DeepFreight: A Model-free Deep-reinforcement-learning-based Algorithm for Multi-transfer Freight Delivery

Jiayu Chen,[1] **Abhishek K. Umrawal,** [1] **Tian Lan,** [2] **Vaneet Aggarwal** [1]
[1] Purdue University, West Lafayette IN 47907, USA
[2] The George Washington University, Washinton DC 20052, USA
chen3686@purdue.edu, aumrawal@purdue.edu, tlan@gwu.edu, vaneet@purdue.edu

Abstract

With the freight delivery demands and shipping costs increasing rapidly, intelligent control of fleets to enable efficient and cost-conscious solutions becomes an important problem. In this paper, we propose DeepFreight, a model-free deep-reinforcement-learning-based algorithm for multi-transfer freight delivery, which includes two closely-collaborative components: truck-dispatch and package-matching. Specifically, a deep multi-agent reinforcement learning framework called QMIX is leveraged to learn a dispatch policy, with which we can obtain the multi-step joint dispatch decisions for the fleet with respect to the delivery requests. Then an efficient multi-transfer matching algorithm is executed to assign the delivery requests to the trucks. Also, DeepFreight is integrated with a Mixed-Integer Linear Programming optimizer for further optimization. The evaluation results show that the proposed system is highly scalable and ensures a 100% delivery success while maintaining low delivery time and fuel consumption.

1 Introduction

According to American Trucking Associations[1], the U.S. trucking industry shipped 11.84 billion tons of goods in 2019 and had a market of 791.7 billion dollars. With shipping costs rising and freight volumes outpacing the supply of available trucks, companies are thinking of radical new initiatives to get their products into customers' hands more efficiently. On the other hand, transportation is currently the largest source (29%) of greenhouse gas emissions in the U.S., where the passenger cars and trucks account for more than 80% of this section[2]. Thus, innovations to enable intelligent and efficient freight transportation are of central importance for the sake of better utilization of the trucks and less fuel consumption.

We propose DeepFreight, a model-free learning framework for the freight delivery problem. The proposed algorithm decomposes the problem into two closely-collaborative components: truck-dispatch and package-matching. In particular, the dispatch policy aims to find the dispatch decisions for the fleet according to the delivery requests. It leverages a multi-agent reinforcement learning framework, QMIX (Rashid et al. 2018), which can train decentralised policies in a centralised end-to-end fashion. Then an efficient matching algorithm is developed to match the delivery requests with the dispatch decisions, which is a greedy approach that makes use of multiple transfers and aims at minimizing the total time use of the fleet. Further, DeepFreight is integrated with a Mixed-Integer Linear Programming (MILP) optimizer for more efficient and reliable dispatch and assignment decisions.

To the best of our knowledge, this is the first work that takes a machine learning aided approach for freight scheduling with multiple transfers. The key contributions of the paper can be summarized as follows: **1)** We propose DeepFreight, a learning-based algorithm for multi-transfer freight delivery, which contains truck-dispatch and package-matching. The dispatch policy is adopted to determine the routes of the trucks, and then matching policy is executed to assign the packages to the trucks efficiently. **2)** We propose to train the dispatch policy through a centralized training with decentralized execution algorithm, QMIX (Rashid et al. 2018), which considers the cooperation among the trucks when training and makes the multi-agent dispatch scalable for execution. **3)** We propose an efficient rule-based matching policy which allows multi-transfer to minimize the usage time of the fleet. **4)** We formulate a MILP model for this freight delivery problem and integrate it with DeepFreight to achieve better reliability and efficiency of the overall system.

2 Related Work

Truck Dispatch Problem. The freight delivery problem is a variant of 'Vehicle Routing Problem (VRP)', which was first introduced in (Dantzig and Ramser 1959) as the 'Truck Dispatch Problem'. Over the decades, the classic VRP has been extended to a lot of variants by introducing additional real-life characteristics (Braekers, Ramaekers, and Van Nieuwenhuyse 2016). For example, VRP with Pickup and Delivery (Parragh, Doerner, and Hartl 2008), that is, goods are required to be picked up and dropped off at certain locations and the pick-up and drop-off must be done with the same vehicle; Multi-Depot VRP (Montoya-Torres et al. 2015), which assumes that there are multiple depots spreading among the customers for scheduling. The

[1] https://www.trucking.org/economics-and-industry-data
[2] https://www.climatecentral.org/gallery/graphics/emissions-sources-2020

extended VRP is NP-hard, and many heuristics and meta-heuristics algorithms have been adopted as the solution, such as simulated annealing (Wang et al. 2015), genetic algorithm (Tasan and Gen 2012), and local search algorithm (Avci and Topaloglu 2015).

We note that these works don't consider that goods can be dropped off in the middle and carried further by another truck (multi-transfer of goods), which has the potential for better utilization of the fleet and is considered in our work. Further, the approaches they adopt are model-based, which need execution every time new observation occurs. Also, the execution time would increase with the scale of the problem, which leads to high time complexity and poor scalability.

Multi-hop Ride-Sharing. As a related area, researches about the ride-sharing system are introduced in this part. To handle the dynamic requests in this scenario, reinforcement learning aided approaches for efficient vehicle-dispatch and passenger-matching have been proposed in (Oda and Joe-Wong 2018; Al-Abbasi, Ghosh, and Aggarwal 2019; de Lima et al. 2020). However, these approaches don't consider passengers going over multiple hops. As shown in (Teubner and Flath 2015), multi-transfer has the potential to greatly increase the ride availability and reduce emissions of the ride-sharing system by making better use of the vehicle capacities. Given this, a reinforcement learning based approach for the multi-hop ride-sharing problem has been recently studied in (Singh, Alabbasi, and Aggarwal 2019).

However, we note that there are some key differences between the ride-sharing system and freight-scheduling system. For ride-sharing, the driving distances are smaller and it's more concerned with the real-time scalable solutions for the highly dynamic requests. While for freight-scheduling, the number of the trucks needed is much fewer due to their huge capacity and the delivery demands can usually be acquired in advance (e.g. one day ahead). In this case, the coordination among trucks is possible and necessary for making better scheduling decisions and we try to take it into consideration with our freight delivery system through a multi-agent reinforcement learning setting.

Multi-Agent Reinforcement Learning. Multi-agent reinforcement learning (MARL) methods hold great potential to solve a variety of real-world problems, and centralized training with decentralized execution (CTDE) is an important paradigm of it. Centralized training exploits the interaction among the agents, while decentralized execution ensures the system's scalability. Specific to the multi-agent cooperative setting (like ours), there are many related works: VDN (Sunehag et al. 2018), QMIX (Rashid et al. 2018), QTRAN (Son et al. 2019) and MAVEN (Mahajan et al. 2019). However, their performance varies greatly in different benchmarks and none of them has the absolute advantage. Considering there have been more applications of QMIX in complicated scenarios and its robust performance (Zhang et al. 2020; Iqbal et al. 2020), we choose QMIX to train the dispatch policy in our algorithm framework.

3 System Model

In this section, we will describe the proposed system model for multi-transfer freight delivery. As an example, consider the scenario shown in Figure 1, where Loads 1 and 4 need to be shipped to a distribution center in Petersburg, VA, while Loads 2 and 3 to the distribution center in New Castle, DE. There are two trucks located at zone A and zone C, with capacity $R_1 = 4$ and $R_2 = 5$, respectively. When freight ride-sharing is enabled, two different possibilities are shown in the figure to serve the requests: 1) serving all the requests using only the truck 2 at zone C (i.e. the dashed red line in the figure), 2) serving Loads 1 and 4 using truck 2 (i.e. the dashed-dotted blue line) and Loads 2 and 3 using truck 1 (i.e. the dotted green line), offering different tradeoffs between delivery time and fuel cost. Furthermore, if multiple transfers are allowed, we may also have the two trucks picking up their loads first and then meeting at zone H to fold the loads into a single truck before the final delivery, which provides additional elasticity for the truck scheduling problem. Through adoption of freight ride-sharing and multi-transfer, fewer truck miles may be required, which will lead to reduced shipping costs and less fuel consumption.

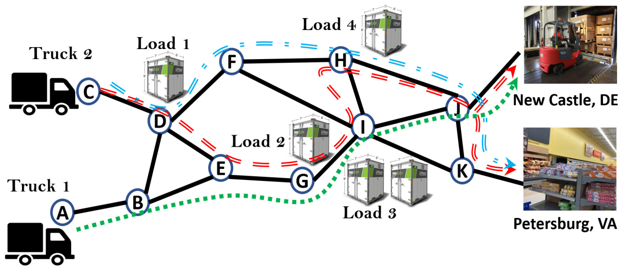


Figure 1: An example for multi-transfer freight delivery

3.1 Model Parameters

In this section, key parameters of the system model are introduced. First, there are N_s distribution centers in the freight delivery system, which can be used as the sources and destinations of packages. Based on them, a delivery request list, denoted by $R_{1:N_r}$ (N_r: the number of requests), is collected at the beginning of each day. The i-th request r_i can be represented as ($source_i$, $destination_i$, $size_i$), which indicates the source, destination, and size of the i-th package.

Second, our target is to complete these requests using a fleet of trucks within a time limit. The number of the trucks for dispatch is N_t and the time limit is denoted as T_{max} which can be one or two days. To fulfill this target, we need to schedule the itinerary of the fleet and the assignment of the packages to the trucks.

Further, in our system, the itinerary of a truck is not decided as a whole and instead consists of a sequence of dispatch decisions, each of which shows the truck's next stop (distribution center) to visit. Also, the assignment of the packages allows multiple transfers, which means that a package can be assigned to more than one truck along the route from its origin to its destination. Note that the total driving time of a truck can't exceed the time limit T_{max} and the volume of the packages it takes at the same time should be within its maximum capacity C_{max}.

3.2 Problem Objectives

The key objectives of the freight delivery system are: (i) maximizing the number of served requests (denoted as N_{rs}) within the time limit T_{max}; and (ii) minimizing the total fuel consumption of the fleet (denoted as F_{total}) during this process. The second component F_{total} is viewed as the function of the total driving time (denoted as T_{drive}) and can be calculated with Equation (1), where $fuel\_factor$ is a constant, representing the fuel consumption per unit time, while eta_j^k is the estimated time of arrival (ETA) for the k-th dispatch decision of truck j that can be obtained through Google Map API.

$$F_{total} = fuel\_factor * T_{drive} \tag{1}$$

$$T_{drive} = \sum_{k=1}^{N_e} \sum_{j=1}^{N_t} eta_j^k \tag{2}$$

4 Proposed Approach: DeepFreight

In this section, we introduce our proposed approach: DeepFreight Algorithm. First, we describe its overall framework in Section 4.1, including the main algorithm modules and connections between them. Then, in Section 4.2-4.4, we talk about these modules further, including the detailed procedure and the intuition to propose them.

4.1 Overall Framework of DeepFreight

Algorithm 1 shows the overall framework of the proposed approach. As mentioned, the proposed approach should give out the itinerary of the fleet and the assignment of the packages to the trucks. In this case, we split the whole system into two parts: the dispatch policy and matching policy. The dispatch policy determines a sequence of dispatch decisions for each truck in the fleet according to the unfinished delivery requests, based on which we can get the route network of the fleet. The matching policy is then executed to assign the requests to the trucks based on the route of each truck. We note that the matching policy is executed based on the dispatch results, while then the matching results provide reward feedback for the training of the dispatch policy. The dispatch policy is trained in a multi-agent reinforcement learning setting called QMIX, which includes: the agent network Q_{single} and mixing network Q_{mix} (Section 4.3).

There are some key points to note about **Algorithm 1**. First, Line 3-6 is the initialization process of the simulator. A new request list will be generated at the beginning of each episode, however, the truck list will not be generated again until the next operation cycle starts. Within an operation cycle, the trucks' locations at the beginning of a new episode are the same as those at the end of the previous episode. Also, we note that the generation of the request list and truck list is model-based, which is introduced in Section 6.1.

Second, Line 7-17 shows the sampling process of the agents which is used to collect the experiences for training. In order to prune the joint search space of the agents and further improve the learning efficiency, which is essential for solving this complex problem, we add some constraints to the truck's routes: 1) It's not allowed for the truck to pass through the same stop twice, except for returning to its origin (no sub-tours); 2) No dispatch decisions are allowed any more after the truck returns to its origin or its cumulative time exceeds the time limit. These constraints are realized through the action mask $m_{1:N_t}$.

Third, not all the dispatch decisions can be matched with delivery requests. For a truck, there may be some idle route segments with no package assignment and these idle dispatch decisions (route segments) can be eliminated on the condition that the truck's route continuity is not affected (show in Line 18). Through this, total fuel consumption is further reduced and better scheduling of the fleet is realized.

Algorithm 1 DeepFreight Algorithm

```
 1: Given: operation cycle $\Delta$, episode time limit $T_{max}$, agent network $Q_{single}$, mixing network $Q_{mix}$
 2: for episode $epi = 1$ to $N_{epi}$ do
 3:     Generate $N_r$ delivery requests randomly
 4:     if $epi$ % $\Delta$ == 1 then
 5:         Generate $N_t$ trucks randomly
 6:     end if
 7:     for epoch $k = 1$ to $N_e$ do
 8:         Obtain the current environment state $s^k$, observation list $z^k_{1:N_t}$, and action mask list $m^k_{1:N_t}$
 9:         Get the q-value list $Q^k_{1:N_t}$ from $Q_{single}$
10:         Choose the available action $u^k_{1:N_t}$ for each truck
            based on the corresponding $Q^k_{1:N_t}$ and $m^k_{1:N_t}$,
            with Boltzmann exploration (Cesa-Bianchi et al. 2017)
11:         Match the delivery requests with the dispatch decisions with Algorithm 2
12:         Calculate the joint reward $r^k$ according to Equation (3)
13:         Update the cumulative time for each truck $t^k_{1:N_t}$
14:         if $min(t^k_{1:N_t}) \geq T_{max}$ or no requests left then
15:             break
16:         end if
17:     end for
18:     Eliminate the useless dispatch decisions, calculate the number of unfinished packages and average driving time of the fleet for evaluation
19:     Update replay buffer $B$ with experiences collected above, including $(s^{1:N_e}, z^{1:N_e}_{1:N_t}, u^{1:N_e}_{1:N_t}, r^{1:N_e}, m^{1:N_e}_{1:N_t})$
20:     for iteration $iter = 1$ to $N_{iter}$ do
21:         Sample a random batch of experiences from $B$
22:         Update $Q_{single}$ and $Q_{mix}$ by minimizing loss function defined as Equation (4)
23:     end for
24: end for
```

4.2 Model Dispatch Problem with Dec-POMDP

This cooperative multi-agent dispatch problem can be modeled with Dec-POMDP (Decentralized Partial Observable Markov Decision Process), and described as a tuple $(S, U, P, r, Z, O, n, \gamma)$. General definition of these elements can be found in Section IV-B of the extended version of our paper (Chen et al. 2021).

Specifically, the environment state space S for this package delivery dispatch problem includes four terms: 1) current epoch number k; 2) distribution of the trucks at epoch k, that is the number of the trucks at each stop; 3) avail-

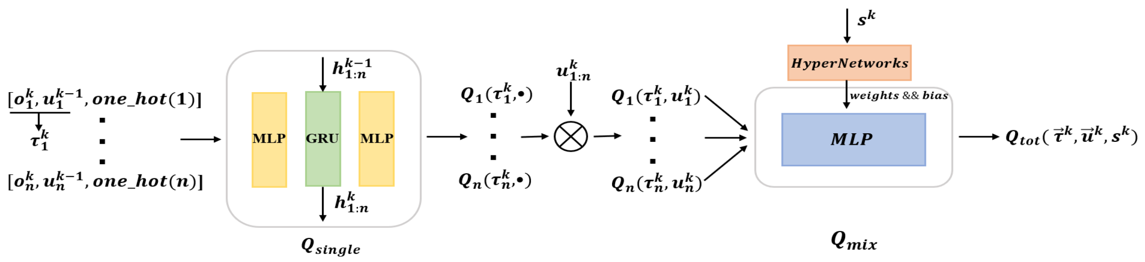


Figure 2: Workflow of QMIX Algorithm

able capacity of each truck; 4) unfinished delivery requests at epoch k, which are initialized at the beginning of each episode and updated with the epochs. As for the individual observation space Z, it is for a single truck, so it replaces the item 2) 3) with the truck's current stop and available capacity respectively and keeps item 1) 4).

Moreover, the individual action space U includes: 1) $next\_stop \in \{1, 2, \cdots, N_s\}$: the index of the next stop; 2) invalid dispatch: there are some constraints for the dispatch decisions to be valid, e.g. within the time limit. In this case, the dimension of the individual action space U is $N_s + 1$.

The joint reward function r should reflect the objectives mentioned in Section 3.2, so the reward for the epoch k is defined as Equation (3), where $\beta_{1:2} > 0$.

$$r_k = \beta_1 N_{rs}^k - \beta_2 F_{total}^k \tag{3}$$

Obviously, it has a positive correlation with the package number served at epoch k and a negative correlation with the fuel consumption during this process.

Based on these elements, the dispatch policy for agent a can be defined as $\pi^a(u^a|z^a) : Z \times U \rightarrow [0, 1]$. The execution of the policy is decentralized, since it conditions only on the observation of the single agent.

4.3 QMIX for Training the Dispatch Policy

QMIX (Rashid et al. 2018) is a value-based reinforcement learning algorithm that can train decentralized policies in a centralized end-to-end fashion. It is suitable to solve this multi-agent freight delivery problem because 1) the centralized training process takes the coordination among trucks into consideration, 2) the decentralized execution process alleviates the complexity to determine the joint dispatch decisions and ensures the scalability of the dispatch policy. The workflow of QMIX is shown as Figure 2, where two key components: Q_{single} and Q_{mix}, need special attention.

The agent network Q_{single} outputs the Q-value list for each truck, which can be used for decentralized execution, or as a part of the centralized training process. The inputs of Q_{single} include: 1) the agent (truck) id a, which is in the form of a one-hot vector; 2) the individual observation for the current epoch o_a^k; 3) the dispatch decision (action) for the previous epoch u_a^{k-1}. Input 2) shows that the execution of the dispatch policy is in a decentralized way, which conditions only on local observations. While, input 3) shows that dispatch decisions of different epochs within an episode are correlated rather than independent.

The mixing network Q_{mix} estimates the joint Q-value Q_{tot} as a complex non-linear combination of Q-values for all the n trucks. *HyperNetworks* (Ha, Dai, and Le 2017) is adopted to transform the environment state s^k into the parameters of Q_{mix}. Based on Q_{tot}, the loss function for QMIX can be defined as Equation (4), where y_{tot}^k (Equation (5)) is the target joint q-value from the target networks.

$$Loss(\boldsymbol{\theta}) = \sum_{i=1}^{batchsize} \left[(y_{tot}^k - Q_{tot}(\boldsymbol{\tau}^k, \mathbf{u}^k, s^k; \boldsymbol{\theta}))_i^2 \right] \tag{4}$$

$$y_{tot}^k = r^k + max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}^{k+1}, \mathbf{u}, s^{k+1}; \boldsymbol{\theta}^-) \tag{5}$$

After the centralized training, the agent network Q_{single} can be used to give dispatch decisions for each truck based on their individual observation in a decentralized way.

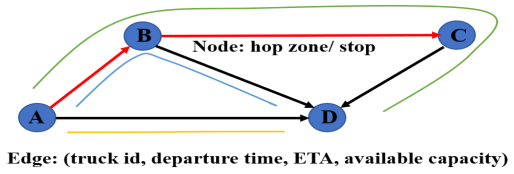


Figure 3: Graph Representation of the Dispatch Decisions

4.4 Matching Algorithm

The matching algorithm, shown as **Algorithm 2**, is executed to match the delivery requests with the dispatch decisions. Considering the number of the packages to deliver is huge, it's nearly impossible to find the optimal matching results for all the packages within a reasonable time. In this case, **Algorithm 2** adopts a greedy approach to ensure the optimality for every single package (assuming no other packages) and takes the least increase in the total driving time of the fleet as the optimization objective. It mainly includes two parts:

First, in Line 3-14, a weighted directed graph, like Figure 3, is created based on the dispatch decisions. Each node in the graph represents for a stop that has occurred in the dispatch decisions. While each edge means there is a truck that has been dispatched between the two stops. Also, the id, departure time, and available capacity of the truck and ETA between the two stops are saved with the edge for later use.

Second, based on the graph, we can adopt DFS to acquire all the available paths from the source to the destination for each delivery request (Line 16). An available path contains a list of edges and their truck id may be different, which means the package may be delivered by more than one truck (multi-transfer). In this case, there are some conditions for a path to be available: 1) The package's arrival time at any node in the path should be earlier than the departure time from that node of the truck it transfers to; 2) The package's size should be smaller than that truck's available capacity.

Further, among all the available paths, the one that causes the least increase in the total driving time of the fleet will be chosen (Line 18). For example, there are three available paths from A to D in Figure 3: $A \rightarrow D$, $A \rightarrow B \rightarrow D$, and $A \rightarrow B \rightarrow C \rightarrow D$ and the edges $A \rightarrow B$ and $B \rightarrow C$ have been matched with other requests. Then, the time cost of the third path should only be the ETA from C to D, since it will share the trucks denoted by the red edges with other requests (freight ride-sharing). Similarly, the time cost of the first two paths are respectively the ETA from A to D and ETA from B to D. Then, in order to minimize total driving time of the fleet, the path with the least time cost should be chosen. Note that if there are more than one path with the least time cost, the shortest path will be chosen.

Algorithm 2 Matching Algorithm

```
1: Given: initial capacity of the trucks, dispatch decisions viz.
   $next\_stop$ for each truck at each valid epoch, request list viz.
   $(source, destination, size)_{1:N_r}$
2: Turn the dispatch decisions into a weighted directed graph as
   follows (Line 3-14):
3: Create as many nodes as stops in the dispatch decisions, using
   the stop id as the node id
4: Set the $current\_time$ for each truck as 0
5: for every truck do
6:     for every valid epoch do
7:         Create a directed edge from $current\_stop$ node
           to $next\_stop$ node
8:         Set the departure time from $current\_stop$ as the
           truck's $current\_time$
9:         Acquire the ETA from $current\_stop$ node to
           $next\_stop$ node through Google Map API
10:        Record the id, departure time, ETA, and initial
           capacity of the truck as the edge's information
11:        Update the truck's $current\_stop$ as $next\_stop$
12:        Update the truck's $current\_time$ as ETA +
           $current\_time$
13:    end for
14: end for
15: for every delivery request do
16:    Find all the available paths from $source$ to
       $destination$ using DFS (depth-first search)
17:    if success then
18:        Choose the path that causes the least increase in
           the total driving time of the fleet
19:        Subtract the package size from the available capacity
           of the edges it passes through
20:    end if
21: end for
```

5 Integration with MILP

We further propose a hybrid approach that harnesses DeepFreight and MILP to ensure successful delivery of all packages. Specifically, experiments show that when most of the requests have been completed and only a small number of packages remain to be optimized, DeepFreight may have unstable training dynamics resulting in undelivered packages (as evidenced in Figure 5(e)). To this end, we leverage MILP to find the exact routing decisions for the small portion of requests that are not efficiently handled by DeepFreight. This leads to an integration of DeepFreight and MILP, denoted as DeepFreight+MILP. The MILP's formulation and the workflow of DeepFreight+MILP are presented in this section.

5.1 MILP Formulation

Parameters:

- $i, j, l \in L = \{0, 1, \ldots, m-1\}$: index and set of locations;
- $k \in N = \{0, 1, \ldots, n-1\}$: index and set of trucks;
- D: depot for each truck k, a dictionary, $D(k) \in L$;
- $t_{i,j} \in \mathbb{N}$: ETA from location i to location j, in seconds;
- $d_{i,j} \in \mathbb{N}$: delivery demand from location i to location j;
- $C \in \mathbb{N}^+$: truck capacity, the same for each truck k;
- $T_k \in \mathbb{N}^+$: the time limit for truck k.

Decision Variables:

- $x_{i,j,k} \in \{0, 1\}$: the variable equals 1, if truck k travels from location i to location j, and 0 otherwise;
- $u_k \in \{0, 1\}$: the variable equals 1, if truck k is in use, and 0 otherwise;
- $r_{i,j,k} \in \{0, 1\}$: the variable equals 1, if truck k takes the delivery demand from location i to location j, and 0 otherwise; (Note that a truck takes the path from i to j doesn't mean it will take the delivery request from i to j.)
- $s_{i,k} \in \mathbb{N}$: the cumulative stop number when truck k arrives at location i;
- $v_{i,k} \in \mathbb{N}$: the cumulative volume when truck k departs from location i.

Optimization Objective:

The optimal solution should minimize the objective function defined as follows, where T_1 and T_2 respectively represent the total time use (in hours) and the total volume of the unserved packages, and $\omega_1 > 0, \omega_2 > 0$ are the weights for each term. Based on the results of parameter-adjustment, ω_1 is set as 0.5 and ω_2 is set as 0.04.

$$target = \omega_1 T_1 + \omega_2 T_2 \tag{6}$$

$$T_1 = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \left[t_{i,j} \times \left(\sum_{k=0}^{n-1} x_{i,j,k} \right) \right] \tag{7}$$

$$T_2 = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \left[d_{i,j} \times \prod_{k=0}^{n-1} (1 - r_{i,j,k}) \right] \tag{8}$$

Constraints:

- No self-loop:

$$\forall i \in L, \forall k \in N, x_{i,i,k} == r_{i,i,k} == 0 \tag{9}$$

- Within the time limit:

$$\forall k \in N, \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} t_{i,j} x_{i,j,k} \leq u_k T_k \tag{10}$$

- Depart from and return to the same depot:

$$\forall k \in N, \sum_{i=0}^{m-1} x_{i,D(k),k} == \sum_{i=0}^{m-1} x_{D(k),i,k} == u_k \quad (11)$$

- Form a tour that includes the depot:

$$\forall k \in N, \forall i \in L, \sum_{j=0}^{m-1} x_{j,i,k} == \sum_{j=0}^{m-1} x_{i,j,k} \leq u_k \quad (12)$$

- Eliminate any subtour:

$$\forall k \in N, s_{D(k),k} == 0 \quad (13)$$

$$\begin{aligned} &\forall k \in N, \forall i \in L \backslash \{D(k)\}, \forall j \in L, \\ &s_{i,k} \geq 1 + s_{j,k} - m(1 - x_{j,i,k}) \end{aligned} \quad (14)$$

$$\forall k \in N, \forall i \in L, s_{i,k} \geq 0 \quad (15)$$

- For each request, one truck can be assigned at most:

$$\forall i \in L, \forall j \in L, \sum_{k=0}^{n-1} r_{i,j,k} \leq 1 \quad (16)$$

- If truck k takes the demand from location i to location j, truck k's trajectory should include $i \rightarrow j$:

$$\begin{aligned} &\forall i \in L, \forall k \in N, \forall j \in L \backslash \{D(k)\}, \\ &r_{i,j,k} \leq \left(\sum_{l=0}^{m-1} x_{i,l,k} \right) \left(\sum_{l=0}^{m-1} x_{l,j,k} \right), \\ &r_{i,j,k}(s_{j,k} - s_{i,k}) \geq 0 \end{aligned} \quad (17)$$

$$r_{i,D(k),k} \leq \left(\sum_{l=0}^{m-1} x_{i,l,k} \right) \left(\sum_{l=0}^{m-1} x_{l,D(k),k} \right) \quad (18)$$

- Within the capacity limit:
$\forall k \in N, \forall i \in L \backslash \{D(k)\}$,

$$v_{i,k} == \sum_{j=0}^{m-1} v_{j,k} x_{j,i,k} + \sum_{l=0}^{m-1} r_{i,l,k} d_{i,l} - \sum_{l=0}^{m-1} r_{l,i,k} d_{l,i} \quad (19)$$

$$\forall k \in N, v_{D(k),k} == \sum_{i=0}^{m-1} r_{D(k),i,k} d_{D(k),i} \quad (20)$$

$$\forall i \in L, \forall k \in N, 0 \leq v_{i,k} \leq u_k C \quad (21)$$

Given the large number of packages, we efficiently combine MILP with DeepFreight for a scalable solution.

5.2 The Workflow of DeepFreight+MILP

To exploit the advantages of DeepFreight and MILP, we propose an integration of the two, that is DeepFreight gives out dispatch and assignment decisions for most of the requests, while MILP is adopted to find the exact truck routing for the remaining unmatched requests. The workflow is as follows:

1. Get the initial dispatch decisions and matching results using **Algorithm 1**;
2. Define a key parameter called $efficiency$, which equals the number of packages delivered by the truck divided by its driving time, and calculate $efficiency$ for each truck;
3. Eliminate all the dispatch decisions of the trucks whose $efficiency$ is lower than a certain threshold;
4. Rematch the package list with the pruned dispatch decisions, and get the **unmatched package list**;
5. Pick the **new truck list**: choose two trucks from the initial truck list for each distribution center that has unmatched packages, based on the trucks' $priority$ defined as Equation (22); ($available\_time$ means the truck's available time for dispatch before the time limit and eta means the ETA from the truck to the distribution center, so the truck with a higher $priority$ is preferred.)

$$priority = available\_time - 2 * eta \quad (22)$$

6. Adopt the MILP optimizer to get the routing result for the **new truck list** to serve the **unmatched package list**.

The threshold of $efficiency$ should be determined by the total number of the packages and the total driving time of the fleet that we'd like to achieve. By eliminating the inefficient dispatch decisions, better utilization of the fleet can be realized. However, note that a higher threshold means a larger **unmatched package list** and the MILP optimizer may not be able to find the optimal routing results when the package number is too large (e.g. shown as Figure 4(e)), so there should be a tradeoff. After fine-tuning, we set the threshold as 0.028 ($\approx 40000/(20*20*3600)$) in our experiment, that is how much the $efficiency$ should be if 20 trucks complete 40000 requests with an average driving time of 20 hours.

6 Evaluation and Results

In this section, we introduce the setup of the simulator and comparisons among: DeepFreight, DeepFreight without Multi-transfer, MILP and DeepFreight+MILP.

6.1 Simulation Setup

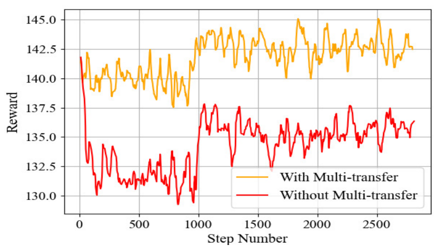
Ten Amazon distribution centers in the eastern U.S. are chosen as the origins and destinations of delivery requests in the simulation (shown as Figure 4 in the extended version of our paper (Chen et al. 2021)). For each episode, N_t trucks should complete N_r randomly generated requests within the time limit T_{max}. The generation of the request and truck list is model-based:

As mentioned above, each delivery request can be represented as $(source, destination, size)$. Equation (23) shows the distribution of the requests' sources, where pop_i represents the population in the vicinity of distribution center i which is acquired through its zip code [3]. After confirming the source of a delivery request, its destination is randomly generated according to the distribution defined as Equation (24), where $norm$ is the normalization coefficient. Further, the size of each package is represented as an integer uniformly distributed between 1 and 30 in our simulation.

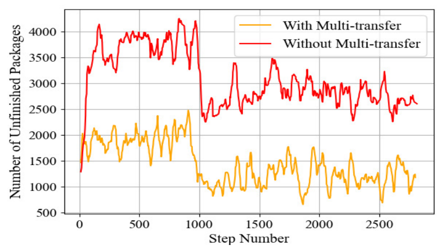
$$P(source = i) = \frac{pop_i}{\sum_{k=1}^{10} pop_k} \quad (23)$$

$$P(destination = j|i) = norm * \frac{pop_j}{\sqrt{eta_{i,j}}} \quad (24)$$

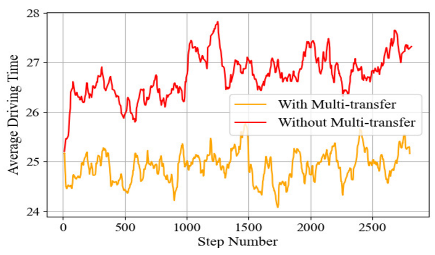
[3] https://www.cdxtech.com/tools/demographicdata/



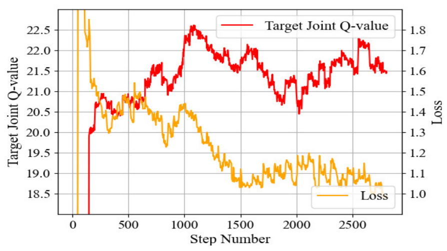
(a) Reward function



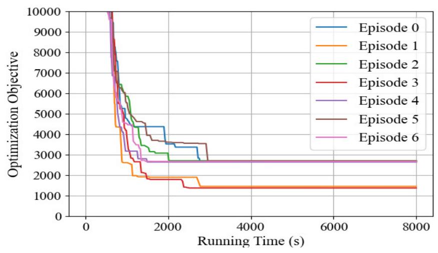
(b) Number of unfinished packages



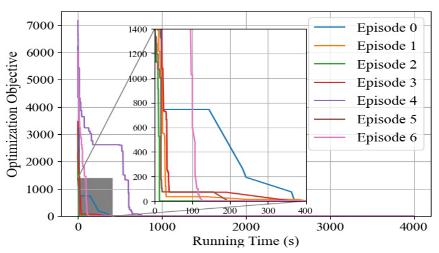
(c) Average driving time of the fleet



(d) Target joint q-value and loss function



(e) Optimization curve of MILP



(f) Optimization curve of DeepFreight+MILP

Figure 4: (a)-(c): Training curves of DeepFreight with/without Multi-transfer. DeepFreight with Multi-transfer can complete more package requests with a smaller driving time. (d): Changing curves of loss function (Equations (4)) and target joint q-value (Equation (5)) for DeepFreight, which show its learning process. (e)-(f): Convergence curves of MILP and DeepFreight+MILP over time. The optimization objective is defined in Equation (6)-(8). It can be observed that MILP falls into a local optimum with poor performance, while DeepFreight+MILP can obtain a significantly improved solution within 10 minutes.

As for the truck list, we need to specify its location at the beginning of each operation cycle. In our simulation, an operation cycle includes 7 episodes and the time limit T_{max} for each episode is 2 days. Within an operation cycle, the trucks' locations at the beginning of a new episode are the same as those at the end of the previous episode, while the distribution of the trucks' locations at the first episode is the same as that of the packages' sources (Equation (23)), which is convenient for the fleet to pick up the packages.

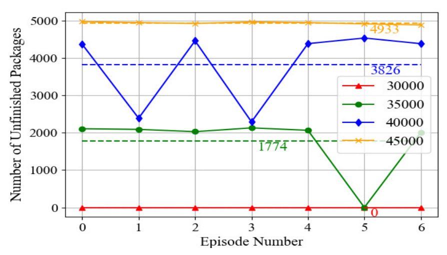
Moreover, we set the truck number N_t as 20, and the maximum capacity for each truck as 30000. Also, with the consideration that trucks should be efficiently used, the total volume of the the delivery requests should match the load capacity of the fleet, so the request number N_r is set as 40000.

6.2 Discussions and Results

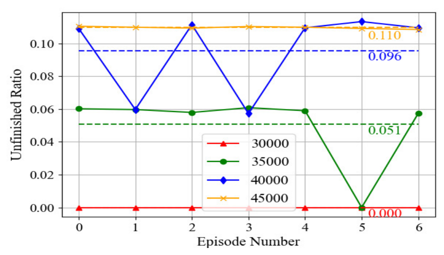
Training Process of DeepFreight: The training process includes 2800 episodes, that is 400 operation cycles, and at each episode, the networks will be trained for 100 iterations. From the loss curve in Figure 4(d), it can be observed that the training process starts to converge at the 1500th episode. Further, the training effect can be seen from the increase of the reward shown in Figure 4(a). The reward function for evaluation is the same as that for training (Equation (3)), which is related to the average driving time of the fleet (0-48 hours) and the number of packages served by them (0-40000). β_1 and β_2 are key parameters for defining the reward function. Based on parameter-adjustment, we set them as: $\beta_1 = 0.004, \beta_2 = 0.5$. The increase of the reward is mainly due to the reduction of the number of unfinished packages, which decreases from ˜2000 to ˜1200 and the lowest number shown in Figure 4(b) is about 700. Also, we pick the optimal checkpoint between the 1500th and 2000th episode for the further experiments, as a comparison with other approaches. The comparison is about their performance within an operation cycle which equals two weeks (defined in Section 6.1).

DeepFreight vs. DeepFreight without Multi-transfer: To show the improvement brought by the flexibility of multiple transfers, we compare our approach with a freight delivery system without multi-transfer, of which the results are shown in Figure 4(a)-4(c). For the system without multi-transfer, each package will be delivered to its destination by a fixed truck, so there is an extra restriction on the truck used when executing the matching policy (**Algorithm 2**). It can be observed that after convergence, with the flexibility of multi-transfer, the average driving time is shorter (˜2 hours (7.5%) reduction) and the number of unfinished packages is lower (˜1500 packages (60%) reduction), which means better utilization of the fleet is realized. For 40000 packages, 53.3% are delivered with no transfer in the middle, 41.2%: one hop, 5.5%: two or more hops, so the extra load/unload effort due to multi-transfer can be overlooked.

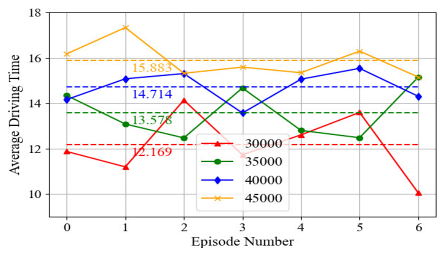
DeepFreight vs. MILP: MILP is used as the baseline algorithm, which is defined in Section 5.1 and solved through a state-of-the-art optimization solver: Gurobi (Gurobi Optimization 2020). The evaluation of MILP shows that it has poor scalability. The fleet can carry up to 40000 packages at the same time. Figure 5(a)-5(c) show that when the number of packages is 30000, the MILP solver can complete all the requests at a fairly low average driving time (12.169 hours). However, its performance drops dramatically with growing number of packages: both the number and the ratio of the



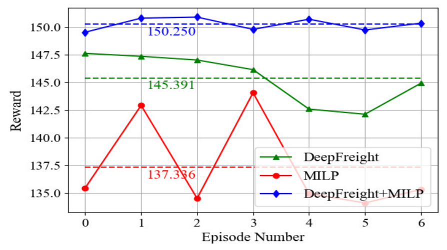
(a) Number of unfinished packages



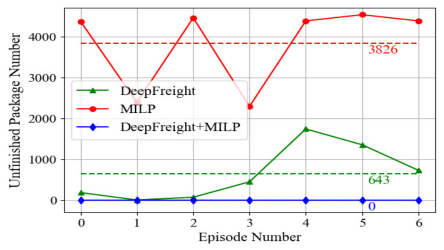
(b) Ratio of the unfinished packages



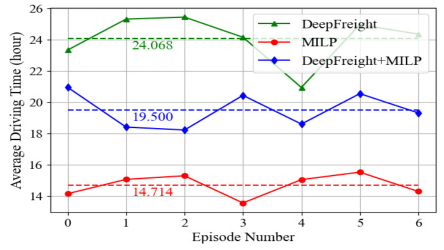
(c) Average driving time of the fleet



(d) Comparison about reward function



(e) Number of unfinished packages



(f) Average driving time of the fleet

Figure 5: (a)-(c): Plotting MILP's performance for varying number of requests. It can be observed that not only the driving time and the number of unfinished packages increase, the ratio of unfinished packages (normalized by the total package number) also grows, demonstrating the poor scalability of MILP. (d)-(f): Comparisons of different algorithms, in terms of the reward function, the number of unfinished packages and the fleet's average driving time. As a comprehensive index, the reward function we use for evaluation is the same as that for training (Equation (3)), which gives priority to the rate of delivery success. DeepFreight+MILP performs best among these algorithms. It achieves 100% delivery success with low fuel consumption.

unfinished packages increase. Further, Figure 4(e) shows the convergence curves of the MILP solver, when the number of delivery requests is 40000. It can be observed that the optimization objective (defined as Equation (6)-(8)) converges within 3000s and then the performance doesn't increase any more, which means the MILP solver cannot find a solution for this task even if given more time. Note that the MILP solver runs on a device with an Intel i7-10850H processor.

Next, we compare Deepfreight and MILP. First, in terms of time complexity, Q_{single} trained with DeepFreight can be executed in a decentralized manner. When testing, it can give out the multi-step dispatch decisions for each truck in real time. While, when using MILP, we have to run the optimizer for every episode, since the problem scenario has changed. Also, the time spent will increase with the number of the trucks, distribution centers and packages. Second, as for their performance, Figure 5(e) shows that DeepFreight can complete more requests than MILP. However, its performance is unstable, which is one of the characteristics of reinforcement learning. Also, the average driving time taken of DeepFreight is much higher than that of MILP.

Overall, as compared to MILP, DeepFreight is transferable among different problem scenarios and has lower time complexity, but the average driving time and number of unfinished packages need to be further reduced, which motivates our design of DeepFreight+MILP.

DeepFreight vs. DeepFreight+MILP: Among all the evaluation metrics, serving all the packages is given top priority. As shown in Figure 5(e), DeepFreight+MILP completes all the delivery requests of an operation cycle, eliminating the uncertainty and instability of DeepFreight, which makes it suitable for the industrial use. Figure 5(f) shows that DeepFreight+MILP has a further reduction in the average driving time as compared with DeepFreight and with comparable driving time as MILP, DeepFreight+MILP realizes a 100% delivery success. Figure 4(f) shows the convergence curves of DeepFreight+MILP when using the optimizer, and it can be observed that we can get the routing results for the rest of the packages within 10 minutes. That is because most of the packages have been served by the dispatch decisions made by DeepFreight and only 4000-5000 packages are left for MILP to serve. Also, not all the trucks are used for dispatch (as the **new truck list** defined in Section 5.2). In this case, DeepFreight+MILP can still be adopted when the number of packages or trucks is large, which ensures its scalability.

Overall, DeepFreight+MILP performs best among these algorithms, because it not only has better scalability and lower time complexity but also can ensure a 100% delivery success with fairly low fuel consumption.

7 Conclusion

This paper proposes DeepFreight, a novel learning-based approach for multi-transfer freight delivery. The problem is sub-divided into truck-dispatch and package-matching, where QMIX is used for training the dispatch policy and DFS is used for matching in a greedy fashion. This approach is then integrated with MILP for further optimization. Evaluation results show superior scalability and improved performance of the combined system as compared to the MILP solver alone and the learning-based solution alone.

References

Al-Abbasi, A. O.; Ghosh, A.; and Aggarwal, V. 2019. Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems* 20(12): 4714–4727.

Avci, M.; and Topaloglu, S. 2015. An adaptive local search algorithm for vehicle routing problem with simultaneous and mixed pickups and deliveries. *Computers & Industrial Engineering* 83: 15–29.

Braekers, K.; Ramaekers, K.; and Van Nieuwenhuyse, I. 2016. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering* 99: 300–313.

Cesa-Bianchi, N.; Gentile, C.; Lugosi, G.; and Neu, G. 2017. Boltzmann exploration done right. *Advances in neural information processing systems* 30: 6284–6293.

Chen, J.; Umrawal, A. K.; Lan, T.; and Aggarwal, V. 2021. DeepFreight: A Model-free Deep-reinforcement-learning-based Algorithm for Multi-transfer Freight Delivery. URL https://arxiv.org/abs/2103.03450.

Dantzig, G. B.; and Ramser, J. H. 1959. The truck dispatching problem. *Management science* 6(1): 80–91.

de Lima, O.; Shah, H.; Chu, T.-S.; and Fogelson, B. 2020. Efficient Ridesharing Dispatch Using Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2006.10897* .

Gurobi Optimization, I. 2020. Gurobi optimizer reference manual.

Ha, D.; Dai, A. M.; and Le, Q. V. 2017. HyperNetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL https://openreview.net/forum?id=rkpACe1lx.

Iqbal, S.; de Witt, C. A. S.; Peng, B.; Böhmer, W.; Whiteson, S.; and Sha, F. 2020. AI-QMIX: Attention and Imagination for Dynamic Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2006.04222* .

Mahajan, A.; Rashid, T.; Samvelyan, M.; and Whiteson, S. 2019. Maven: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems*, 7613–7624.

Montoya-Torres, J. R.; Franco, J. L.; Isaza, S. N.; Jiménez, H. F.; and Herazo-Padilla, N. 2015. A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering* 79: 115–129.

Oda, T.; and Joe-Wong, C. 2018. MOVI: A model-free approach to dynamic fleet management. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, 2708–2716. IEEE.

Parragh, S. N.; Doerner, K. F.; and Hartl, R. F. 2008. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft* 58(2): 81–117.

Rashid, T.; Samvelyan, M.; De Witt, C. S.; Farquhar, G.; Foerster, J.; and Whiteson, S. 2018. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485* .

Singh, A.; Alabbasi, A.; and Aggarwal, V. 2019. A distributed model-free algorithm for multi-hop ride-sharing using deep reinforcement learning. *arXiv preprint arXiv:1910.14002* .

Son, K.; Kim, D.; Kang, W. J.; Hostallero, D. E.; and Yi, Y. 2019. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1905.05408* .

Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M.; Zambaldi, V. F.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J. Z.; Tuyls, K.; et al. 2018. Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *AAMAS*, 2085–2087.

Tasan, A. S.; and Gen, M. 2012. A genetic algorithm based approach to vehicle routing problem with simultaneous pick-up and deliveries. *Computers & Industrial Engineering* 62(3): 755–761.

Teubner, T.; and Flath, C. M. 2015. The economics of multihop ride sharing. *Business & Information Systems Engineering* 57(5): 311–324.

Wang, C.; Mu, D.; Zhao, F.; and Sutherland, J. W. 2015. A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup–delivery and time windows. *Computers & Industrial Engineering* 83: 111–122.

Zhang, C.; Odonkor, P.; Zheng, S.; Khorasgani, H.; Serita, S.; and Gupta, C. 2020. Dynamic Dispatching for Large-Scale Heterogeneous Fleet via Multi-agent Deep Reinforcement Learning. *arXiv preprint arXiv:2008.10713* .