optimization methods are also developed for TSP-D. Bouman et al. (2018) developed a dynamic programming method that can solve instances with up to 15 customers to optimality, and Poikonen et al. (2019) introduced a branch-and-bound method for up to 20 customer nodes. Schermer et al. (2020) developed a branch-and-cut approach to find optimal solutions for instances up to 20 customers. El-Adle et al. (2021) presented a novel Mixed Integer Programming formulation for TSP-D, which was later improved using pre-processing and valid inequalities. Vásquez et al. (2021) proposed a Benders-type decomposition method and Roberti and Ruthmair (2021) devised a branch-and-price algorithm to solve instances with up to 15 and 39 customer nodes, respectively.

### 2.2. Learning methods to solve routing problems

Recently reinforcement learning has received increased attention for solving routing problems. Starting from Bello et al. (2016), which used Pointer Networks (Vinyals et al., 2015) to solve TSP, learning methods have shown comparable results with optimization heuristics. For instance, Khalil et al. (2017) proposed a novel graph embedding structure to learn Q-function using DQN algorithm (Mnih et al., 2013) to solve a group of combinatorial optimization problems, including TSP.

To directly learn the routing policies, the encoder-decoder structure has been a common choice for many studies. The encoder is a neural network with the primary goal to learn the graph representation given some input. The output of the encoder is passed to the decoder, a neural network that learns routing policy given the current state of the graph. Different neural network architectures have been proposed both for encoder and decoder, including fully attention-based models and LSTM-based models (Bogyrbayeva et al., 2022). The attention mechanism is a technique that aims to find the most critical parts between inputs or between inputs and outputs. On the other hand, LSTM is a type of recurrent neural network that is designed to retain memories of both long and short-term dependencies in a sequence.

A series of research (Vinyals et al., 2015; Nazari et al., 2018; Kool et al., 2018) has shown that end-to-end learning, where only machine learning methods are applied, is a viable option for solving routing problems. In particular, Kool et al. (2018) demonstrated that the Attention Model (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 used 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. Such conditional independence is sensible for single-vehicle routing problems in which the ordering of past actions is irrelevant to future actions. However, it is not well-suited for problems like TSP-D requiring coordination among multiple vehicles.

Another set of studies focused on routing a fleet of vehicles consisting of homogeneous vehicles using a multi-agent model (Sykora et al., 2020; Zhang et al., 2020) or a single-agent model (Bogyrbayeva et al., 2021). However, TSP-D considers routing multiple heterogeneous vehicles, including the truck and drone, with different routing constraints and capabilities.

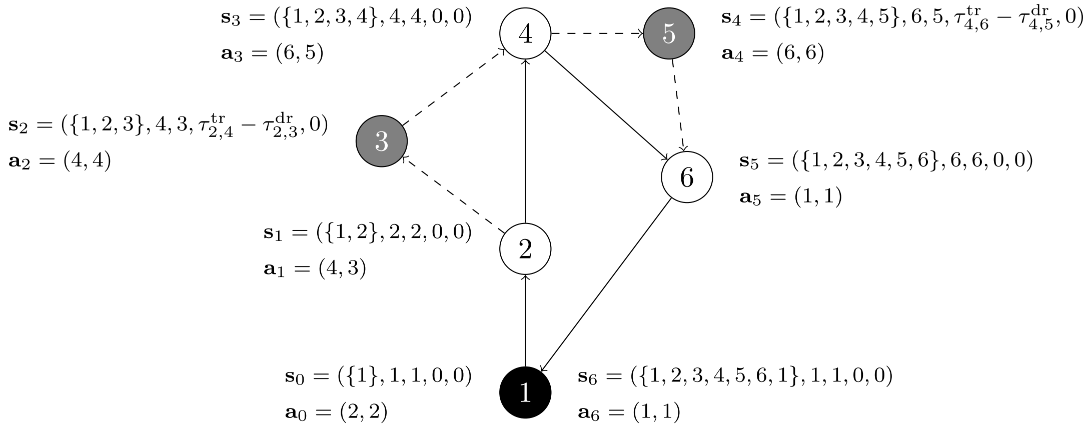

$\mathbf{s}_3 = (\{1,2,3,4\},4,4,0,0)$
$\mathbf{a}_3 = (6,5)$
$\mathbf{s}_4 = (\{1,2,3,4,5\},6,5,\tau^{\mathrm{tr}}_{4,6} - \tau^{\mathrm{dr}}_{4,5},0)$
$\mathbf{a}_4 = (6,6)$
$\mathbf{s}_2 = (\{1,2,3\},4,3,\tau^{\mathrm{tr}}_{2,4} - \tau^{\mathrm{dr}}_{2,3},0)$
$\mathbf{a}_2 = (4,4)$
$\mathbf{s}_5 = (\{1,2,3,4,5,6\},6,6,0,0)$
$\mathbf{a}_5 = (1,1)$
$\mathbf{s}_1 = (\{1,2\},2,2,0,0)$
$\mathbf{a}_1 = (4,3)$
$\mathbf{s}_0 = (\{1\},1,1,0,0)$
$\mathbf{a}_0 = (2,2)$
$\mathbf{s}_6 = (\{1,2,3,4,5,6,1\},1,1,0,0)$
$\mathbf{a}_6 = (1,1)$
1
2
3
4
5
6

**Fig. 1.** An example of TSP-D with state $\mathbf{s}_t := (\mathcal{V}_t, c^{\mathrm{tr}}_t, c^{\mathrm{dr}}_t, r^{\mathrm{tr}}_t, r^{\mathrm{dr}}_t)$ and action $\mathbf{a}_t = (a^{\mathrm{tr}}_t, a^{\mathrm{dr}}_t)$ on six nodes: black, gray 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 at the last customer location and traverse back together to the depot.

Beyond end-to-end learning models, there is another stream of research that combines optimization methods, multi-start approaches, or iterative processes with learning models to solve routing problems. Examples include Lu et al. (2020), Kwon et al. (2020), Hottung and Tierney (2020), Kim et al. (2021), and Kool et al. (2021). However, all the above-mentioned papers do not focus on routing together heterogeneous vehicles such as drones and trucks, the defining feature of TSP-D. Therefore, this paper focuses on developing an end-to-end model for TSP-D, which can provide a basis for integrated and iterative methods as well.

## 3. Problem statements and formulations

In this section, we first define TSP-D and then present the Markov Decision Process formulation of the problem.

### 3.1. TSP-D definition

In TSP-D, we consider a complete graph $\mathcal{G}$ consisting of the set of nodes $\mathcal{N} = \{1, 2, \ldots, N\}$. Node 1 is a depot representing a warehouse, where a single truck equipped with a single drone is loaded with customer orders. Then, each of $N-1$ customers whose locations are known must be visited either by drone, truck, or both only once. 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 $\tau^{\mathrm{tr}}_{i,j}$ and $\tau^{\mathrm{dr}}_{i,j}$, respectively, where we assume $\tau^{\mathrm{dr}}_{i,j} \leq \tau^{\mathrm{tr}}_{i,j}$ 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 $d_{i,j} = \tau^{\mathrm{tr}}_{i,j} = \alpha\tau^{\mathrm{dr}}_{i,j}$. We use the Euclidean distance to compute the traveling times of both the drone and the truck, but our method straightforwardly generalizes to other distance metrics.

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. For simplicity, we allow the drone to fly for an unlimited distance. However, the drone can serve only a single customer per launch, after which the drone must rendezvous with the truck to refresh its battery. Both the drone and truck are allowed to wait for each other at a rendezvous node. We assume that the customer service times, the parcel pick-up times, and the drone battery swapping times are negligible compared to the transportation time. Therefore, only customer nodes and the depot can be used to launch, recharge, and load the drone. Fig. 1 shows a simple example and explains the MDP formulation proposed in the next section.

### 3.2. MDP formulation of TSPD

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 and deterministic transitions.

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 $t$. We let $c^{\mathrm{tr}}_t$ and $c^{\mathrm{dr}}_t$ 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 $c^{\mathrm{tr}}_t$ (or $c^{\mathrm{dr}}_t$) is set as the destination node of the vehicle. Or, if the truck (or the drone) is exactly located at a node, $c^{\mathrm{tr}}_t$ (or $c^{\mathrm{dr}}_t$) 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. $r^{\mathrm{tr}}_t$ and $r^{\mathrm{dr}}_t$ represent the remaining times to arrive at destinations $c^{\mathrm{tr}}_t$ and $c^{\mathrm{dr}}_t$ respectively. When the truck (or the drone) is located at a node, $r_t^{\text{tr}} = 0$ (or $r_t^{\text{dr}} = 0$). Then, $\mathbf{s}_t := (\mathcal{V}_t, c_t^{\text{tr}}, c_t^{\text{dr}}, r_t^{\text{tr}}, r_t^{\text{dr}})$ represents the state. At step 0, vehicles are located at node 1. Therefore, $\mathcal{V}_0 = \{1\}$, $c_0^{\text{tr}} = 1$, $c_0^{\text{dr}} = 1$, $r_t^{\text{tr}} = 0$, $r_t^{\text{dr}} = 0$ and $\mathbf{s}_0 = (\{1\}, 1, 1, 0, 0)$.

At each step, the central controller determines the next destination of the truck $a_t^{\text{tr}} \in \mathcal{N}$ and that of the drone $a_t^{\text{dr}} \in \mathcal{N}$. We cannot change the destination of a vehicle that is in transit from one node to another. When the drone is in transit, i.e. $r_t^{\text{dr}} > 0$, we restrict $a_t^{\text{dr}} = c_t^{\text{dr}}$. Analogously, $a_t^{\text{tr}} = c_t^{\text{tr}}$ if the truck is in transit. For convenience, we refer to $\mathbf{a}_t := (a_t^{\text{tr}}, a_t^{\text{dr}})$ 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 $\hat{r}_t^{\text{tr}} = r_t^{\text{tr}} + \tau_{c_t^{\text{tr}}, a_t^{\text{tr}}}$ and $\hat{r}_t^{\text{dr}} = r_t^{\text{dr}} + \tau_{c_t^{\text{dr}}, a_t^{\text{dr}}}$ be "updated" versions of $r_t^{\text{tr}}$ and $r_t^{\text{dr}}$ 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 $C_t := \min(\hat{r}_t^{\text{tr}}, \hat{r}_t^{\text{dr}})$. The remaining time is updated as $r_{t+1}^{\text{tr}} = \hat{r}_t^{\text{tr}} - C_t$, $r_{t+1}^{\text{dr}} = \hat{r}_t^{\text{dr}} - C_t$. The set of visited nodes is updated as $\mathcal{V}_{t+1} = \mathcal{V}_t \cup \{a_t^v \mid r_{t+1}^v = 0, v \in \{\text{tr}, \text{dr}\}\}$. Also, $c_{t+1}^v = a_t^v$ 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 customers and return vehicles. Then, our cost function (negative reward) is the total time spent in the system or the makespan:

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

## 4. Solution methods

In this section, we discuss in detail the proposed Attention-LSTM Hybrid Model and present training methods.

### 4.1. The attention-LSTM hybrid model

Following Vinyals et al. (2015), we aim to learn a probabilistic routing policy $\pi_\theta$ 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 $p_\theta$ is a function parameterized by $\theta$. Then, we employ policy gradient methods (Williams, 1992) to learn $\theta$. In practice, encoder-decoder architecture has shown its effectiveness in solving routing problems (Bogyrbayeva et al., 2022). Therefore, we also utilize an encoder-decoder structure that enables learning both a graph representation through encoding and policy $p_\theta$ through decoding.

In particular, to coordinate multiple vehicles with different capacities, we propose a hybrid model (HM) that uses multi-head attention for encoding as in Kool et al. (2018), 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 are 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 Fig. 2. We formally describe HM below.

*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.

*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 $\mathbf{h_n}$:

$$\mathbf{q}_n = \mathbf{W}^Q \mathbf{h}_n, \quad \mathbf{k}_n = \mathbf{W}^K \mathbf{k}_n, \quad \mathbf{v}_n = \mathbf{W}^V \mathbf{h}_n \quad \forall n \in \mathcal{N}, \tag{4}$$

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:

$$u_{i,j} = \frac{\mathbf{q}_i^\top \mathbf{k}_j}{\sqrt{d_k}} \quad \forall i, j \in \mathcal{N}. \tag{5}$$

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

$$a_{i,j} = \frac{e^{u_{ij}}}{\sum_{j'} e^{u_{i,j'}}}. \tag{6}$$

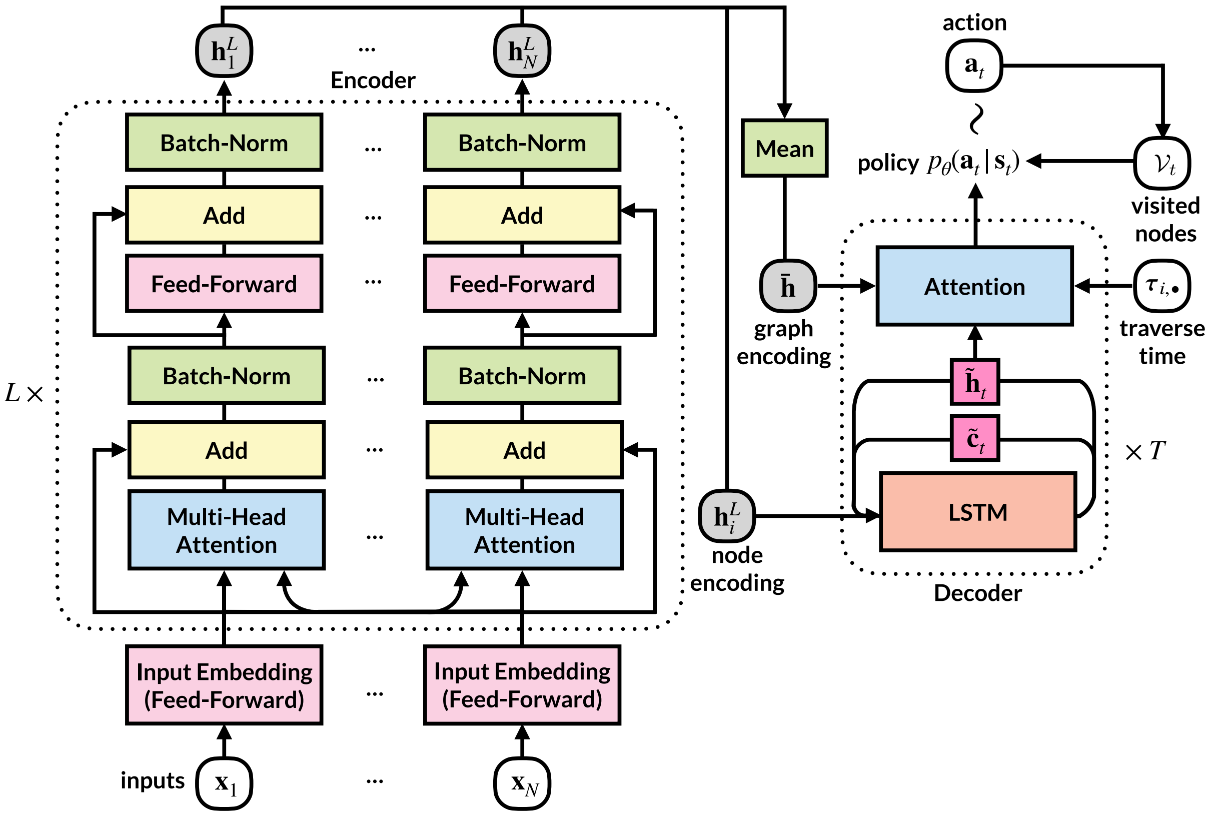


**Fig. 2.** An encoder-decoder structure of attention-LSTM Hybrid model (HM).

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

$$\mathbf{h}'_n = \sum_j a_{n,j}\mathbf{v}_j. \tag{7}$$

Instead of using a single head, we use a multi-head attention layer with head size $M$, which allows passing different messages from other nodes. Then after computing messages from each head using (6), we can combine all messages coming to node $n$ as follows:

$$\mathbf{MHA}_n(\mathbf{h}_1,\dots,\mathbf{h}_N) = \sum_{m=1}^{M} \mathbf{W}^O_m \mathbf{h}'_{nm}. \tag{8}$$

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

$$\widehat{\mathbf{h}}^l_n = \mathbf{BN}^l(\mathbf{h}^{l-1}_n + \mathbf{MHA}_n(\mathbf{h}^{l-1}_1,\dots,\mathbf{h}^{l-1}_N)). \tag{9}$$

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

$$\mathbf{h}^l_n = \mathbf{BN}^l(\widehat{\mathbf{h}}^l_n + \mathbf{FF}^l(\widehat{\mathbf{h}}^l_n)), \tag{10}$$

where

$$\mathbf{FF}^l(\widehat{\mathbf{h}}^l_n) = \mathbf{W}^{\mathrm{ff},1}\cdot \mathrm{ReLU}(W^{\mathrm{ff},0}\widehat{\mathbf{h}}^l_n + \mathbf{b}^{\mathrm{ff},0}) + \mathbf{b}^{\mathrm{ff},1}. \tag{11}$$

*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 $\mathbf{h}^L_i$ where $i$ represents the current location:

$$\widetilde{\mathbf{h}}'_{t+1}, \widetilde{\mathbf{c}}'_{t+1} = \mathrm{LSTM}(\mathbf{h}^L_i, (\widetilde{\mathbf{h}}_t, \widetilde{\mathbf{c}}_t)). \tag{12}$$

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 $p$:

$$\widetilde{\mathbf{h}}_{t+1} = \mathrm{Dropout}(\widetilde{\mathbf{h}}'_{t+1}, p), \quad \widetilde{\mathbf{c}}_{t+1} = \mathrm{Dropout}(\widetilde{\mathbf{c}}'_{t+1}, p). \tag{13}$$

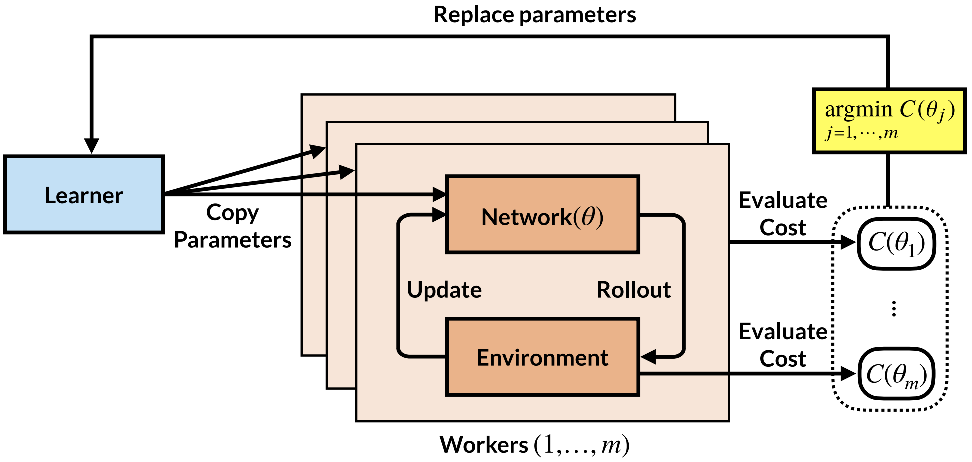


**Fig. 3.** Overview of distributed RL training for HM: each of multiple workers executes rollout with different training instances, updates its parameters, and evaluates costs. The learner replaces its network parameters with the ones from the best worker.

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:

$$\mathbf{a}_{i,\cdot} = \mathbf{v}_a^\top \tanh\left(\mathbf{W}^a[\bar{\mathbf{h}};\widetilde{\mathbf{h}}_{t+1};\mathbf{W}^d\boldsymbol{\tau}_{i,\cdot}]\right), \tag{14}$$

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:

$$p_\theta(a_t^v = j|\mathbf{s}_t) = \begin{cases} \dfrac{\exp(a_{i,j})}{\sum_{j'\in\mathcal{N}\setminus\mathcal{V}_t}\exp(a_{i,j'})} & : j \in \mathcal{N}\setminus\mathcal{V}_t, \\ 0 & : j \in \mathcal{V}_t, \end{cases} \tag{15}$$

so that we do not allow nodes to be revisited by either vehicle. In general, optimal solutions to TSP-D can have certain nodes visited more than one time (Agatz et al., 2018). 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 Section 6.1 for further discussion on revisiting.

### 4.2. Training methods

The presented HM must be trained through efficiently exploring actions and receiving feedback in a form of rewards.

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:

$$J(\theta|\mathcal{G}) = \mathbb{E}_{\pi_\theta(\mathcal{V}_T|\mathcal{G})}[(C - b(\mathcal{G}))\log\pi_\theta(\mathcal{V}_T|\mathcal{G})], \tag{16}$$

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 $\pi_\theta$; hence we can compute the gradients $\nabla_\theta J(\theta|\mathcal{G})$ using the REINFORCE algorithm (Williams, 1992).

*Distributed RL training* To accelerate and stabilize the training of HM, we devise a new training algorithm that takes advantage of parallelization. We modify the Advantage Actor-Critic (A2C) algorithm to develop the *Distribute-and-Follow Policy Gradient* (DFPG) training algorithm. 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 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 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 Fig. 3.

Our experiments show that DFPG stabilizes the final performance of the proposed HM at the convergence point, shown in Fig. 4. We also observe 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.

**Algorithm 1:** Distribute-and-Follow Policy Gradient

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

**Algorithm 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 $(\tilde{h}_0, \tilde{c}_0)$
$x_0$, mask$_0$ $\leftarrow$ Env.Reset(B)
for $t=0$ to $T$ do
    $a_t^{\text{tr}}, (\tilde{h}_{t'}, \tilde{c}_{t'}) \leftarrow \pi_{\theta_k^a}(x_t^{\text{tr}}, \text{mask}_t^{\text{tr}}, (\tilde{h}_t, \tilde{c}_t))$
    $a_t^{\text{dr}}, (\tilde{h}_t, \tilde{c}_t) \leftarrow \pi_{\theta_k^a}(x_t^{\text{dr}}, \text{mask}_t^{\text{dr}}, (\tilde{h}_{t'}, \tilde{c}_{t'}))$
    $x_{t+1}$, 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
$d\theta_k^a \leftarrow \frac{1}{M_{\text{epis}}} \sum_{m=1}^{M_{\text{epis}}} (R^m - b^m(x_0^m; \theta_k^c)) \nabla_{\theta_k^a} \log \pi_{\theta_k^a}$
$d\theta_k^c \leftarrow \frac{1}{M_{\text{epis}}} \sum_{m=1}^{M_{\text{epis}}} \nabla_{\theta_k^c} (R^m - b^m(x_0^m; \theta_k^c))^2$
```

## 5. Computational studies

We empirically demonstrate the strength of HM against learning and optimization heuristics for TSP-D. In all experiments, we assumed the drone is twice as fast as the truck; i.e. $\alpha = 2$.

### 5.1. 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. This generation method is identical to Agatz et al. (2018).

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). In particular, to identify potential demand locations for urban mobility and logistics services, we used an electric vehicle (EV) parking location dataset 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 Scott's Rule (Scott, 2015).

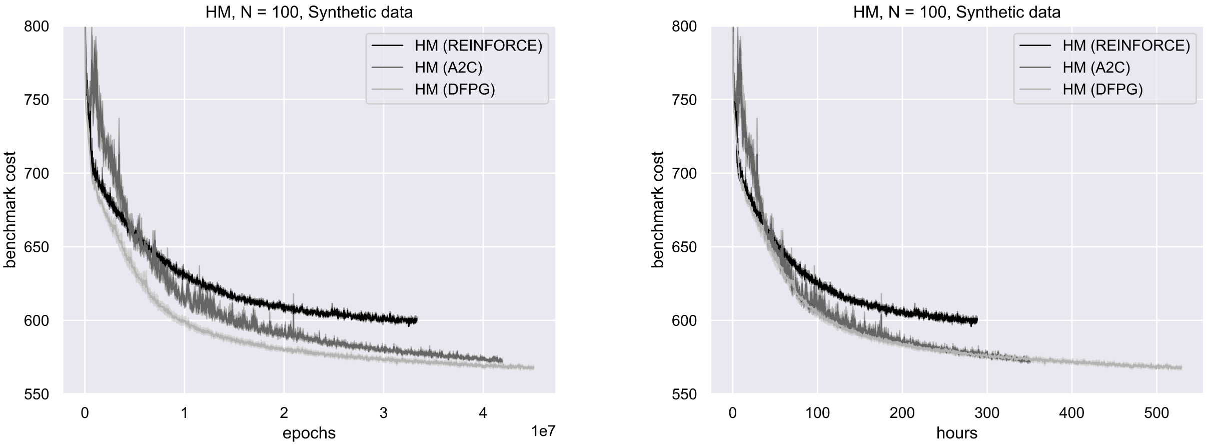


**Fig. 4.** Benchmark cost curve comparison between A2C and DFPG on 100-node graphs from the environment without revisiting. Synthetic data refers to 100 benchmark instances used in Table 2.

*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. We use Python 3.8 and PyTorch 1.5 for all our implementations.

*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:

$$\text{Gap}_i = \frac{Z_i - Z_i^*}{Z_i^*} \times 100\%, \tag{17}$$

where $Z_i$ is the cost of a solution method for each instance $i$ and $Z_i^*$ is the cost of the best-performing solution method among all methods compared for instance $i$. Then we report 'Gap' as the mean value of all $\text{Gap}_i$ values for the same setting.

*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 a high-performing *end-to-end* learning algorithm for routing problems. We also compare with NM (Nazari et al., 2018). Since our HM is built upon AM's encoder and NM's decoder, they are natural choices 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* algorithms for the TSP tour and subsequent partitioning. On the other hand, the DCH of Poikonen et al. (2019) splits the initial TSP tour into several subgroups and partition nodes within each subgroup using a branch-and-bound algorithm. We create a new DCH approach by applying TSP-ep-all to partition each subgroup instead of branch-and-bound; we call this method the Divide-Partition-and-Search (DPS) algorithm. In particular, we use DPS/$g$, 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 also increase. Note that when $g = N$, the DPS/$N$ is identical to TSP-ep-all. We implemented TSP-ep-all and DPS/$g$ 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 (80 GiB) and AMD EPYC 7713 64-Core Processor CPU (128 threads used) 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, NM 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 use a single thread of the CPU

**Table 1**
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%) | 227.90 (0.00%) |
| Mean | 226.33 | 228.38 (0.93%) | 227.84 (0.69%) |

**Table 2**
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 mean relative difference to the cost of the best algorithm for each instance as in (17). 'Time' is the average solution time of the algorithm for a single instance.

| Method | $N = 20$ | | | $N = 50$ | | | $N = 100$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | Cost | Gap | Time | Cost | Gap | Time | Cost | Gap | Time |
| TSP-ep-all | 281.62$_{\pm 18.05}$ | 0.64% | (0.17 s) | 397.21$_{\pm 20.19}$ | 1.43% | (43 s) | **535.50**$_{\pm 21.83}$ | **0.78%** | (3992 s) |
| DPS/10 | 292.23$_{\pm 19.00}$ | 4.48% | (0.02 s) | 420.51$_{\pm 23.98}$ | 7.39% | (0.08 s) | 570.74$_{\pm 20.61}$ | 7.55% | (0.35 s) |
| DPS/25 | – | – | | 404.78$_{\pm 22.03}$ | 3.37% | (1.04 s) | 548.23$_{\pm 22.36}$ | 3.19% | (2.27 s) |
| AM (greedy) | 294.88$_{\pm 24.44}$ | 5.36% | (0.18 s) | 439.21$_{\pm 28.09}$ | 12.16% | (0.48 s) | 642.82$_{\pm 25.12}$ | 21.08% | (0.74 s) |
| NM (greedy) | 304.70$_{\pm 22.74}$ | 8.89% | (0.20 s) | 445.44$_{\pm 24.51}$ | 13.78% | (0.30 s) | – | – | – |
| HM (greedy) | 285.59$_{\pm 18.10}$ | 2.07% | (0.18 s) | 408.51$_{\pm 18.78}$ | 4.33% | (0.49 s) | 564.42$_{\pm 22.03}$ | 6.27% | (1.07 s) |
| AM (4800) | 285.06$_{\pm 18.14}$ | 1.54% | (0.30 s) | 411.50$_{\pm 20.02}$ | 4.03% | (0.78 s) | 602.95$_{\pm 19.92}$ | 13.55% | (1.92 s) |
| HM (100) | 282.65$_{\pm 17.71}$ | 1.01% | (0.26 s) | 399.11$_{\pm 17.59}$ | 1.93% | (0.61 s) | 550.01$_{\pm 19.49}$ | 3.56% | (1.23 s) |
| HM (1200) | 281.90$_{\pm 17.45}$ | 0.75% | (0.29 s) | 396.80$_{\pm 17.10}$ | 1.35% | (0.88 s) | 546.28$_{\pm 18.99}$ | 2.86% | (1.95 s) |
| HM (2400) | 281.75$_{\pm 17.47}$ | 0.70% | (0.30 s) | 396.16$_{\pm 17.08}$ | 1.18% | (1.14 s) | 545.32$_{\pm 19.02}$ | 2.68% | (2.58 s) |
| HM (4800) | **281.53**$_{\pm 17.35}$ | **0.62%** | (0.40 s) | **395.65**$_{\pm 17.00}$ | **1.05%** | (1.72 s) | 544.55$_{\pm 19.01}$ | 2.53% | (4.09 s) |

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. The entire code, including the environment for HM and AM, supports GPU parallelization, boosting the solution time up to 4.5 times from the environment with CPU computations for HM (4800) and AM (4800) on $N = 100$ instances. 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.

### 5.2. Results on TSP-D with unlimited flying range

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%. Figs. 5 and 6 visualize the optimal solutions reported in the literature and the solution produced by HM with greedy decoding. Agatz et al. (2018) provide 10 instances of size $N = 11$ with exact optimal solutions, available at the 'TSP-D-Instances' repository: https://github.com/pcbouman-eur/TSP-D-Instances. We provide the performance of HM greedy and HM sampling on these benchmark instances in Table 1. On average, the optimality gap of HM greedy and HM sampling is 0.93% and 0.69%, respectively.

The detailed routes are provided in Figs. 5 and 6. 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.

HM outperforms AM and NM 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 2, HM (greedy) has a gap of 6.27%, while AM (greedy) has a gap of 21.08% when compared to the best performing method. Since NM performed even worse than AM for $N = 20$ and $N = 50$, we did not test NM for $N = 100$. Figs. 7(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 DPS/10 and DPS/25, the batch sampling approach using HM produces the best performance, as shown in Table 2, without increasing the solution time as much as TSP-ep-all. HM's scalability is comparable to DPS/25, but HM outperformed DPS/25 in terms of the average cost. Fig. 8 also illustrates HM (sampling) is more effective than DPS and AM (sampling). No direct comparison of solution time between DPS and HM (sampling) is possible, although HM (sampling)

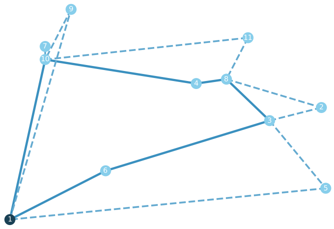
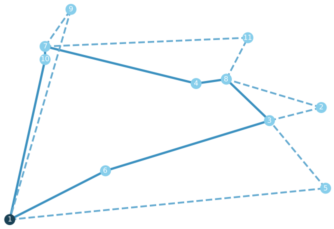
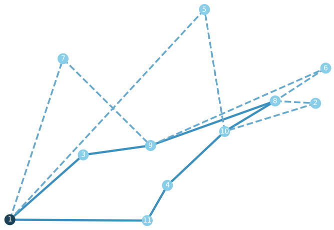
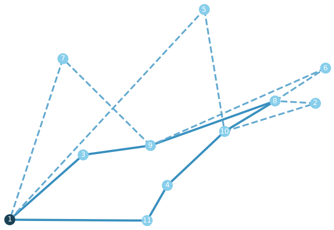
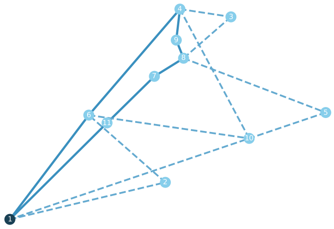
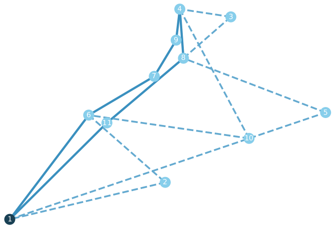
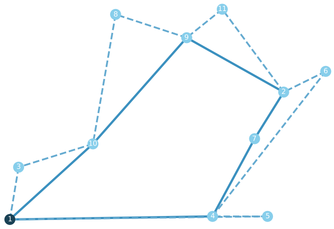
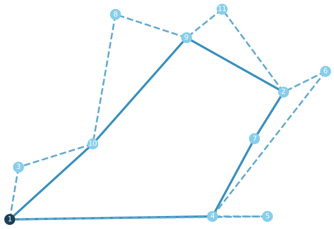
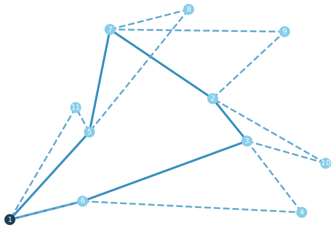
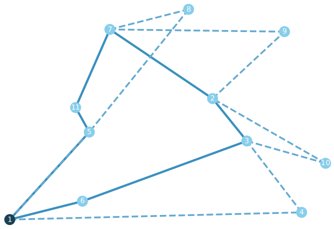
| Instance | Optimal Solution | HM Greedy |
|---|---|---|
| 1 | <br>Cost = 221.19 | <br>Cost = 223.41 (Gap = 1.00%) |
| 2 | <br>Cost = 205.76 | <br>Cost = 205.76 (Gap = 0.00%) |
| 3 | <br>Cost = 192.96 | <br>Cost = 193.99 (Gap = 0.53%) |
| 4 | <br>Cost = 241.26 | <br>Cost = 241.26 (Gap = 0.00%) |
| 5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

**Fig. 5.** 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.

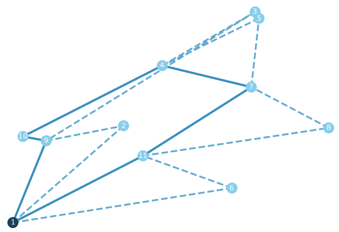
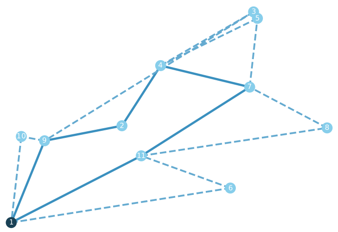
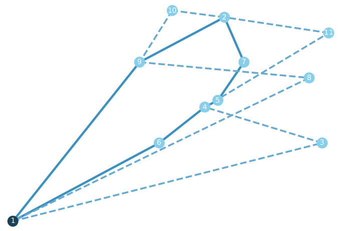
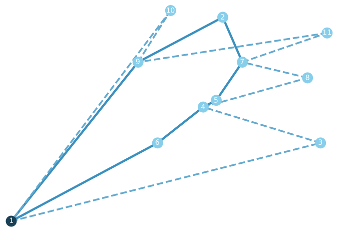
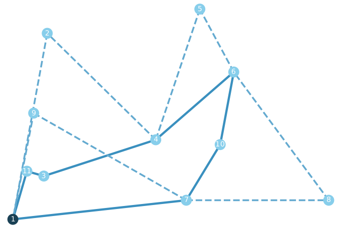
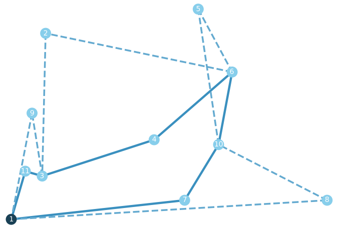
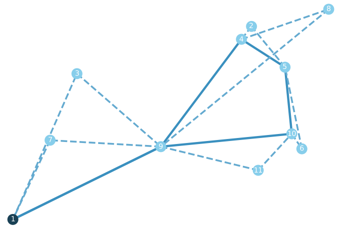
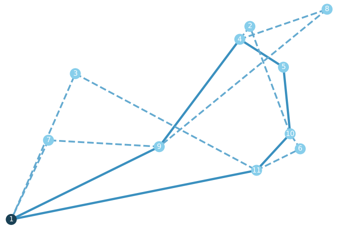
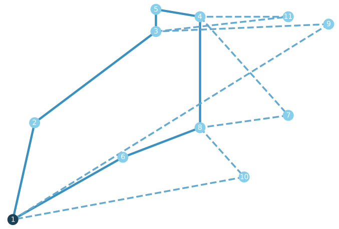
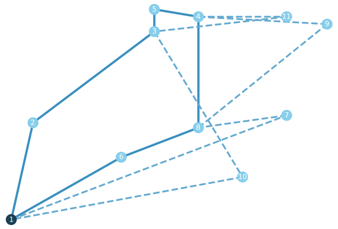
| Instance | Optimal Solution | HM Greedy |
|---|---|---|
| 6 | <br>Cost = 217.69 | <br>Cost = 217.69 (Gap = 0.00%) |
| 7 | <br>Cost = 237.34 | <br>Cost = 240.47 (Gap = 1.32%) |
| 8 | <br>Cost = 214.77 | <br>Cost = 226.64 (Gap = 5.53%) |
| 9 | <br>Cost = 256.34 | <br>Cost = 256.83 (Gap = 0.19%) |
| 10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

**Fig. 6.** 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.

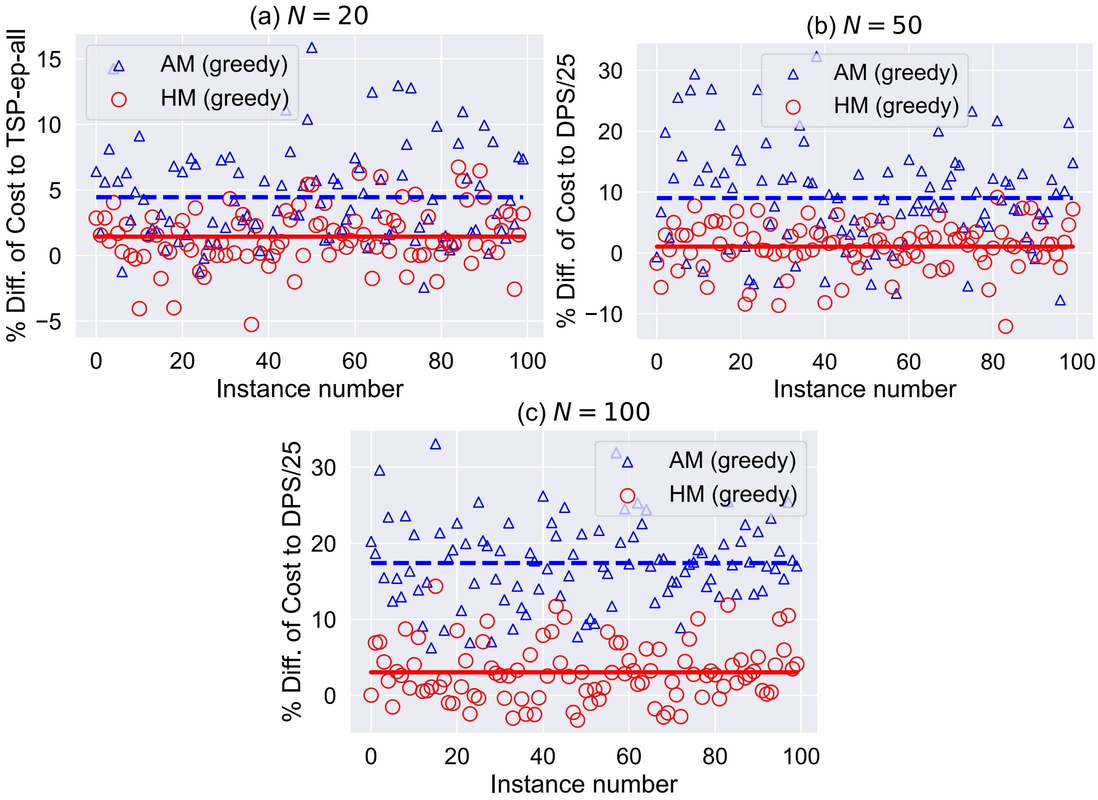


**Fig. 7.** AM (greedy) vs. HM (greedy). The percentage difference to TSP-ep-all ($N = 20$) or DPS/25 ($N = 50, 100$) is reported for each instance. The dashed and solid lines represent the mean values of AM and HM, respectively.

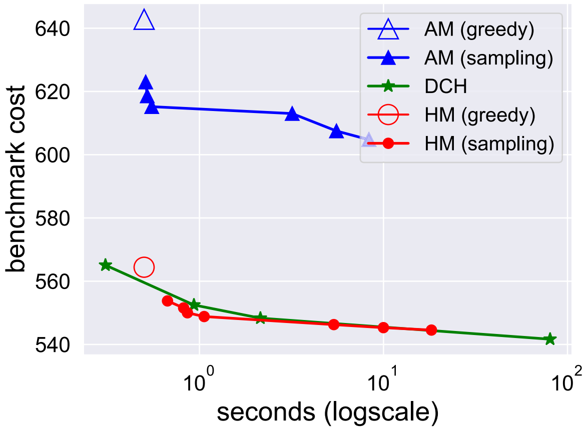


**Fig. 8.** DPS/$g$ vs. AM ($s$) vs. HM ($s$) for various $g$ and $s$ values. The average cost to the solution time is reported.

shows high efficiency in our experiments. As DPS can be also run in parallel on multiple CPU threads, the speed of DPS can be further boosted. However, such parallelization for DPS is not as straightforward as in batch sampling on GPU.

The above observations are consistent in the real Amsterdam dataset in Table 3: HM provides better solutions than AM. For $N = 50$, AM (greedy) and HM (greedy) had gap of 25.24% and 8.15%, 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 DPS. 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).

In Table 4, we also report the performance of HM on larger instances ($N = 20, 50, 100$) from the 'TSP-D-Instances' repository, in particular the instances generated by uniform distributions. As before, we assumed $\alpha = 2$. Note that these instances are generated

**Table 3**
TSP-D results on the Amsterdam dataset. Averages of 100 problem instances.

| Method | $N = 10$ | | | $N = 20$ | | | $N = 50$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | Cost | Gap | Time | Cost | Gap | Time | Cost | Gap | Time |
| TSP-ep-all | **2.02**$_{\pm 0.23}$ | **0.69%** | (0.01 s) | **2.35**$_{\pm 0.21}$ | **0.99%** | (0.15 s) | **3.26**$_{\pm 0.22}$ | **0.83%** | (42 s) |
| DPS/10 | – | – | | 2.49$_{\pm 0.24}$ | 6.82% | (0.03 s) | 3.55$_{\pm 0.25}$ | 9.90% | (0.11 s) |
| DPS/25 | – | – | | – | – | | 3.37$_{\pm 0.22}$ | 4.14% | (1.04 s) |
| AM (greedy) | 2.16$_{\pm 0.29}$ | 7.38% | (0.12 s) | 2.61$_{\pm 0.32}$ | 11.95% | (0.18 s) | 4.06$_{\pm 0.43}$ | 25.24% | (0.47 s) |
| HM (greedy) | 2.08$_{\pm 0.26}$ | 3.46% | (0.13 s) | 2.49$_{\pm 0.26}$ | 6.97% | (0.19 s) | 3.50$_{\pm 0.28}$ | 8.15% | (0.49 s) |
| AM (4800) | 2.08$_{\pm 0.25}$ | 3.63% | (0.18 s) | 2.44$_{\pm 0.23}$ | 4.62% | (0.27 s) | 3.72$_{\pm 0.34}$ | 14.85% | (0.89 s) |
| HM (100) | 2.05$_{\pm 0.25}$ | 2.14% | (0.15 s) | 2.41$_{\pm 0.24}$ | 3.27% | (0.28 s) | 3.36$_{\pm 0.25}$ | 3.84% | (0.65 s) |
| HM (1200) | 2.05$_{\pm 0.25}$ | 1.75% | (0.22 s) | 2.39$_{\pm 0.23}$ | 2.53% | (0.33 s) | 3.33$_{\pm 0.24}$ | 2.81% | (0.82 s) |
| HM (2400) | 2.04$_{\pm 0.24}$ | 1.56% | (0.24 s) | 2.39$_{\pm 0.23}$ | 2.36% | (0.39 s) | 3.32$_{\pm 0.24}$ | 2.55% | (0.99 s) |
| HM (4800) | 2.04$_{\pm 0.24}$ | 1.58% | (0.32 s) | 2.38$_{\pm 0.23}$ | 2.24% | (0.52 s) | 3.31$_{\pm 0.24}$ | 2.40% | (1.41 s) |

**Table 4**
TSP-D results on the instances of the ‘TSP-D-Instances’ repository (uniform distribution, $\alpha = 2$).

| Method | $N = 20$ | | | $N = 50$ | | | $N = 100$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | Cost | Gap | Time | Cost | Gap | Time | Cost | Gap | Time |
| TSP-ep-all | **276.70** | **0.85%** | (0.2 s) | **408.16** | **0.79%** | (40 s) | **548.09** | **0.63%** | (3635 s) |
| DPS/25 | – | – | | 421.94 | 4.23% | (1.1 s) | 559.58 | 2.74% | (2.0 s) |
| HGVNS | 293.60 | | (0.9 s[a]) | 420.80 | | (2.3 s[a]) | 553.43 | | (37.8 s[a]) |
| HM (greedy) | 277.87 | 1.40% | (0.2 s) | 426.93 | 5.53% | (0.4 s) | 579.59 | 6.48% | (0.8 s) |
| HM (4800) | 276.09 | 0.73% | (1.0 s) | 408.67 | 1.08% | (3.9 s) | 556.82 | 2.20% | (14.4 s) |

[a] The computational time for HGVNS is as reported in de Freitas and Penna (2020).

using the identical method as the random data set in Table 2, but there are only 10 instances for each size instead of 100 instances. We compare the results with the performance of the hybrid general variable neighborhood search (HGVNS) method as reported in de Freitas and Penna (2020) where the algorithm is implemented in C++ and the computational time (marked by ‘a’) is measured using the Intel i7 3.6 GHz CPU and 16 GB RAM with Ubuntu. We observe that HM (4800) finds solutions similar to TSP-ep-all for $N = 20$ and $N = 50$, while it does not perform as good for $N = 100$, on average. On the other hand, HGVNS finds good solutions for $N = 100$, but not as good for $N = 20$ and $N = 50$, on average. Since de Freitas and Penna (2020) reported the average cost values only, the mean gap values in Table 4 are based on the solutions by other algorithms.

### 5.3. Results on TSP-D with limited flying range

The presented HM can be easily adapted to solve TSP-D with the limited flying range of the drone. In this case, the state of MDP also includes the remaining battery of the drone at time $t$. Thus, we have $\mathbf{s}_t := (\mathcal{V}_t, c_t^{\mathrm{tr}}, c_t^{\mathrm{dr}}, r_t^{\mathrm{tr}}, r_t^{\mathrm{dr}}, \beta_t^{\mathrm{dr}})$, where $\beta_t^{\mathrm{dr}}$ is the battery of the drone at time $t$. We update the battery as follows $\beta_{t+1}^{\mathrm{dr}} = \beta_t^{\mathrm{dr}} - C_t$ if the drone and truck do not travel together and the drone does not wait at its current location. Also, we set the battery to its maximum level when the truck retrieves the drone. Next, we pass the state information as an input to the decoder at each step. In particular, after element-wise projection of the current battery, we use this projection along with the hidden state of the LSTM, the graph embedding, and the traveling time from the current location of the drone to compute the attention vector similar to (14) as follows:

$$\mathbf{a}_{i,\cdot} = \mathbf{v}_a^{\top} \tanh\left(\mathbf{W}^a[\bar{\mathbf{h}};\widetilde{\mathbf{h}}_{t+1};\mathbf{W}^d\boldsymbol{\tau}_{i,\cdot},\mathbf{W}^{\beta}\boldsymbol{\beta}_{\mathbf{t}}]\right), \tag{18}$$

We also note that the current battery of the drone is passed to the decoder only when selecting the next node to be visited by the drone. In order to speed up the training, we also use the current battery level of the drone for masking. For instance, any unserved nodes located further than the current battery level of the drone cannot be selected as the next action. Further, we also make sure to check that there is at least one unserved node where the drone can fly after serving such a node, making sure the drone will have enough battery to be retrieved by the truck.

Table 5 presents the performance comparison of TSP-ep-all and HM on benchmark instances from ‘TSP-D-Instances’ repository with uniform distribution and restricted maximum radios for the drone’s flying range. We used 60%, 20%, and 20% to set maximum flying ranges for graphs with 10, 20, and 50 nodes, respectively. As shown in Table 5, HM produces comparable results with TSP-ep-all heuristic in solution quality for small-sized instances.

## 6. Additional experiments

In this section, we provide results from additional experiments to show the performance of the proposed neural network structure and the training algorithm. First, we demonstrate the performance of the proposed DFPG training algorithm with A2C, when *revisiting nodes* is allowed. Second, we show that HM is also effective for another class of routing problems, the min-max Capacitated Vehicle Routing Problem (mmCVRP). Third, we test if HM can perform well on the classical TSP instances.

**Table 5**
TSP-D results with limited flying range for drone.

| Method | $N = 10$ | | | $N = 20$ | | | $N = 50$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | Cost | Gap | Time | Cost | Gap | Time | Cost | Gap | Time |
| TSP-ep-all | **277.12±28.75** | **0.00%** | (0.01 s) | **387.57±28.19** | **0.00%** | (0.05 s) | **514.65±25.85** | **0.00%** | (2.79 s) |
| HM (greedy) | 279.32±27.53 | 0.85% | (0.07 s) | 395.35±33.38 | 1.94% | (0.09 s) | 550.31±32.76 | 6.93% | (0.21 s) |
| HM (100) | 278.61±28.29 | 0.56% | (0.21 s) | 393.65±32.29 | 1.51% | (0.40 s) | 535.04±29.79 | 3.96% | (1.05 s) |
| HM (1200) | 278.18±28.26 | 0.40% | (0.21 s) | 393.00±31.92 | 1.36% | (0.36 s) | 531.39±29.62 | 3.25% | (1.39 s) |
| HM (2400) | 278.13±28.29 | 0.38% | (0.20 s) | 393.09±32.05 | 1.37% | (0.43 s) | 530.98±28.75 | 3.18% | (1.84 s) |
| HM (4800) | 278.15±28.26 | 0.40% | (0.27 s) | 392.93±32.10 | 1.33% | (0.57 s) | 529.97±29.71 | 2.97% | (3.23 s) |

**Table 6**
The HM performance when revisiting the nodes is allowed on the instances from Table 1.

| Instance | HM (greedy) (Gap) | HM (1200) (Gap) |
|---|---|---|
| 1 | 223.41 (1.00%) | 223.41 (1.00%) |
| 2 | 206.82 (0.52%) | 206.82 (0.52%) |
| 3 | 214.46 (11.14%) | 195.70 (1.42%) |
| 4 | 241.26 (0.00%) | 241.26 (0.00%) |
| 5 | 257.91 (3.94%) | 248.82 (0.27%) |
| 6 | 218.37 (0.31%) | 218.37 (0.31%) |
| 7 | 246.53 (3.87%) | 237.34 (0.00%) |
| 8 | 223.88 (4.24%) | 223.88 (4.24%) |
| 9 | 261.12 (1.86%) | 257.02 (0.27%) |
| 10 | 227.90 (0.00%) | 227.90 (0.00%) |
| Mean | 232.17 (2.69%) | 228.05 (0.80%) |

### *6.1. Distributed training algorithms for TSP-D when revisiting is allowed*

In Instance 9 of Fig. 6, 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 Fig. 6, we compared the performances of training algorithms in Fig. 9. 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. Table 6 shows the testing results of HM greedy and sampling on the Agatz's instances with 11 nodes, which in general is worse than the results with the environment not allowing revisiting nodes reported in Table 1.

### *6.2. Additional results on mmCVRP*

While our main focus is TSP-D, we also test the 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 details of the experiments follow.

*Problem statement* We have a set of nodes representing customer locations and a depot. Each customer has a 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 and 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.

*Data generation* We generate the random data for mmCVRP by sampling the $x$ and $y$ 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:

$$Q = \left\lceil 1.2/m \sum_{n \in \mathcal{N}} D_n \right\rceil, \tag{19}$$

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

$$\tau_{i,j} = \left\lfloor 2000 d_{ij} \right\rfloor \quad \forall i \in \mathcal{N}, j \in \mathcal{N}, \tag{20}$$

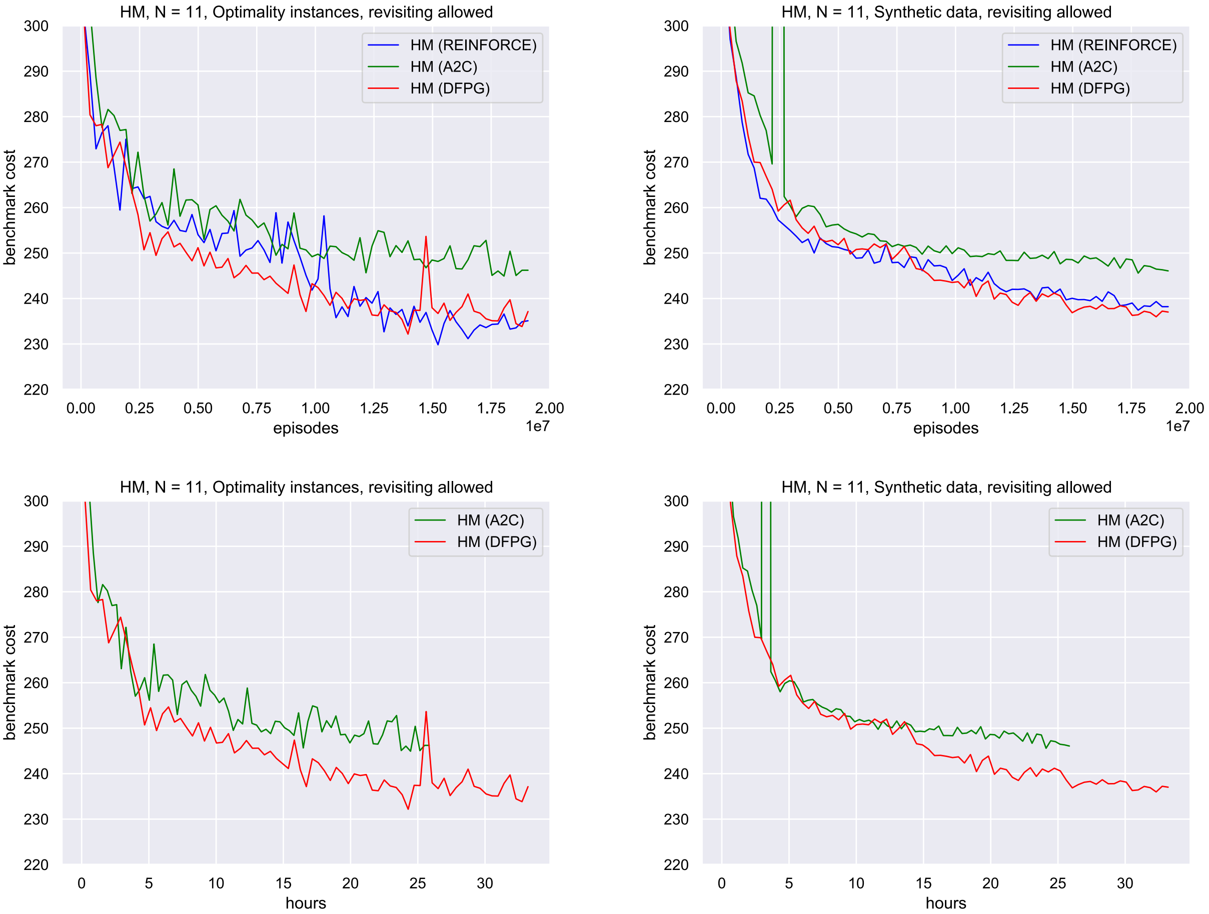


**Fig. 9.** The average benchmark cost curves of HM with REINFORCE, A2C, and DFPG on 11-node graphs from the environment that *allows revisiting*. Optimality instances refer to 11-node instances from Agatz et al. (2018). Synthetic data refers to 100 benchmark instances used in Table 2. The top figures represent the average benchmark costs over the number of episodes. Bottom figures compare the average benchmark costs over wall-clock time.

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 $i \in \mathcal{N}$, we compute expected capacity if it is visited by a decision taker at time $t$: $Q'_{t,i} = Q_t - D_{t,i}$, where $Q_t$ is a capacity of a decision taker at time $t$ and $D_{t,i}$ is demand at node $i$ at time $t$. Then we use a vector of expected remaining capacities $\mathbf{Q}'_t = (Q'_{t,i} : i \in \mathcal{N})$ to compute attention in the decoder:

$$\mathbf{a}_{i,\cdot} = \mathbf{v}_a^\top \tanh\left(\mathbf{W}^a[\bar{\mathbf{h}}; \widetilde{\mathbf{h}}_{t+1}; \mathbf{W}^d \mathbf{Q'}_t]\right), \tag{21}$$

*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:

$$\mathbf{h}_c = \left[\bar{\mathbf{h}}, \mathbf{h}_i^L, \mathbf{h}_{j_1}^L ... \mathbf{h}_{j_{m-1}}^L, Q_t\right] \tag{22}$$

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 $Q_t$ at time $t$.

*Results* The results in Table 7 demonstrate that HM provides competitive results with OR-Tools to solve the mmCVRP, while AM was not as competitive. This indicates that the proposed HM can be effective in learning solutions to other coordinated routing problems, especially when the objective is to minimize the makespan.

### 6.3. 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

**Table 7**
MmCVRP results with three vehicles. Averages of 128 problem instances.

| Method | $N = 20$ | | | $N = 30$ | | |
|---|---|---|---|---|---|---|
| | Cost | Gap | Time | Cost | Gap | Time |
| OR-Tools | **4086.99**$\pm$**471.78** | **0.00%** | (1.0 s) | 4871.06$\pm$453.74 | 7.26% | (1.0 s) |
| AM (greedy) | 4396.11$\pm$535.81 | 7.56% | (0.1 s) | 4939.37$\pm$473.15 | 8.76% | (0.2 s) |
| HM (greedy) | 4298.83$\pm$523.19 | 5.18% | (0.1 s) | 4779.27$\pm$488.44 | 5.24% | (0.2 s) |
| AM (1200) | 4142.74$\pm$478.88 | 1.36% | (14.3 s) | 4574.05$\pm$420.01 | 0.01% | (27.6 s) |
| HM (1200) | 4129.41$\pm$496.54 | 1.04% | (11.4 s) | **4541.45**$\pm$**429.89** | **0.00%** | (25.2 s) |

**Table 8**
TSP results on random locations. An average of 10,000 problem instances is reported. Cost refers to the cost function (1). The gap is the relative difference to the cost of the best algorithm for the setting.

| Method | $N = 20$ | | $N = 50$ | | $N = 100$ | |
|---|---|---|---|---|---|---|
| | 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 | |

AM and HM for TSP by ourselves, we used the result reported by Nazari et al. (2018) for NM. Random location instances are generated using the same rule. **HM for TSP** is similar to TSP-D, except we pass a distance vector, $\mathbf{d}_{i,\cdot}$, from the current location of an agent $i$ to other nodes as follows in computing attention in decoder:

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

The result in Table 8 shows that HM is not as competitive as AM in the classical TSP instances, while HM performs as similar as NM.

## 7. Conclusions

This study proposed a new end-to-end learning model that can learn the 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 routing problems, the proposed model efficiently deals with the coordination of multiple vehicles and achieves comparable performances as strong optimization heuristics methods. This could be 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 performance of end-to-end learning methods. Understanding its impact will be critical for real-world applications.

## CRediT authorship contribution statement

**Aigerim Bogyrbayeva:** Methodology, Software, Data curation, Writing – original draft. **Taehyun Yoon:** Software, Data curation, Writing – original draft. **Hanbum Ko:** Software, Data curation, Writing – original draft. **Sungbin Lim:** Methodology, Writing – review & editing. **Hyokun Yun:** Conceptualization, Writing – review & editing. **Changhyun Kwon:** Conceptualization, Software, Writing – review & editing.

## Data availability

Data will be made available on request.

## Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP), South Korea grant (2020-0-01336, Artificial Intelligence Graduate School Program, UNIST, South Korea), 2021 Research Fund (1.210107.01) of UNIST, the National Research Foundation of Korea (2021R1A4A3033149 and 2021H1D3A2A01039401), the National Science Foundation of the U.S. (CMMI-2032458), and the internal grant of Suleyman Demirel University 2021–2022.

## References

Agatz, N., Bouman, M., Schmidt, P., 2018. Optimization approaches for the traveling salesman problem with drone. Transp. Sci. 52 (4), 965–981.

Amazon Prime Air, 2022. Amazon prime air. https://www.aboutamazon.com/news/transportation/amazon-prime-air-prepares-for-drone-deliveries. Accessed: August 16, 2022.

Applegate, D., Bixby, V., Chvátal, W., Cook, R., 2001. TSP cuts which do not conform to the template paradigm. In: Computational Combinatorial Optimization. Springer, pp. 261–303.

Bello, I., Pham, Q.V., Le, M., Norouzi, S., Bengio, H., 2016. Neural combinatorial optimization with reinforcement learning. arXiv preprint arXiv:1611.09940.

Bezanson, J., Edelman, S., Karpinski, V.B., Shah, A., 2017. Julia: A fresh approach to numerical computing. SIAM Rev. 59 (1), 65–98.

Boccia, M., Masone, A., Sforza, C., Sterle, A., 2021. A column-and-row generation approach for the flying sidekick travelling salesman problem. Transp. Res. C 124, 102913.

Bogyrbayeva, A., Jang, A., Shah, Y.J., Jang, C., Kwon, S., 2021. A reinforcement learning approach for rebalancing electric vehicle sharing systems. IEEE Trans. Intell. Transp. Syst. Accepted, 1–11.

Bogyrbayeva, A., Meraliyev, T., Mustakhov, B., Dauletbayev, M., 2022. Learning to solve vehicle routing problems: A survey. arXiv preprint arXiv:2205.02453.

Bouman, P., Agatz, M., Schmidt, N., 2018. Dynamic programming approaches for the traveling salesman problem with drone. Networks 72 (4), 528–542.

Business Insider, 2017. Ups tests drone delivery system. https://www.businessinsider.com/ups-tests-drone-delivery-system-2017-2. Accessed: August 18, 2022.

Carlsson, J.G., Song, S., 2018. Coordinated logistics with a truck and a drone. Manage. Sci. 64 (9), 4052–4069.

Chung, S.H., Sah, B., 2020. Optimization for drone and drone-truck combined operations: A review of the state of the art and future directions. Comput. Oper. Res. 105004.

de Freitas, J.C., Penna, P.H.V., 2020. A variable neighborhood search for flying sidekick traveling salesman problem. Int. Trans. Oper. Res. 27 (1), 267–290.

Dell'Amico, M., Montemanni, S., Novellani, R., 2019. Models and algorithms for the flying sidekick traveling salesman problem. http://dx.doi.org/10.48550/ARXIV.1910.02559, URL https://arxiv.org/abs/1910.02559.

Dell'Amico, M., Montemanni, S., Novellani, R., 2021a. A random restart local search matheuristic for the flying sidekick traveling salesman problem. In: 2021 the 8th International Conference on Industrial Engineering and Applications (Europe). pp. 205–209.

Dell'Amico, M., Montemanni, S., Novellani, R., 2021b. Algorithms based on branch and bound for the flying sidekick traveling salesman problem. Omega 104, 102493.

Dell'Amico, M., Montemanni, S., Novellani, R., 2021c. Drone-assisted deliveries: New formulations for the flying sidekick traveling salesman problem. Optim. Lett. 15 (5), 1617–1648.

Devlin, J., Chang, K., Lee, K., Toutanova, M.-W., 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

El-Adle, A.M., Ghoniem, M., Haouari, A., 2021. Parcel delivery by vehicle and drone. J. Oper. Res. Soc. 72 (2), 398–416.

Gonzalez-R, P.L., Canca, J.L., Andrade-Pineda, M., Calle, J.M., Leon-Blanco, D., 2020. Truck-drone team logistics: A heuristic approach to multi-drop route planning. Transp. Res. C 114, 657–680.

Ha, Q.M., Deville, Q.D., Pham, M.H., Hà, Y., 2018. On the min-cost traveling salesman problem with drone. Transp. Res. C 86, 597–621.

Haider, Z., Charkhgard, S.W., Kim, C., Kwon, H., 2019. Optimizing the relocation operations of free-floating electric vehicle sharing systems. Available at SSRN.

Hottung, A., Tierney, K., 2020. Neural large neighborhood search for the capacitated vehicle routing problem. In: ECAI 2020. IOS Press, pp. 443–450.

Joerss, M., Schroder, F., Neuhaus, C., Klink, F., Mann, J., 2016. Parcel Delivery: The Future of Last Mile. Tech. rep., McKinsey & Company, Travel, Transport and Logistics, URL https://www.mckinsey.com//media/mckinsey/industries/travel%20logistics%20{and}%20infrastructure/our%20insights/how%20customer%20dem{and}s%20are%20reshaping%20last%20mile%20delivery/parcel_delivery_the_future_of_last_mile.pdf.

Khalil, E., Dai, Y., Zhang, B., Dilkina, L., Song, H., 2017. Learning combinatorial optimization algorithms over graphs. Adv. Neural Inf. Process. Syst. 30, 6348–6358.

Kim, M., Park, J., Kim, J., 2021. Learning collaborative policies to solve NP-hard routing problems. In: Thirty-Fifth Conference on Neural Information Processing Systems.

Kool, W., van Hoof, J., Gromicho, M., Welling, H., 2021. Deep policy dynamic programming for vehicle routing problems. arXiv preprint arXiv:2102.11756.

Kool, W., van Hoof, M., Welling, H., 2018. Attention, learn to solve routing problems!. In: International Conference on Learning Representations.

Kwon, Y.-D., Choo, B., Kim, I., Yoon, Y., Gwon, S., Min, J., 2020. Pomo: Policy optimization with multiple optima for reinforcement learning. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (Eds.), Advances in Neural Information Processing Systems, Vol. 33. Curran Associates, Inc., pp. 21188–21198.

Liu, Z., Li, A., Khojandi, X., 2022. The flying sidekick traveling salesman problem with stochastic travel time: A reinforcement learning approach. Transp. Res. E 164, 102816.

Lu, H., Zhang, S., Yang, X., 2020. A learning-based iterative method for solving vehicle routing problems. In: International Conference on Learning Representations. URL https://openreview.net/forum?id=BJe1334YDH.

Macrina, G., Pugliese, F., Guerriero, G., Laporte, L.D.P., 2020. Drone-aided routing: A literature review. Transp. Res. C 120, 102762.

Mazyavkina, N., Sviridov, S., Ivanov, E., Burnaev, S., 2021. Reinforcement learning for combinatorial optimization: A survey. Comput. Oper. Res. 105400.

Mnih, V., Kavukcuoglu, D., Silver, A., Graves, I., Antonoglou, D., Wierstra, M., Riedmiller, K., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Murray, C.C., Chu, A.G., 2015. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. Transp. Res. C 54, 86–109.

Nazari, M., Oroojlooy, L., Snyder, M., Takác, A., 2018. Reinforcement learning for solving the vehicle routing problem. Adv. Neural Inf. Process. Syst. 983, 9–9849.

Perron, L., Furnon, V., 2019. OR-tools. URL https://developers.google.com/optimization/.

Poikonen, S., Golden, E.A., Wasil, B., 2019. A branch-and-bound approach to the traveling salesman problem with a drone. INFORMS J. Comput. 31 (2), 335–346.

Roberti, R., Ruthmair, M., 2021. Exact methods for the traveling salesman problem with drone. Transp. Sci. 55 (2), 315–335.

Schermer, D., Moeini, O., Wendt, M., 2020. A b ranch-and-cut approach and alternative formulations for the traveling salesman problem with drone. Networks 76 (2), 164–186.

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

Sykora, Q., Ren, R., Urtasun, M., 2020. Multi-agent routing value iteration network. In: International Conference on Machine Learning. PMLR, pp. 9300–9310.

Vásquez, S.A., Angulo, M.A., Klapp, G., 2021. An exact solution method for the tsp with drone based on decomposition. Comput. Oper. Res. 127, 105127.

Vaswani, A., Shazeer, N., Parmar, J., Uszkoreit, L., Jones, A.N., Gomez, L., Kaiser, I., Polosukhin, N., 2017. Attention is all you need. arXiv preprint arXiv:1706.03762.

Vinyals, O., Fortunato, N., Jaitly, M., 2015. Pointer networks. arXiv preprint arXiv:1506.03134.

Virtanen, P., Gommers, T.E., Oliphant, M., Haberland, T., Reddy, D., Cournapeau, E., Burovski, P., Peterson, W., Weckesser, J., Bright, S.J., van der Walt, M., Brett, J., Wilson, K.J., Millman, N., Mayorov, A.R.J., Nelson, E., Jones, R., Kern, E., Larson, C.J., Carey, İ., Polat, Y., Feng, E.W., Moore, J., VanderPlas, D., Laxalde, J., Perktold, R., Cimrman, I., Henriksen, E.A., Quintero, C.R., Harris, A.M., Archibald, A.H., Ribeiro, F., Pedregosa, P., van Mulbregt, SciPy 1.0 Contributors, R., 2020. SciPy 1.0: Fundamental algorithms for scientific computing in python. Nature Methods 17, 261–272. http://dx.doi.org/10.1038/s41592-019-0686-2.

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

Wing Aviation, 2021. Wing delivers library books to students in Virginia. https://blog.wing.com/2020/06/wing-delivers-library-books-to-students.html. Accessed: March 10, 2021.

Yurek, E.E., Ozmutlu, H.C., 2018. A decomposition-based iterative optimization algorithm for traveling salesman problem with drone. Transp. Res. C 91, 249–262.

Zhang, K., He, Z., Zhang, X., Lin, M., Li, F., 2020. Multi-vehicle routing problems with soft time windows: A multi-agent reinforcement learning approach. Transp. Res. C 121, 102861.