THE TRANSFORMER NETWORK FOR THE DIAL-A-RIDE PROBLEM

Aziz Laadhar[*], **Aoyu Gong**[*], **Benedek Harsanyi**[*]
École Polytechnique Fédérale de Lausanne (EPFL)
{aziz.laadhar, aoyu.gong, benedek.harsanyi}@epfl.ch

ABSTRACT

In this paper, we introduce a Transformer-based learning framework for the Dial-a-Ride Problem (DARP), a pickup and delivery transportation problem that has traditionally been solved using mixed integer programming. The Transformer, a neural network architecture originally developed for natural language processing, is adapted to the DARP and trained through supervised learning and reinforcement learning. Decoding is done using greedy sampling and beam search. We report the performance gap compared to, a recent Gurobi-based solver. Our results reinforce the recent trend, that deep learning techniques may be able to learn better heuristics for combinatorial problems than human-engineered approaches and that the Transformer architecture may be a promising tool for tackling NP-hard problems in the industry.

***K*eywords** dial-a-ride problem, transformer, transportation, vehicle routing, reinforcement learning

1 Introduction

The Dial-a-Ride problem (DARP) is a transportation optimization problem that involves scheduling pick-ups and drop-offs for a fleet of vehicles serving a particular geographical region. The goal is to minimize the total travel time or distance for the fleet while also meeting the demand of the riders. The DARP has many real-world applications, including public transit systems, ride-sharing services, and logistics companies. It is a challenging problem due to the complexity of the constraints involved, such as capacity limits, time windows, and distance constraints. As a result, finding efficient algorithms to solve the DARP is an active area of research in the field of transportation and logistics. The DARP is closely related to well-known optimization problems since it can be viewed as a generalization of the Traveling Salesman Problem (TSP), in which the objective is to find the shortest possible route that visits each city exactly once and returns to the origin city. The DARP can also be seen as a variant of the Vehicle-Routing-Problem (VRP), in which the objective is to find the optimal routes for a fleet of vehicles serving a set of locations. The VRP assumes that each vehicle has an infinite capacity and can visit any location at any time, while in DARP, there are capacity constraints and time windows for each vehicle and each location.

The remainder of the paper is organized as follows. In Section 2, a literature review is presented, discussing previous work on the DARP. In Section 3, the problem statement is presented, outlining the specific objectives and constraints of the problem as a mixed integer programming task. In Section 4, we describe how the DARP can be modeled as a Markov decision process (MDP). In Section 5, a deep learning framework, for solving the problem is presented, using a novel variant of the Transformer Encoder architecture. Section 6 presents numerical experiments evaluating the proposed approach using both supervised and reinforcement learning. Finally, in the seventh section, the main conclusions of the paper are drawn.

2 Literature Review

The majority of articles in this field focus on static and deterministic DARP formulations, which means that the problem is not changing over time and the data used to model the problem is known with certainty. [1] successfully applied the branch-and-cut (B&C) algorithm to DARP, where cutting planes are added to the problems in the B&B

[*]Equal contribution. Listing order is random.

tree. The addition of cuts tightens the LP-relaxations in the B&B tree, leading to a higher chance of finding integer solutions. In 2015 [2], a B&C-based solution was proposed, where ordinary and dynamic time windows are handled in a column-generation subproblem. Since, DARP is an NP-hard, problem, exact solvers are focused on small-sized instances, and many heuristic-based solutions were developed, to overcome this issue. The large neighborhood search (LNS) [3] approach involves repeatedly modifying the current solution by removing a portion of it and then using the remaining elements to create a new complete solution. Simulated annealing (SA) is a metaheuristic that uses a stochastic local search approach, where a new solution is generated by selecting a neighbor of the current solution at each iteration [4]. In 2021, a new formulation was proposed [5], the paper introduced the notion of "Restricted fragments", which are segments of routes that can represent any dial-a-ride problem (DARP) route. It is possible to enumerate these restricted fragments and prove results on domination between them. The proposed formulation can be solved using a branch-and-cut (B&C) algorithm, which includes new valid inequalities specifically designed for the restricted fragment formulation.

In recent years, deep learning techniques were successfully applied to NP-hard combinatorial problems. In [6], the Transformer architecture, developed for natural language processing (NLP), was proposed as a policy network, to construct routes for the Traveling Salesman Problem (TSP) in an autoregressive manner.

3 Problem Statement

The DARP involves designing vehicle routes and schedules for N users, indexed by $n \in \mathcal{N} = \{1, 2, \ldots, N\}$, who specify requests for transportation from pick-up sources to drop-off destinations. The transportation is supplied by a group of K identical vehicles, indexed by $k \in \mathcal{K} = \{1, 2, \ldots, K\}$, departing from the same source station. The objective is to design a set of K minimum-cost vehicle routes serving N users under a number of constraints.

Let T and Q be the maximum route duration and maximum vehicle capacity of a vehicle. Let L be the maximum ride time of a user. Each user $n \in \mathcal{N}$ is associated with a pick-up location and a drop-off location, denoted by p_n and d_n respectively. The set of all locations is denoted by $\mathcal{V} = \mathcal{P} \cup \mathcal{D} \cup \{s^+, s^-\}$, where $\mathcal{P}$ and $\mathcal{D}$ are the sets of pick-up and drop-off locations, and s^+ and s^- are the source and destination stations. Let (x_i, y_i) be the coordinates of location i. Each location $i \in \mathcal{V}$ corresponds to a service duration δ_i and a load q_i satisfying $\delta_{s^+} = \delta_{s^-} = 0$, $\delta_{p_n} = \delta_{d_n} > 0$ for each $n \in \mathcal{N}$, $q_{s^+} = q_{s^-} = 0$, and $q_{p_n} = -q_{d_n} > 0$ for each $n \in \mathcal{N}$. Moreover, each location $i \in \mathcal{V}$ is associated with a time window $[e_i, l_i]$, where e_i and l_i are the earliest and latest time at which service may start at location i. Hence, a user has two time windows, one for its pick-up location and the other for its drop-off location. The user will specify either of them. If user n specifies $[e_{p_n}, l_{p_n}]$, it will be called an inbound user. Otherwise, it will be called an outbound user. Note that, although a user only specifies one time window, the other time window can be tightened [1].

By considering a location as a vertex, the DARP can be defined on a complete digraph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the same as before and $\mathcal{E} = \big\{\{i, j\} \mid i, j \in \mathcal{V}\big\}$ is the set of edges. Each edge $\{i, j\} \in \mathcal{E}$ is associated with a routing cost $c_{i,j}$ and a travel time $t_{i,j}$ computed based on the Euclidean distance, i.e.,

$$\begin{aligned} c_{i,j} &= \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \\ t_{i,j} &= \delta_i + c_{i,j}. \end{aligned}$$

The travel time includes the service duration at location i and the time taken from location i to location j [5]. For each user $n \in \mathcal{N}$, each vehicle $k \in \mathcal{K}$, each location $i \in \mathcal{V}$, and/or each edge $\{i, j\} \in \mathcal{E}$, we define four decision variables.

- Let $\phi^k_{i,j} = 1$ if vehicle k traverses edge $\{i, j\}$, and $\phi^k_{i,j} = 0$ otherwise.
- Let B^k_i be the time at which vehicle k starts a service at location i.
- Let Q^k_i be the load of vehicle k when leaving location i.
- Let L^k_n be the ride time of user n on vehicle k.

As mentioned before, we first tighten time windows. For an inbound user n specifying $[e_{p_n}, l_{p_n}]$, we have

$$\begin{aligned} e_{d_n} &= \max(0, e_{p_n} + \delta_{p_n} + t_{p_n,d_n}), \\ l_{d_n} &= \min(l_{p_n} + \delta_{p_n} + L, T). \end{aligned}$$

For an outbound user n' specifying $[e_{d_{n'}}, l_{d_{n'}}]$, we have

$$\begin{aligned} e_{p_{n'}} &= \max(0, e_{d_{n'}} - L - \delta_{p_{n'}}), \\ l_{p_{n'}} &= \min(l_{d_{n'}} - t_{p_{n'},d_{n'}} - \delta_{p_{n'}}, T). \end{aligned}$$

Following [1, Sec. 3], the DARP can be formulated as a mixed-integer program as follows.

$$\text{minimize} \quad \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{V}} c_{i,j}^k \phi_{i,j}^k, \tag{1}$$

$$\text{subject to} \quad \sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{V}} \phi_{i,j}^k = 1,\ \forall i \in \mathcal{P}, \tag{2}$$

$$\sum_{j\in\mathcal{V}} \phi_{p_n,j}^k - \sum_{j\in\mathcal{V}} \phi_{d_n,j}^k = 0,\ \forall n \in \mathcal{N},\ \forall k \in \mathcal{K}, \tag{3}$$

$$\sum_{j\in\mathcal{V}} \phi_{s^+,j}^k = 1,\ \forall k \in \mathcal{K}, \tag{4}$$

$$\sum_{j\in\mathcal{V}} \phi_{i,j}^k - \sum_{j\in\mathcal{V}} \phi_{j,i}^k = 0,\ \forall i \in \mathcal{P}\cup\mathcal{D},\ \forall k \in \mathcal{K}, \tag{5}$$

$$\sum_{i\in\mathcal{V}} \phi_{i,s^-}^k = 1,\ \forall k \in \mathcal{K}, \tag{6}$$

$$B_j^k \geq (B_i^k + t_{i,j})\phi_{i,j}^k,\ \forall i \in \mathcal{V},\ \forall j \in \mathcal{V},\ \forall k \in \mathcal{K}, \tag{7}$$

$$Q_j^k \geq (Q_i^k + q_j)\phi_{i,j}^k,\ \forall i \in \mathcal{V},\ \forall j \in \mathcal{V},\ \forall k \in \mathcal{K}, \tag{8}$$

$$L_n^k = B_{d_n}^k - (B_{p_n}^k + \delta_{p_n}),\ \forall n \in \mathcal{N},\ \forall k \in \mathcal{K}, \tag{9}$$

$$B_{s^-}^k - B_{s^+}^k \leq T,\ \forall k \in \mathcal{K}, \tag{10}$$

$$e_i \leq B_i^k \leq l_i,\ \forall i \in \mathcal{V},\ \forall k \in \mathcal{K}, \tag{11}$$

$$t_{p_n,d_n} - \delta_{p_n} \leq L_n^k \leq L,\ \forall n \in \mathcal{N},\ \forall k \in \mathcal{K}, \tag{12}$$

$$\max(0, q_i) \leq Q_i^k \leq \min(Q, Q + q_i),\ \forall i \in \mathcal{V},\ \forall k \in \mathcal{K}, \tag{13}$$

$$\phi_{i,j}^k \in \{0, 1\},\ \forall i \in \mathcal{V},\ \forall j \in \mathcal{V},\ \forall k \in \mathcal{K}. \tag{14}$$

The objective function (1) minimizes the total routing costs. Equality (2) ensures that each pick-up location is visited only one time. Equality (3) ensures that the pick-up and drop-off locations of each user are visited by the same vehicle. Equalities (4), (5), and (6) ensure that the route of each vehicle begins at the source station and ends at the destination station. Inequalities (7) and (8) guarantee the consistency of time and load variables. Equality (9) defines the ride time of each user and inequalities (12) bounds it. Inequality (10) bounds the route duration of each route. Inequality (11) imposes the time-window constraints. Inequality (13) imposes the vehicle-capacity constraints.

4 Model Formulation

In this section, we first formulate the DARP as a Markov decision process (MDP) by relaxing Constraints (10), (11), and (12). Based on the original formulation and the MDP formulation, we further design the inputs and outputs of the proposed architecture.

4.1 MDP Formulation

Based on Section 3, we define an MDP, denoted by $\mathcal{M}$, by introducing the following definitions.

1. *States:* At step τ, let $s_\tau = (u_\tau, v_\tau)$ be the state of the system, where $u = (u_\tau^1, u_\tau^2, \dots, u_\tau^N)$ is the state of N users and $v_\tau = (v_\tau^1, v_\tau^2, \dots, v_\tau^K)$ is the state of K vehicles. Furthermore, let

$$u_\tau^n = \big((x_{p_n}, y_{p_n}), (x_{d_n}, y_{d_n}), \delta_{p_n}, q_{p_n}, [e_{p_n}, l_{p_n}], [e_{d_n}, l_{d_n}]\big)$$

be the state of user $n \in \mathcal{N}$, and $v_\tau^k = (r_\tau^k, b_\tau^k, f_\tau^k)$ be the state of vehicle $k \in \mathcal{K}$. Here, r_τ^k is the path of vehicle k consisting of a set of locations, i.e.,

$$r_\tau^k = \Big\{s^+, i_1, i_2, \dots, i_{|r_\tau^k|-1}\Big\},$$

where $|r_\tau^k|$ is the number of locations visited by the vehicle. b_τ^k is the schedule of vehicle k consisting of a set of start time, i.e.,

$$b_\tau^k = \Big\{B_{s^+}^k, B_{i_1}^k, B_{i_2}^k, \dots, B_{i_{|r_\tau^k|-2}}^k\Big\},$$

where $|b^k_\tau| = |r^k_\tau| - 1$ due to the fact that the vehicle has not started a service at location $i_{|r^k_\tau|-1}$. f^k_τ is the next free time of vehicle k. Let $\mathcal{S}_\tau$ be the state space at step τ. Note that, at steps $1, 2, \dots, \tau$, the ride time of all users and the loads of all vehicles can be computed based on the states.

2. *Actions:* At each step, among vehicles that have the smallest next free time, the vehicle with the smallest index, denoted by κ_t, performs an action. At step τ, let a_τ be the action of the system. Given $s_\tau = s$ and $\kappa_\tau = \kappa$, we have

$$a_\tau \in \begin{cases} \mathcal{P}_{s,\kappa} \cup \mathcal{D}_{s,\kappa} \cup \{s^-\}, & \text{if vehicle } \kappa \text{ starts a service at location } i_{|r^\kappa|-1} \text{ and then moves to location } a_\tau, \\ \mathcal{W}, & \text{if vehicle } \kappa \text{ waits at location } i_{|r^\kappa|-1} \text{ for } a_\tau \text{ time units,} \end{cases}$$

where $\mathcal{P}_{s,\kappa}$ is the set of unvisited pick-up locations of which the corresponding users can be picked up by vehicle κ, $\mathcal{D}_{s,\kappa}$ is the set of unvisited drop-off locations of which the corresponding pick-up locations have been visited, and $\mathcal{W} = [0, l_{i_{|r^\kappa|-1}} - f^\kappa]$. Here, $\mathcal{P}_{s,\kappa}$ and $\mathcal{D}_{s,\kappa}$ can be given by

$$\mathcal{P}_{s,\kappa} = \{p_n \mid p_n \in \mathcal{P} \setminus \cup_{k=1}^{K}(\mathcal{P} \cap r^k),\ Q^\kappa_{i_{|r^\kappa|-1}} + q_{p_n} \le Q,\ n \in \mathcal{N}\}$$
$$\mathcal{D}_{s,\kappa} = \{d_n \mid p_n \in r^\kappa,\ d_n \notin r^\kappa,\ n \in \mathcal{N}\}.$$

According to Constraint (7), since waiting may happen after the beginning of time windows, it is necessary to take it into account. Let $\mathcal{A}_{s,\tau}$ be the action space in state s at step τ.

3. *State Transition Function:* At step τ, given $s_\tau = s$, $\kappa_\tau = \kappa$, and $a_\tau = a$, the state transition function yields the state of the system at step $\tau + 1$, defined as $s_{\tau+1} = F_\tau(s, a)$. For vehicle κ, we consider the following three cases. 1) If $a_\tau \in \mathcal{P}_{s,\kappa} \cup \mathcal{D}_{s,\kappa}$, then

$$\begin{aligned} r^\kappa_{\tau+1} &= r^\kappa \cup \{a\}, \\ b^\kappa_{\tau+1} &= b^\kappa \cup \{f^\kappa\}, \\ f^\kappa_{\tau+1} &= \max\Big(f^\kappa + \delta_{i_{|r^\kappa|-1}} + t_{i_{|r^\kappa|-1},a}, e_a\Big). \end{aligned}$$

2) If $a_\tau = s^-$, then

$$\begin{aligned} r^\kappa_{\tau+1} &= r^\kappa \cup \{s^-\}, \\ b^\kappa_{\tau+1} &= b^\kappa \cup \{f^\kappa, f^\kappa + \delta_{i_{|r^\kappa|-1}} + t_{i_{|r^\kappa|-1},s^-}\}, \\ f^\kappa_{\tau+1} &= 2T. \end{aligned}$$

3) If $a_\tau \in \mathcal{W}$, then

$$\begin{aligned} r^\kappa_{\tau+1} &= r^\kappa, \\ b^\kappa_{\tau+1} &= b^\kappa, \\ f^\kappa_{\tau+1} &= f^\kappa + a. \end{aligned}$$

In addition, we have $u^n_{\tau+1} = u^n$ for each user $n \in \mathcal{N}$ and $v^k_{\tau+1} = v^k$ for each vehicle $k \in \mathcal{K} \setminus \{\kappa\}$. Note that, if all vehicles arrive at the destination station (i.e., the smallest next free time of K vehicles equals $2T$), the MDP will be terminated. We denote the last step by $\bar{\tau}$.

4. *Policies:* A policy $\boldsymbol{\pi}$ is defined by a sequence of decision rules, i.e.,

$$\boldsymbol{\pi} = (\pi_1, \pi_2, \dots, \pi_{\bar{\tau}}),$$

where $\pi_\tau \colon \mathcal{S}_\tau \to \mathcal{A}_{s,\tau}$ specifies the action choice when the system occupies state s at step τ.

5. *Rewards:* Given $s_\tau = s$, $\kappa_\tau = \kappa$, and $a_\tau = a$, let $R_\tau(s, a)$ be the reward of the system when vehicle κ takes action a in state s at step τ. So, we have

$$R_\tau(s, a) = \begin{cases} c_{i_{|r^\kappa|-1},a}, & \text{if } a_\tau \in \mathcal{P}_{s,\kappa} \cup \mathcal{D}_{s,\kappa} \cup \{s^-\}, \\ 0, & \text{if } a_\tau \in \mathcal{W}. \end{cases}$$

Let $R^{\boldsymbol{\pi}}(s_0)$ be the expected total reward from step 1 to step $\bar{\tau}$ when $s_1 = s_0$ and policy $\boldsymbol{\pi}$ is used, defined by

$$R^{\boldsymbol{\pi}}(s_0) = \mathbf{E}^{\boldsymbol{\pi}}\Bigg[\sum_{\tau=1, s_1=s_0}^{\bar{\tau}} R_\tau\big(s_\tau, \pi_\tau(s_\tau)\big)\Bigg],$$

where s_0 is the initial state of the system given by an instance of the DARP and $v^k = \big\{\{s^+\}, \emptyset, 0\big\}$ for each $k \in \mathcal{K}$.

4.2 Inputs and Outputs

Based on Sections 3 and 4.1, we are ready to design the inputs and outputs.

At step τ, given $s_\tau = s$ and $\kappa_\tau = \kappa$, let $X_\tau = (\widetilde{u}_\tau^1, \widetilde{u}_\tau^2, \ldots, \widetilde{u}_\tau^N)$ be the input consisting of the feature tuples of N users. Furthermore, each user $n \in \mathcal{N}$ has $9 + K$ features as follows.

- The coordinates of the pick-up and drop-off locations of user n: (x_{p_n}, y_{p_n}) and (x_{d_n}, y_{d_n}).
- The beginning and end of the shifted pick-up and drop-off time windows of user n at step τ, denoted by $(\widetilde{e}_{p_n,\tau}, \widetilde{l}_{p_n,\tau})$ and $(\widetilde{e}_{d_n,\tau}, \widetilde{l}_{d_n,\tau})$:

$$\begin{aligned}(\widetilde{e}_{p_n,\tau}, \widetilde{l}_{p_n,\tau}) &= \big(\max(0, e_{p_n} - f^\kappa), \max(0, l_{p_n} - f^\kappa)\big),\\ (\widetilde{e}_{d_n,\tau}, \widetilde{l}_{d_n,\tau}) &= \big(\max(0, e_{d_n} - f^\kappa), \max(0, l_{d_n} - f^\kappa)\big).\end{aligned}$$

  Here, we shift time windows to make the proposed architecture more sensitive to time constraints.
- The served status of user n at step τ, denoted by $k_{n,\tau}$:

$$k_{n,\tau} \in \begin{cases}\{1, 2, \ldots, K\}, & \text{if vehicle } k_{n,\tau} \text{ has visited location } p_n \text{ and has not visited location } d_n,\\ \{K+1\}, & \text{otherwise.}\end{cases}$$

- The ride time of user n at step τ, denoted by $L_{n,\tau}$:

$$L_{n,\tau} = \begin{cases} f^{k_{n,\tau}} - B_{p_n}^{k_{n,\tau}}, & \text{if vehicle } k_{n,\tau} \text{ has picked up user } n \text{ and has not started to drop off user } n,\\ 0, & \text{otherwise.}\end{cases}$$

- The request flag of user n at step τ, denoted by $\alpha_{n,\tau}$:

$$\alpha_{n,\tau} = \begin{cases} 0, & \text{if no vehicle has visited location } p_n,\\ 1, & \text{if vehicle } k_{n,\tau} \text{ has visited location } p_n \text{ and has not visited location } d_n,\\ 2, & \text{otherwise.}\end{cases}$$

- The index of the vehicle which performs an action at step τ: κ.
- The service flag of user n at step τ, denoted by $\beta_{n,\tau}$:

$$\beta_{n,\tau} = \begin{cases} 0, & \text{if no vehicle has visited location } p_n \text{ and vehicle } \kappa \text{ is able to pick up user } n,\\ 1, & \text{if vehicle } \kappa \text{ has visited location } p_n \text{ and has not visited location } d_n,\\ 2, & \text{otherwise.}\end{cases}$$

- The travel time from each vehicle $k \in \mathcal{K}$ to the pick-up or drop-off location of user n, denoted by $t_{n,\tau}^k$:

$$t_{n,\tau}^k = \begin{cases} \delta_{i_{|r^k|-1}} + t_{i_{|r^k|-1}, p_n}, & \text{if } \alpha_{n,\tau} = 0,\\ \delta_{i_{|r^k|-1}} + t_{i_{|r^k|-1}, d_n}, & \text{otherwise.}\end{cases}$$

Thus, we have $\widetilde{u}_\tau^n = \big((x_{p_n}, y_{p_n}), (x_{d_n}, y_{d_n}), (\widetilde{e}_{p_n,\tau}, \widetilde{l}_{p_n,\tau}), (\widetilde{e}_{d_n,\tau}, \widetilde{l}_{d_n,\tau}), k_{n,\tau}, L_{n,\tau}, \alpha_{n,\tau}, \kappa, \beta_{n,\tau}, t_{n,\tau}^1, \ldots, t_{n,\tau}^K\big)$ for each user $n \in \mathcal{N}$.

Before designing the outputs, we first modify $\mathcal{M}$ to $\widetilde{\mathcal{M}}$ by only redefining the action at each step. At step τ, given $s_\tau = s$ and $\kappa_\tau = \kappa$, the new action, denoted by $\widetilde{a}_\tau$, is given as follows:

$$\widetilde{a}_\tau \in \begin{cases} \mathcal{P}_{s,\kappa} \cup \mathcal{D}_{s,\kappa} \cup \{s^-\}, & \text{if vehicle } \kappa \text{ starts a service at location } i_{|r^\kappa|-1} \text{ and then moves to location } \widetilde{a}_\tau,\\ \widetilde{\mathcal{W}}, & \text{if vehicle } \kappa \text{ waits at location } i_{|r^\kappa|-1} \text{ for } \widetilde{a}_\tau \text{ time units,}\end{cases}$$

where $\widetilde{\mathcal{W}} = \{w\}$ and $w \in [0, \max_{i \in \mathcal{P} \cap \mathcal{D}} l_i - e_i]$ is a constant. Here, when vehicle κ decides to wait at location $i_{|r^\kappa|-1}$, the range of values of the original action is modified from a real interval $\mathcal{W}$ to a singleton set $\widetilde{\mathcal{W}}$. A vehicle may take the new action for multiple consecutive steps to approximate the original action at one step. The motivation for doing this is to transfer action prediction into a classification problem.

At step τ, given $s_\tau = s$, $\kappa_\tau = \kappa$, and $\widetilde{a}_\tau = \widetilde{a}$, let Y_τ be the output as follows:

$$Y_\tau \in \begin{cases} \mathcal{N}, & \text{if } \widetilde{a}_\tau = p_{Y_\tau} \text{ or } \widetilde{a}_\tau = d_{Y_\tau}, \\ \{s^-\}, & \text{if } \widetilde{a}_\tau = s^-, \\ \widetilde{\mathcal{W}}, & \text{otherwise.} \end{cases}$$

Thus, Y_τ has $|\mathcal{N}| + 2$ possible values. When doing inference, given $s_\tau = s$, $\kappa_\tau = \kappa$, $X_\tau = X$, and $Y_\tau = Y$, we have

$$\widetilde{a}_\tau = \begin{cases} p_Y, & \text{if } Y_\tau \in \mathcal{N} \text{ and } \beta_{n,\tau} = 0, \\ d_Y, & \text{if } Y_\tau \in \mathcal{N} \text{ and } \beta_{n,\tau} = 1, \\ s^-, & \text{if } Y_\tau = s^-, \\ w, & \text{otherwise.} \end{cases}$$

Given an instance of the DARP and policy $\boldsymbol{\pi}$ (such as the solutions of the RF algorithm), we can generate $\bar{\tau}$ pairs of (X, Y) by the realization of $\widetilde{\mathcal{M}}$, which can be used to create training and validation sets for supervised learning.

5 Proposed Architecture

5.1 Embeddings

As designed in Section 4.2, an input consists of the feature tuples of N user and each user has $9 + K$ features. For all users, we use learned embeddings to convert the features, of which the values are discrete, to vectors of dimension $d_{\text{model}} = 128$, and use learned linear transformations to convert the features, of which the values are continuous, to vectors of the same dimension. Given an arbitrary input X, for each user $n \in \mathcal{N}$, we have:

$$h_{n,1} = \text{embedding}_1(k_n) \in \mathbb{R}^{d_{\text{model}}},\ h_{n,2} = \text{embedding}_2(\alpha_n) \in \mathbb{R}^{d_{\text{model}}},$$
$$h_{n,3} = \text{embedding}_3(\kappa) \in \mathbb{R}^{d_{\text{model}}},\quad h_{n,4} = \text{embedding}_4(\beta_n) \in \mathbb{R}^{d_{\text{model}}},$$

and

$$\begin{aligned} &h_{n,5} = (x_{p_n}, y_{p_n})W_1 + b_1 \in \mathbb{R}^{d_{\text{model}}}, && h_{n,6} = (x_{d_n}, y_{d_n})W_1 + b_1 \in \mathbb{R}^{d_{\text{model}}}, && W_1 \in \mathbb{R}^{2\times d_{\text{model}}},\ b_1 \in \mathbb{R}^{d_{\text{model}}}, \\ &h_{n,7} = (\widetilde{e}_{p_n}, \widetilde{l}_{p_n})W_2 + b_2 \in \mathbb{R}^{d_{\text{model}}}, && h_{n,8} = (\widetilde{e}_{d_n}, \widetilde{l}_{d_n})W_2 + b_2 \in \mathbb{R}^{d_{\text{model}}}, && W_2 \in \mathbb{R}^{2\times d_{\text{model}}},\ b_2 \in \mathbb{R}^{d_{\text{model}}}, \\ &h_{n,9} = L_n W_3 + b_3 \in \mathbb{R}^{d_{\text{model}}}, && h_{n,9+k} = t_n^k W_3 + b_3 \in \mathbb{R}^{d_{\text{model}}},\ \forall k \in \mathcal{K}, && W_3 \in \mathbb{R}^{d_{\text{model}}},\ b_3 \in \mathbb{R}^{d_{\text{model}}}. \end{aligned}$$

Here, the four lookup tables that store embeddings, and the weights and bias of the three linear transformations are shared among all N users.

5.2 Cascade Encoders

The proposed architecture consists of two standard Transformer encoders [7]: a user encoder and a main encoder. Both encoders consist of a stack of identical encoding layers. Moreover, each encoding layer has two sub-layers: a multi-head self-attention (MHA) mechanism and a position-wise feed-forward network (FFN). The user encoder, of which the number of input tokens is $9 + K$, has two encoding layers. Formally, for each user $n \in \mathcal{N}$, it takes $H_n^0 = \text{Concat}(h_{n,1}, h_{n,2}, \ldots, h_{n,9+K}) \in \mathbb{R}^{(9+K)\times d_{\text{model}}}$ as inputs, and its equations are given by

$$\begin{aligned} H_n^{\text{enc}} &= H_n^2 \in \mathbb{R}^{(9+K)\times d_{\text{model}}}, \\ \text{where } H_n^{m+1} &= \text{LayerNorm}\big(\text{LayerNorm}(H_n^m + \text{MHA}_n^m) + \text{FFN}_n^m\big) \in \mathbb{R}^{(9+K)\times d_{\text{model}}}, \\ \text{FFN}_n^m &= \max\big(0, \text{LayerNorm}(H_n^m + \text{MHA}_n^m)W_m^1 + b_m^1\big)W_m^2 + b_m^2 \in \mathbb{R}^{(9+K)\times d_{\text{model}}}, \\ & \qquad W_m^1 \in \mathbb{R}^{d_{\text{model}}\times d_{\text{ff}}},\ b_m^1 \in \mathbb{R}^{d_{\text{ff}}},\ W_m^2 \in \mathbb{R}^{d_{\text{ff}}\times d_{\text{model}}},\ b_m^2 \in \mathbb{R}^{d_{\text{model}}}, \\ \text{MHA}_n^m &= \text{Concat}(\text{head}_n^{m,1}, \text{head}_n^{m,2}, \ldots, \text{head}_n^{m,Z})W_m^O \in \mathbb{R}^{(9+K)\times d_{\text{model}}},\ W_m^O \in \mathbb{R}^{Zd_V\times d_{\text{model}}}, \\ \text{head}_n^{m,z} &= \text{softmax}\left(\frac{Q_n^{m,z}(K_n^{m,z})^T}{\sqrt{d_K}}\right)V_n^{m,z} \in \mathbb{R}^{(9+K)\times d_V},\ z \in \{1, 2, \ldots, Z\}, \qquad (15) \\ Q_n^{m,z} &= H_n^m W_{m,z}^Q \in \mathbb{R}^{(9+K)\times d_K},\ W_{m,z}^Q \in \mathbb{R}^{d_{\text{model}}\times d_K},\ z \in \{1, 2, \ldots, Z\}, \\ K_n^{m,z} &= H_n^m W_{m,z}^K \in \mathbb{R}^{(9+K)\times d_K},\ W_{m,z}^K \in \mathbb{R}^{d_{\text{model}}\times d_K},\ z \in \{1, 2, \ldots, Z\}, \\ V_n^{m,z} &= H_n^m W_{m,z}^V \in \mathbb{R}^{(9+K)\times d_V},\ W_{m,z}^V \in \mathbb{R}^{d_{\text{model}}\times d_V},\ z \in \{1, 2, \ldots, Z\}. \end{aligned}$$

We set $Z = 8$ (i.e., eight parallel heads), $d_K = d_V = 64$, and $d_{\text{ff}} = 2048$. Then, for each user $n \in \mathcal{N}$, we apply a learned linear transformation on H_n^{enc}, i.e.,

$$\widetilde{h}_n = \text{Flatten}(H_n^{\text{enc}})W_4 + b_4 \in \mathbb{R}^{d_{\text{model}}},\ W_4 \in \mathbb{R}^{(9+K)d_{\text{model}} \times d_{\text{model}}},\ b_4 \in \mathbb{R}^{d_{\text{model}}}.$$

The weights and bias of the user encoder and the linear transformation are also shared among all N users.

Furthermore, the main encoder, of which the number of input tokens is N, has four encoding layers. Formally, it takes $\widetilde{H}^0 = \text{Concat}(\widetilde{h}_1, \widetilde{h}_2, \dots, \widetilde{h}_N) \in \mathbb{R}^{N \times d_{\text{model}}}$ as inputs, and its equations are given by

$$\begin{aligned}
\widetilde{H}^{\text{enc}} &= \widetilde{H}^4 \in \mathbb{R}^{N \times d_{\text{model}}},\\
\text{where } \widetilde{H}^{m+1} &= \text{LayerNorm}\big(\text{LayerNorm}(\widetilde{H}^m + \widetilde{\text{MHA}}^m) + \widetilde{\text{FFN}}^m\big) \in \mathbb{R}^{N \times d_{\text{model}}},\\
\widetilde{\text{FFN}}^m &= \max\big(0, \text{LayerNorm}(\widetilde{H}^m + \widetilde{\text{MHA}}^m)\widetilde{W}_m^1 + \widetilde{b}_m^1\big)\widetilde{W}_m^2 + \widetilde{b}_m^2 \in \mathbb{R}^{N \times d_{\text{model}}},\\
&\qquad \widetilde{W}_m^1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}},\ \widetilde{b}_m^1 \in \mathbb{R}^{d_{\text{ff}}},\ \widetilde{W}_m^2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}},\ \widetilde{b}_m^2 \in \mathbb{R}^{d_{\text{model}}},\\
\widetilde{\text{MHA}}^m &= \text{Concat}(\widetilde{\text{head}}^{m,1}, \widetilde{\text{head}}^{m,2}, \dots, \widetilde{\text{head}}^{m,Z})\widetilde{W}_m^O \in \mathbb{R}^{N \times d_{\text{model}}},\ \widetilde{W}_m^O \in \mathbb{R}^{Zd_V \times d_{\text{model}}},\\
\widetilde{\text{head}}^{m,z} &= \text{softmax}\bigg(\frac{\widetilde{Q}^{m,z}(\widetilde{K}^{m,z})^T}{\sqrt{d_K}}\bigg)\widetilde{V}^{m,z} \in \mathbb{R}^{N \times d_V},\ z \in \{1, 2, \dots, Z\}, \qquad (16)\\
\widetilde{Q}^{m,z} &= \widetilde{H}^m \widetilde{W}_{m,z}^Q \in \mathbb{R}^{N \times d_K},\ \widetilde{W}_{m,z}^Q \in \mathbb{R}^{d_{\text{model}} \times d_K},\ z \in \{1, 2, \dots, Z\},\\
\widetilde{K}^{m,z} &= \widetilde{H}^m \widetilde{W}_{m,z}^K \in \mathbb{R}^{N \times d_K},\ \widetilde{W}_{m,z}^K \in \mathbb{R}^{d_{\text{model}} \times d_K},\ z \in \{1, 2, \dots, Z\},\\
\widetilde{V}^{m,z} &= \widetilde{H}^m \widetilde{W}_{m,z}^V \in \mathbb{R}^{N \times d_V},\ \widetilde{W}_{m,z}^V \in \mathbb{R}^{d_{\text{model}} \times d_V},\ z \in \{1, 2, \dots, Z\}.
\end{aligned}$$

As shown above, the user encoder converts the $9 + K$ embedding vectors of a user to a vector of dimension d_{model} by learning the relationship among its different features. The main encoder further converts the N output vectors to a vector of dimension d_{model} by learning the relationship among different users. By designing the cascade-encoder architecture, we largely reduce the number of learnable parameters, i.e., the numbers of the input tokens of the two encoders only increase linearly with K and N.

5.3 Softmax and Masks

First, we apply a learned pre-softmax linear transformation to convert $\widetilde{H}^{\text{enc}}$ to a vector of dimension $N + 2$, i.e.,

$$h^{\text{pre}} = \text{Flatten}(\widetilde{H}^{\text{enc}})W_5 + b_5 \in \mathbb{R}^{N+2},\ W_5 \in \mathbb{R}^{Nd_{\text{model}} \times (N+2)},\ b_5 \in \mathbb{R}^{N+2}.$$

To ensure that each user must be picked up or dropped off by the same vehicle, we design an output mask $M^O \in \mathbb{R}^{N+2}$ based on the service flag of each user:

$$M^O(\widetilde{n}) = \begin{cases} 1, & \text{if } \widetilde{n} \in \mathcal{N} \text{ and } \beta_{\widetilde{n}} \in \{0, 1\},\\ 0, & \text{otherwise.} \end{cases}$$

Next, we apply the softmax function with the output mask to convert h^{pre} to a vector of predicted probabilities:

$$\sigma = \text{softmax}(h^{\text{pre}} \oplus M^O) \in \mathbb{R}^{N+2},$$

where $\sigma(\widetilde{n}) \in [0, 1]$ and r$\sum_{\widetilde{n}=1}^{N+2} \sigma(\widetilde{n}) = 1$. Furthermore, the predicted output can be determined by σ.

5.4 "One for All"

Although the numbers of the input tokens of the user encoder and the main encoder only increase linearly with K and N respectively, we still need to train different models for different types of instances (i.e., with different values of K and N). Is it possible to train a model on one type of instances and use the model to do inference on all other types of instances? To deal with this issue, we extend the cascade-encoder architecture by considering dummy users and vehicles, namely the one-for-all architecture. Assuming that we train a model on one type of instances with $K = K_{\text{train}}$ and $N = N_{\text{train}}$, for an arbitrary type of instances with $K = K_{\text{test}} \leq K_{\text{train}}$ and $N = N_{\text{test}} \leq N_{\text{train}}$, the number of dummy users and vehicles are given by $N_{\text{train}} - N_{\text{test}}$ and $K_{\text{train}} - K_{\text{test}}$ respectively.

At step τ, for the state of each dummy user $n \in \{N_{\text{test}} + 1, N_{\text{test}} + 2, \dots, N_{\text{train}}\}$, we have

$$u_\tau^n = \big((0, 0), (0, 0), 0, 0, [0, 0], [0, 0]\big).$$

For the state of each dummy vehicle $k \in \{K_{\text{test}}+1, K_{\text{test}}+2, \ldots, K_{\text{train}}\}$, we have $v_\tau^k = \big(\{s^+\}, \emptyset, 2T\big)$. Furthermore, for the feature tuple of each dummy user $n \in \{N_{\text{test}}+1, N_{\text{test}}+2, \ldots, N_{\text{train}}\}$, we have

$$k_{n,\tau} = K+1,\ \alpha_{n,\tau} = 2,\ \beta_{n,\tau} = 2.$$

To reduce the influence of introducing dummy users and vehicles on the two encoders, we further design a user mask $M^U \in \mathbb{R}^{9+K_{\text{train}}}$ and a main mask $M^M \in \mathbb{R}^{N_{\text{train}}}$:

$$M^U(\widetilde{n}) = \begin{cases} 1, & \text{if } 1 \le \widetilde{n} \le 9 + K_{\text{test}}, \\ 0, & \text{otherwise,} \end{cases}$$

$$M^M(\widetilde{n}) = \begin{cases} 1, & \text{if } 1 \le \widetilde{n} \le N_{\text{test}}, \\ 0, & \text{otherwise.} \end{cases}$$

Hence, the softmax functions in Eqs. (15) and (16) can be modified by

$$\text{head}_n^{m,z} = \text{softmax}\bigg(\frac{Q_n^{m,z}(K_n^{m,z})^T}{\sqrt{d_K}} \oplus M^U\bigg)V_n^{m,z} \in \mathbb{R}^{(9+K_{\text{train}})\times d_V},\ z \in \{1,2,\ldots,Z\},$$

$$\widetilde{\text{head}}^{m,z} = \text{softmax}\bigg(\frac{\widetilde{Q}^{m,z}(\widetilde{K}^{m,z})^T}{\sqrt{d_K}} \oplus M^M\bigg)\widetilde{V}^{m,z} \in \mathbb{R}^{N_{\text{train}}\times d_V},\ z \in \{1,2,\ldots,Z\}.$$

5.5 Decoding Technique

In a fixed state, the model outputs a probability distribution on the possible next actions. We can recover a routing schedule, by greedily sampling the most promising action

$$a_t = \arg\max_a \mathbb{P}(a|a_{t-1}, \ldots, a_1, s_t). \tag{17}$$

The time complexity of this procedure is $\mathcal{O}(n)$. Better sampling techniques, enable us to explore the search space more exhaustively. Beam search maintains B partial solutions at each step, based on the sum of the log probabilities.

$$\{(a_{1,j})_{j=1}^{t_1}, \ldots, (a_{B,j})_{j=1}^{t_B}\} = \arg topB\{\sum_{j=1}^{t_b} \mathbb{I}\{(a_{b,j})_{j=1}^{t_b} \neq wait\} \log \mathbb{P}(a_{b,j}|a_{b,j-1}, \ldots, a_{b,1}, s_k)\}_{b=1}^{B^2} \tag{18}$$

At each step, a partial solution is expanded by B new leaf, but we keep only the B most promising ones. After terminating, the procedure outputs, B routing schedules, we choose the one with the lowest time penalty. The time complexity increases to $\mathcal{O}(nB)$. The waiting action can be performed as many times as we want, this would make the partial solutions impossible to compare. We can overcome this issue, by letting the model sample the wait action greedily, and only branch along all other actions, when waiting is not the most probable choice anymore. We don't include the waiting log probs in the sum, this means each sum becomes comparable with $2N + K$ variables each.

5.6 Reinforcement Learning

In this paper, we trained our model with Reinforcement Learning on top of the Supervised Learning trained transformer. For this purpose, the policy gradient algorithm is used to train the parameters. This method iteratively updates the parameters of the policy network based on minimizing the loss.

$$\nabla L(\theta|s) = -\mathbb{E}_{\pi\sim p_\theta}[R_s(\mathcal{A}_{\pi_\theta}^s, \mathcal{A}_{RIST}^s)\nabla \log p_\theta(\pi_\theta|s)] \tag{19}$$

Where $p_\theta(\pi_\theta|s)$ is the probability distribution of the set of actions taken by the policy π_θ in order to solve the instance s and R_s is defined as follows:

- Given $\mathcal{A}_{RIST}^s$ the set of actions taken by the RIST2021 algorithm, $\mathcal{A}_{\pi_\theta}^s$ the set of actions taken by the policy π_θ to solve the instance s and $\Delta_d(a)$ the cost of the action a, we define $R_d(\mathcal{A}_{\pi_\theta}^s, \mathcal{A}_{RIST}^s)$ as the cost reward for taking the set of actions $\mathcal{A}_{\pi_\theta}^s$ by:

$$R_d(\mathcal{A}_{\pi_\theta}^s, \mathcal{A}_{RIST}^s) = \max\left\{\frac{\left[\sum_{a\in\mathcal{A}_{\pi_\theta}^s} \Delta_d(a) - \sum_{a\in\mathcal{A}_{RIST}^s} \Delta_d(a)\right]\cdot 100}{\sum_{a\in\mathcal{A}_{RIST}^s} \Delta_d(a)}, 0\right\}$$

- Let δ_i be the ride time of user i, $[e_{p_i}, l_{p_i}]$ and $[e_{d_i}, l_{d_i}]$ the pick-up and the drop-off time windows of user i and t_{p_i}, t_{d_i} the actual pick up and drop off time. The time reward is:

$$R_t(\mathcal{A}^s_{\pi_\theta}) = \sum_{i=1}^{N} \left[\max\{\delta_i - \delta_{max}, 0\} + \max\{e_{p_i} - t_{p_i}, 0\} + \max\{t_{p_i} - l_{p_i}, 0\}] + \max\{e_{d_i} - t_{d_i}, 0\} + \max\{t_{d_i} - l_{d_i}, 0\} \right]$$

Let α the cost ratio hyper parameter, then the total reward for the instance s solved with the policy π_θ is:

$$R_s(\mathcal{A}^s_{\pi_\theta}, \mathcal{A}^s_{RIST}) = R_d(\mathcal{A}^s_{\pi_\theta}, \mathcal{A}^s_{RIST}) + \alpha R_t(\mathcal{A}^s_{\pi_\theta}) \tag{20}$$

Adam optimizer is used to train the policy network parameters by minimizing the $\nabla L(\theta|s)$. The pseudocode for the training steps is illustrated in Algorithm 1.

Algorithm 1: Deep Reinforcement Learning Algorithm

```
1  Input: Number of instances N; Instance type; The supervised learning policy $\pi$; cost ratio $\alpha$
2  Output: A new policy $\pi_\theta$
3  Load the model $\pi$
4  $\pi_\theta \leftarrow \pi$
5  foreach $s = 1, 2, ..., N$ do
6      Initiate the environment with instance s;
7      Reset gradients $\nabla_\theta \leftarrow 0$;
8      Solve the instance with policy $\pi_\theta$:
9      $\mathcal{A}^s_{\pi_\theta} \leftarrow Greedy(\pi_\theta, s)$ ;
10     Solve the instance with RIST2021;
11     $\mathcal{A}^s_{RIST} \leftarrow RIST(s)$ ;
12     Compute policy reward $R_s(\mathcal{A}^s_{\pi_\theta}, \mathcal{A}^s_{RIST}, \alpha)$;
13     $\nabla L(\theta|s) = -\mathbb{E}_{\pi \sim p_\theta}[R_s(\mathcal{A}^s_{\pi_\theta}, \mathcal{A}^s_{RIST}, \alpha) \nabla \log p_\theta(\pi_\theta|s)]$;
14     Update $\theta$ using $\nabla L(\theta|s)$;
15 end
16 return $\pi_\theta$
```

6 Numerical Experiments

Cordeau's [1] proposed benchmark dataset is used to evaluate the models' performance. The dataset consists of 48 instances, splitted into 24 type A and B instances. The pickup and delivery nodes are generated randomly on a $[-10, 10]^2$ grid. Vehicles, in type A instances, have a maximum capacity of 3, each user's associated load is set to one, while the service duration $d_i = 3$. The maximum ride time $L = 30$. In the case of type B instances, the maximum capacity $Q = 6$, while the load varies from one to six, and the corresponding service time is equal to the load. The maximum ride time L is equal to 45 for all users. Half of the users are considered outbound, meaning their pickup time window is $[0, T]$, similarly, the other half are considered inbound, and their dropoff time window does not have restrictions. The other time windows have a length of 15 and are generated independently. The routing cost is equal to the Euclidean distance between the nodes.

Using the same setting, we generated for each instance a training set of 10000 instances, along with a test set of 100 instances. T Time is the average training time for 10k instances for an epoch. I Time is the average inference time to run a single instance. Results are compared to the Restricted Fragment (RF) algorithm [5], which solves a mixed integer programming problem with Gurobi [8]. For each test instance, the gap is calculated between RF and our solution, and the average is reported, along with the average routing costs. Finally, since our model considers soft constraints, we calculate the number of the broken time window and maximum ride time constraints, along with the amount of time they are violated. The average is reported in each case. For comparison, Table 3 contains the results on standard "type a" instances, and similarly, Table 4 contains the results on standard "type b" instances, using the supervised models, reinforcement models, the One for All formulation, along with the beam search decoder.

6.1 Supervised learning

The network embedding dimension for each token was set to $d_{model} = 128$, the number of encoder layers were set to 4. The training set was first solved with the RF algorithm, we generated state-action pairs, from these solutions and feed it into the model through 15 epochs, with a batch size of 256. The Adam optimizer, with a learning rate of 10^{-4} was used. During the training process the forward pass of the model was optimized, making the inference time much quicker, an asterisk denotes the cases when the improved model was used. The results on 100 instances are summarized in Table 1 below.

Instance	Obj	Cost	Δ	N_{TW}	N_{RT}	Penalty	T time	I Time
a2-16	271.24	292.30	8.52	1.11	2.66	41.35	1152.03	7.35
a2-20	337.04	361.46	8.42	1.39	3.21	48.35	1584.24	10.58
a2-24	402.54	429.40	7.82	1.85	4.15	64.66	2160.96	15.38
a3-24	371.17	402.11	8.82	2.69	4.65	93.79	2419.01	17.31
a3-36	547.44	591.87	8.37	4.58	7.08	155.50	3979.82	22.98
a4-32	470.79	520.17	10.72	4.40	7.33	152.61	3632.00	22.05
a4-40	567.43	630.11	11.17	5.38	8.36	176.57	2080.68*	7.29*
a4-48	676.54	749.78	10.98	7.00	11.37	235.13	2902.71*	9.4*
b2-16	281.14	297.38	6.64	1.20	0.62	34.43	1072.43	5.81
b2-20	349.83	369.39	6.67	1.74	0.81	51.80	1523.72	10.19
b2-24	413.06	439.34	7.09	2.13	0.98	60.57	2088.09	14.13
b3-24	390.31	416.80	7.35	2.96	1.29	84.93	2278.98	16.59
b3-36	574.15	617.31	7.91	4.33	1.87	131.48	3765.35	22.20
b4-32	491.11	531.76	8.60	4.90	1.67	154.95	1339.21*	5.03*
b4-40	-	-	-	-	-	-	-	-
b4-48	-	-	-	-	-	-	-	-

Table 1: Supervised results on 100 test instances

6.2 Reinforcement Learning

The agent is trained on 300 instances from the training set with cost ratio $\alpha \in \{0, 1, 10\}$ depending on the gap between penalties and cost. The Adam optimizer is used to optimize over the loss for one iteration per instance with a learning rate = 10^{-6}. The model is then tested on the same 100 test instances as the Supervised Learning. The T time wasn't reported since it was not significant for a small train set.

Instance	Obj	Cost	Δ	N_{TW}	N_{RT}	Penalty	T Time	I Time
a2-16	271.24	293.50	8.84	0.7	2.06	25.26	-	1.37
a2-20	337.04	365.04	9.19	1.09	2.28	34.77	-	2.39
a2-24	402.54	434.72	8.65	1.39	3.3	49.19	-	2.35
a3-24	371.17	402.91	8.83	2.92	3.95	92.28	-	3.86
a3-36	547.44	598.23	9.42	4.17	5.38	126.67	-	6.72
a4-32	470.79	523.55	11.35	9.23	6.94	288.39	-	6.14
a4-40	567.43	628.54	10.85	4.68	7.41	159.43	-	9.11
a4-48	-	-	-	-	-	-	-	-
b2-16	281.14	297.73	6.61	0.84	0.52	21.54	-	1.65
b2-20	349.83	369.25	6.47	1.36	0.59	38.77	-	2.27
b2-24	413.06	445.53	8.42	1.61	0.64	41.39	-	2.31
b3-24	390.31	419.77	8.02	2.85	1.06	78.32	-	2.84
b3-36	570.26	611.58	7.5	4.05	1.73	119.53	-	6.21
b4-32	491.11	533.9	9.08	10.79	1.53	535.38	-	8.47
b4-40	-	-	-	-	-	-	-	-
b4-48	-	-	-	-	-	-	-	-

Table 2: Reinforcement learning results on 100 test instances

7 Discussion

Given a snapshot of the environment, the Transformer model managed to reproduce single decisions of the Restricted Fragments algorithm in many cases (the waiting action being the vast majority of the actions), however the MDP formulation allows to make only local decisions. For larger instances the search space grows, and our model breaks more and more constraints. Additionally, the training set contains only feasable state-action pairs, this means every time our model deviates from a feasable solution, it will make its decsion based on unfeasable states, which are not part of the training dataset, which will lead to accumulated time penalties. To overcome this problem, we tuned the supervised model with the help of the policy gradient algorithm. As we can see, reinforcement learning training has improved significantly the number of broken constraints with a slight increase in cost compared to the supervised model and this shows that optimizing the model over cost and time penalties is a good approach to take when dealing with constrained routing problems. But when we analyze the bigger instances we can see a significant increase in broken time windows compared to the supervised learning model and this might be caused by the combining cost and time rewards in the same reward which might cause the agent to misunderstand the reward when the instances become lager and this highlight the need for a more complex reinforcement learning algorithm to separate time and cost rewards when optimizing the model. It also shows the huge potential of neural networks to solve NP-hard problems by just training them on resolved instances and then optimizing them to reduce the gaps with the optimal solutions.

8 Future Works

8.1 Improvement On The Reinforcement Learning Model

The current RL model already achieved good results and showed its efficiency but it can be further improved by the following:

- *Using a more sophisticated learning algorithm*
  The current model is only using Policy Gradient algorithm to reduce the penalty which is composed of Time and Cost penalties, so we plan to use a more sophisticated algorithm to better target the parameters when optimizing the policy depending cost or time.
- *Changing the state formulation*
  The current model is solving the instances in a centralized way, meaning that at each state we only have the state of the vehicle that will take the action and not the whole environment like a decentralized approach which can improve the coordination between vehicles on bigger instances.
- *Adding user ride time prediction*
  We will add the ride time prediction for the users to give the model more information about the ride before assigning it to a vehicle, this can influence the agent's behavior when picking up deliveries to improve his strategy.
- *Giving the instant penalties*
  This can results in a bigger training storage and a longer computation time when training but makes the training process much more efficient.

8.2 Online Version

This formulation is supposed to be online as it takes the tasks as an input stream in time. In opposition, the same problem can be considered offline, when all tasks are known from the start. In real applications, online solving makes more sense because customers of a Dial-A-Ride service will be calling for a ride along the day and this model needs to adapt itself according to new requests.

9 Conclusion

This paper introduced a Multi-Vehicle Pickup and Delivery Problem with time constraints (DARP) and modeled it as a Markov Decision Process. A solution method was proposed based on deep learning; therefore, this study makes the following contributions: (i) A new MDP formulation along with the Transformer architecture was created to enable us to clone these human-crafted heuristics to some extent through a supervised learning framework to solve the DARP problem. (ii) A new Reinforcement Learning model was added to further improve the first results and optimize the policy and reduce the number of broken time constraints. (iii) An adaptive model was created where we can seamlessly change between instance types without the need of training them on that specific instance. (vi) Optimized the states embedding and the environment so that the solutions are almost instantaneous. (iv) A set of computational experiments was conducted to evaluate the proposed method and analyze its properties. The main results can be summarized as follows:

- The proposed model with supervised learning was able to produce solutions with an average cost gap of 9.35% on A instances with an average number of broken time constraints of 9.65 and 7.38% on b instances with an average number of broken time constraints of 4.08.
- The proposed model when adding reinforcement learning has achieved an average cost gap of 9.86% on A instances with an average number of broken time constraints of 7.92 and 7.68% on b instances with an average number of broken time constraints of 4.5.

To the best of our knowledge, there has been no published paper that has considered deep learning methods to solve this specific type of problem. Thus we can't compare our approach results and considering the computation time needed to solve DARP it is also unfair to compare it with the exact solver. Although this current model is not able to guarantee to find feasible solutions for every instance, we can build on this paper to further improve these results since there is a lot of room for optimization that are still left to exploit.

10 Acknowledgment

We would like to thank Prof. Alahi Alexandre for his weekly supervision, this project couldn't have been possible without his insights and guidance.

References

[1] Jean-François Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[2] Timo Gschwind and Stefan Irnich. Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. *Transportation Science*, 49(2):335–354, 2015.

[3] Siddhartha Jain and Pascal Van Hentenryck. Large neighborhood search for dial-a-ride problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 400–413. Springer, 2011.

[4] Geraldo Regis Mauri, Luiz Antonio, and Nogueira Lorena. Customers' satisfaction in a dial-a-ride problem. *IEEE Intelligent Transportation Systems Magazine*, 1(3):6–14, 2009.

[5] Yannik Rist and Michael A Forbes. A new formulation for the dial-a-ride problem. *Transportation Science*, 55(5):1113–1135, 2021.

[6] Xavier Bresson and Thomas Laurent. The transformer network for the traveling salesman problem. *arXiv preprint arXiv:2103.03012*, 2021.

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

[8] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022.

Results on the "type a" standard instances

Instance	Obj	Method	Cost	Δ	N_{TW}	N_{RT}	Penalty	T time	I Time
a2-16	294.25	Supervised	317.33	7.84	0	0	0	1152.03	5.87
		Reinforcement	317.33	7.84	0	0	0	-	1.15
		10-Beam	317.33	7.84	0	0	0	-	172.78
		One for All	324.03	10.12	6	2	231.51	2902.71*	3.88*
a2-20	344.83	Supervised	404.19	17.21	0	2	13.07	1584.24	9.01
		Reinforcement	394.24	14.33	0	0	0	-	1.45
		10-Beam	396.86	15.09	0	2	8.93	-	255.96
		One for All	338.66	1.79	8	3	384.37	2902.71*	5.89*
a2-24	431.12	Supervised	476.80	10.59	1	4	46.46	2160.96	17.05
		Reinforcement	472.08	9.5	0	3	14.61	-	2.26
		10-Beam	472.27	9.55	0	3	16.61	-	358.48
		One for All	447.41	3.78*	9	6	402.43	2902.71*	6.09*
a3-24	344.17	Supervised	382.11	10.81	1	6	52.21	2419.01	15.05
		Reinforcement	381.08	10.51	2	4	64.47	-	4.28
		10-Beam	-	-	-	-	-	-	-
		One for All	367.95	6.70	9	8	395.03	2902.71*	6.35*
a3-36	583.19	Supervised	581.22	0.34	5	8	175.64	3979.82	6.67*
		Reinforcement	571.89	1.94	7	6	178.65	-	4.81
		10-Beam	575.44	1.33	4	7	165.66	-	555.30*
		One for All	527.73	9.51	12	8	399.91	2902.71*	7.36*
a4-32	485.50	Supervised	520.82	7.28	5	8	158.26	3632.00	6.07*
		Reinforcement	545.52	12.36	9	6	226.21	-	5.28
		10-Beam	-	-	-	-	-	-	-
		One for All	519.65	7.03	7	4	260.51	2902.71*	8.11*
a4-40	557.69	Supervised	643.87	15.45	10	9	392.36	2080.68*	6.22*
		Reinforcement	621.22	11.39	7	5	201.15	-	7.77
		10-Beam	630.68	13.09	3	7	85.26	-	691.88*
		One for All	593.76	6.47	14	9	356.13	2902.71*	8.85*
a4-48	668.82	Supervised	744.85	11.37	7	10	174.37	2902.71*	8.32*
		Reinforcement	773.98	15.72	9	8	271.21	-	10.88
		10-Beam	779.77	16.59	10	16	309.08	-	1003.97*
		One for All	744.85	11.37	7	10	174.37	2902.71*	8.32*

Table 3: All model results on the "type a" standard instances, the supervised model was used as a base model for the beam search decoding technique with beam width $B = 10$, the a4-48 supervised model was used for the One for All evaluations

Results on the "type b" standard instances

Instance	Obj	Method	Cost	Δ	N_{TW}	N_{RT}	Penalty	T time	I Time
b2-16	309.41	Supervised	312.81	1.1	2	0	55.61	1072.43	5.81
		Reinforcement	339.47	9.72	0	0	0	-	1.06
		10-Beam	311.80	0.77	2	0	48.41	-	169.26
		One for All	-	-	-	-	-	-	-
b2-20	332.64	Supervised	346.02	4.02	5	0	102.88	1523.72	10.53
		Reinforcement	346.02	4.02	5	0	102.88	-	2.17
		10-Beam	346.77	4.25	3	1	61.61	-	264.23
		One for All	-	-	-	-	-	-	-
b2-24	444.71	Supervised	479.56	7.84	4	0	84.47	2088.09	15.26
		Reinforcement	496.3	11.6	4	0	85.17	-	2.25
		10-Beam	491.19	10.45	4	0	84.47	-	373.58
		One for All	-	-	-	-	-	-	-
b3-24	394.51	Supervised	398.54	1.02	5	2	180.92	2278.98	12.85
		Reinforcement	409.26	3.74	6	2	178.31	2.76	
		10-Beam	383.05	2.91	3	1	72.90	-	346.13
		One for All	-	-	-	-	-	-	-
b3-36	603.79	Supervised	607.78	0.66	6	4	162.61	3765.35	5.56*
		Reinforcement	627.95	4.0	3	1	125.21	-	5.64
		10-Beam	619.32	2.57	3	4	69.69	-	549.06*
		One for All	-	-	-	-	-	-	-
b4-32	494.82	Supervised	565.08	14.20	5	2	207.88	1339.21*	4.84*
		Reinforcement	549.72	11.09	5	1	172.94	-	5.91
		10-Beam	-	-	-	-	-	-	-
		One for All	-	-	-	-	-	-	-
b4-40	-	Supervised	-	-	-	-	-	-	-
		Reinforcement	-	-	-	-	-	-	-
		10-Beam	-	-	-	-	-	-	-
		One for All	-	-	-	-	-	-	-
b4-48	-	Supervised	-	-	-	-	-	-	-
		Reinforcement	-	-	-	-	-	-	-
		10-Beam	-	-	-	-	-	-	-
		One for All	-	-	-	-	-	-	-

Table 4: All model results on the "type b" standard instances, the supervised model was used as a base model for the beam search decoding technique with beam width $B = 10$, the b4-48 supervised model was used for the One for All evaluations