FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Vehicle Routing Problem with Time Windows solved via Machine Learning and Optimization Heuristics
Student:	Bc. Adam Zvada
Supervisor:	doc. Ing. Pavel Kordík, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

The thesis will follow these steps:

- Research state of the art methods for solving VRPTW using optimization heuristics
- Research a various solution which leverages machine learning techniques for solving VRPTW
- Implement selected method for VRPTW using optimization heuristics
- Implement selected method for VRPTW using machine learning techniques
- Benchmark implemented methods on a public dataset and discuss their results in detail.

Electronically approved by Ing. Karel Klouda, Ph.D. on 11 February 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Vehicle Routing Problem with Time Windows solved via Machine Learning and Optimization Heuristics

Bc. Adam Zvada

Department of Applied Mathematics
Supervisor: doc. Ing. Pavel Kordík, Ph.D.

May 6, 2021

Acknowledgements

This is giving me the remarkable opportunity to express many thanks to the Big Bang for all its kindness in creating our space-time just right.

Great thanks also to my supervisor doc. Ing. Pavel Kordík, Ph.D. for all his help, time and advice.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on May 6, 2021 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Zvada, Adam. *Vehicle Routing Problem with Time Windows solved via Machine Learning and Optimization Heuristics*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Pro řešení vehicle routing problému byly navrženy nové přístupy v oblasti strojového učení, ale téměř žádná pozornost nebyla věnována variantě vehicle routing problému s časovými okny s mírným omezením (VRPTW). I přesto, že tato varianta je nutná řešit v každém produkčním řešení plánovacího logistického systému.

Tato práce navrhuje novou metodu řešení VRPTW pomocí hlubokého posilovaného učení. Model je postaven na architektuře Transformer využívající Graph Attention Network pro vložení vstupní instance. Model používá nově navrženou funkci odměny, která zahrnuje omezení časových oken. Práce také zkoumá další metaheuristické metody pro řešení VRPTW, které slouží ke vyhodnocení výsledného modelu.

Výsledkem této práce je end-to-end model hlubokého učení, který řeší VRPTW, který ale stále předčí metaheuristiké metody.

Klíčová slova vehicle routing problém, časová okna, vrptw, hluboké posilované učení, umělá inteligence, strojové učení, transformer, graph attention network

Abstract

A novel approaches in the field of machine learning has been proposed to solve the vehicle routing problem, but yet a variant of vehicle routing with soft constrained time windows has received almost no attention. Even though it is a must for any production ready logistics planner.

This thesis proposes a new method for solving a vehicle routing problem with soft constrained time windows (VRPTW) using deep reinforcement learning. The model is built upon Transformer architecture utilizing Graph Attention Network for embedding the input instance. The model is using the proposed reward function that incorporates the time window constraint. The thesis also explores other metaheuristics methods for solving VRPTW, which is used to benchmark the model performance.

The result of this thesis is end-to-end deep learning model solving VRPTW but it is still outperformed by metaheuristics solvers.

Keywords vehicle routing problem, time windows, vrptw, deep reinforcment learning, artificial intelligence, machine learning, transformer, graph attention network

Contents

List of Figures

List of Tables

Introduction

The VRP is one of the most extensively studied combinatorial problems. It is easy to define but very difficult to solve[8]. The reason VRP is attracting many researchers is the fact that finding a near-optimal solution in a reasonable time would have a great impact on many industries, especially in the domain of transportation and logistics.

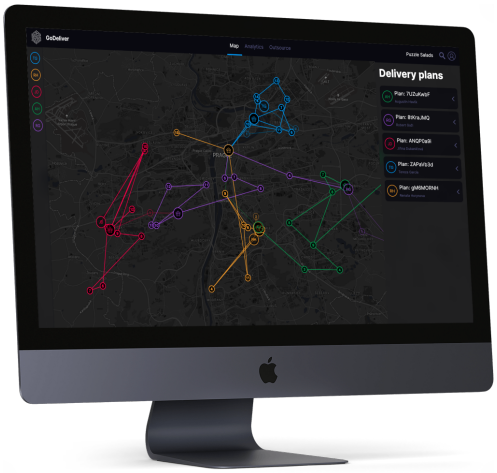

Figure 0.1: GoDeliver dashboard visualizing solution for an instance of vehicle routing problem with time windows.

Motivation

The paradigm shift in logistics business models towards instant gratification of customers are pushing the planning systems to be flexible and dynamic. The environment is constantly changing and planning systems have to update or entirely replan the instance in a short amount of time but maintaining

the best delivery efficiency. Having a powerful planning system results in a dramatic reduction of delivery expanses.

Challenges

In the real world, the general VRP problem is not enough to solve the business-related problems. VRP has multiple variants adding various constraints such as capacity, demand or time windows for given set of customers. The VRPTW is main focus of this thesis and we will be looking at some novel approaches how to solve it with artificial intelligence (AI).

Assumptions

We expect that leveraging AI or machine learning (ML) techniques to solve VRP would lead in a drastic reduction of computational time for solving given instance of VRP. Moreover, the time complexity would not be exponentially increasing with the problem size. The trade-off lays in the required time to allow AI to train and learn how to solve the problem of vehicle routing.

Thesis structure

The rest of this thesis is organized as follows:

- Chapter 1 presents a formal introduction to Vehicle Routing Problem.
- Chapter 2 provides an advanced theoretical background.
- Chapter 3 describes solutions of VRPTW for optimization heuristics.
- Chapter 5 describes how a delivery planning system works.
- Chapter 4 describes the proposed method for solving VRPTW via deep learning.
- Chapter 6 evaluates and benchmarks the proposed deep learning model.

Chapter 1

Introduction to Vehicle Routing Problem

The problem objective of VRP is simply finding the shortest route for multiple vehicles to serve all the given set of customers. The shortest route can be differently interpreted based on your minimalization criteria, e.g, traveled distance, time, or a combination of both. It was first proposed by Dantzig and Ramser [9] in 1959, and since then researchers are coming up with different approaches how to solve the problem.

1.1 Vehicle Routing Problem Definition

The general VRP can be defined as a problem in a complete graph $G = (V, E)$ of finding the optimal permutation $\pi_l = (\pi_0, \cdots, \pi_m)$ of nodes V all starting from a node v_0 for given number of paths k which results in minimal traversal cost where $\forall v \in (V \setminus v_0)$ are visited only once. VRP is generalization of traveling salesman problem (TSP) which only has one path.

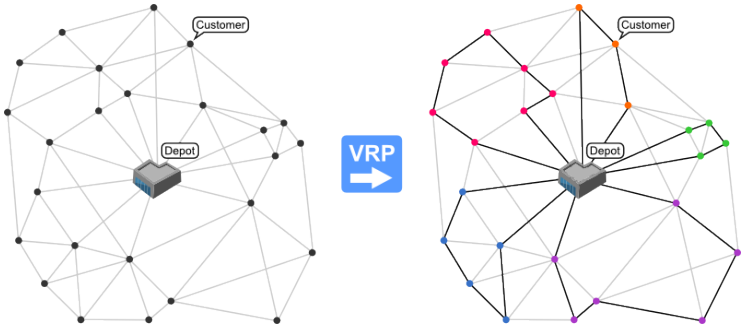

Figure 1.1: Intuitive view of VRP instance on left and proposed solution for 5 vehicles on the right [1]

VRPs are classified as NP-Hard problems which was proved by Lenstra and Kan [8]. It means that in the worst case, adding new nodes, i.e., customers results in an exponential increase of computational complexity.

1.1.1 VRP Notation

Let's introduce our used notation and its real-world interpretation.

- $G = (V, E)$ is a complete undirected graph
  - Network of routes
- v_0 is the initial node
  - A depot
- $V' = (v_1, \cdots, v_n)$ nodes expect the initial node
  - Geographically scattered location of customers
- $E = \{(v_i, v_j) | v_i, v_j \in V, i \neq j\}$ with associated weight as a cost $c : E \rightarrow \mathbb{N}^+$
  - A single route between two locations with associated cost, e.g., distance.
- C is a matrix of edge weights indexed by nodes. c_{ij} where $i, j \in V$
  - Matrix of costs between customers
- $R_i \subset V$ is a path that starts and ends at v_0. $(r_0 = v_0 \wedge r_{|R_i|} = v_0)$
  - Route visit a subset of customers starting and ending at the depot, it can be referred to it as a delivery plan.
- k number of paths
  - Number of vehicles
- $R = R_1, \cdots, R_k$ is a set of paths
  - All routes (delivery plans) for a given instance of VRP.
- $\pi = (\pi_1, \cdots, \pi_k)$ solution for a given instance of VRP.
  - Customer locations in visiting order for multiple vehicles.

Feasibility of VRP solution for VRP of routes R is feasible only if each node V_1 is visited exactly once.

The cost of route R_i which we aim to minimize is the sum of its weights (costs). If we operate in Euclidean space, then it is L2 norm of route locations.

$$C(R_i) := \sum_{k=0}^{|R_i|} c_{r_k r_k+1} \tag{1.1}$$

The cost of VRP solution is the sum of route costs.

$$C(R) := \sum_{i=1}^{|R|} C(R_i) \tag{1.2}$$

1.2 Vehicle Routing Flavors

Our modern world heavily relies on complex logistics networks. It requires to synchronize multimodal planning to ship your goods from one side of the world to your doorstep. In order to achieve this, multiple variants and flavours of VRP had to be studied and implemented in the real world use cases. It goes from ordinary variants like measuring the capacity of cars to a more niche problem like eVRP where vehicles are required to make stops to recharge.

All the flavours of VRP can be mutually combined, which is usually the main area of research.

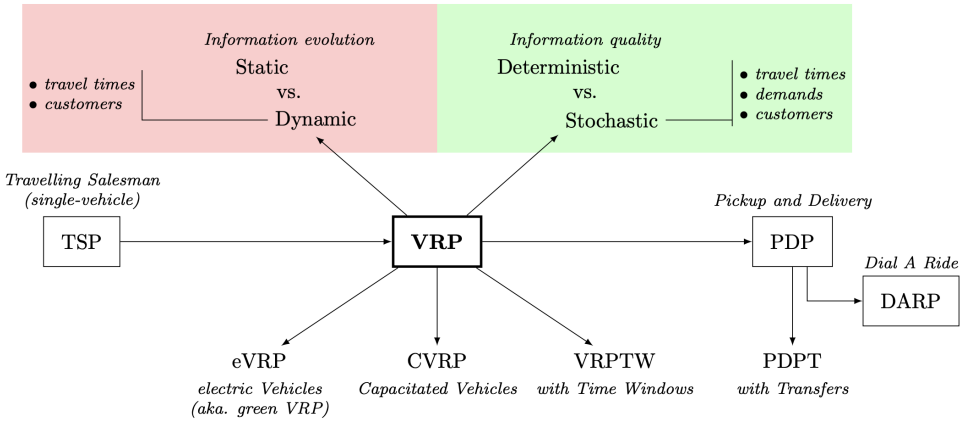

Figure 1.2: Taxonomy of VRPs [2]

The sections below are describing each flavour shown in 1.2.

1.2.1 Capacitated Vehicle Routing Problem

The capacitated vehicle routing problem (CVRP) extends the regular CVRP in introducing a capacity element for each customer. In the literature, it is

sometimes referred to as a demand. The customer's demand is $d \in \mathbb{N}^+$ which may represent capacity in the form of weight, size but also in some abstract concepts such as a basket of apples. Additionally, each vehicle has a predefined capacity $Q > 0$.

The CVRP extends the solution feasibility formula by the following capacity constrain.

$$q(R) := \sum_{i \in R} d_i \leq Q \tag{1.3}$$

If the vehicle capacity of the fleet stays the same, we are dealing with CVRP with homogeneous fleet. A fleet with varying capacity for each vehicle is a heterogeneous fleet.

1.2.2 Vehicle Routing Problem with Time Windows

The VRPTW [10] extends the regular VRP by time constraint for each customer. Customers have assigned time window interval $[e_i, l_i]$ where $e_l i < l_i$. The time interval is the request within a vehicle is supposed to visit the node.

The time window can be either implemented as a hard constraint or a soft constraint. Hard constraint forces the vehicle to visit the node, i.e., the customer either in the given time interval or the solution is not feasible. Soft constrains are not strictly enforcing the vehicle to visit the customer, but they introduce a penalty for a violated interval barring a penalty cost. The penalty becomes a part of the cost function which VRP aims to minimize.

In this thesis, we will be focusing on soft constraints for time windows since it is a better reflection of real world use cases. Most businesses allow couriers to arrive late or early, but these types of arrivals are supposed to be minimized.

1.2.3 Pick and Deliver

The Pick and Deliver (PDP) extends the regular VRP by pairing pick and drop with precedence relationships, in which a pickup point must precede the paired delivery point. This flavour of VRP is one of the most complex and even challenging for conventional methods like optimization heuristics algorithms.

The feasibility of a PDP solution is checking whether all delivery points have preceded pickup point.

1.2.4 Static vs. Dynamic

When solving the vehicle routing model, usually we assume that all the input data are static and known with certainty. However, this is not the case in real-life applications where data such as customer demand or travel time are often incomplete or not precise during the planning phase, they are only gradually revealed and specified.

Static VRP does not assume that the input data could be subject to change. The **dynamic VRP** is aware about the information evolution[11] and its goal is to obtain a robust routing planner that will be able to solve already seen instances with subject to small changes without the need of recalculating the whole instance again. This is called a priori optimization, after solving a given instance of a combinatorial optimization problem, it becomes necessary to repeatedly solve many other instances with a small variation from the original instance but without reconsideration of the entire problem [12].

In this thesis, the VRP based on AI could be a great candidate for dynamic VRP even though, the entire instance is being recalculated. The reason is that the problem solution is calculated in a seconds instead of minutes and the AI technically already seen the instance in some variation during the training phase.

Dynamic VRP can be achieved with enough robust architecture around the core planner and periodically recalculating the instance with newly revealed information. The planner needs to take its previous solution as an input so the part of the problem does not need to be recalculated. This approach tends to be more exploitative since it is finding a solution in a predefined search space. It would benefit from introducing an explorative element which would diversify the search and could find better cost in a different local optimum.

1.2.5 Deterministic vs. Stochastic

Psaraftis [11] stated that there are two important dimensions of input data, the information evolution which is used in dynamic VRP and quality of information for stochastic VRP. The majority of studied VRP models are under the assumption that all the information necessary to formulate the problems is known and readily available. This is true but only for the deterministic settings [13].

A **VRP is stochastic** [14] when some of its data behave as random variables, and the routes must be defined before the values of these random variables become known. Based on the probability distribution of the random variables, we may extract some hidden information and use it to our advantage in the planning process. The newly created plans will have incorporated stochastic information and the routing decisions may lead to different decisions because of the stochastic information being part of the cost function.

A specific real-life example of stochastic VRP would be if we consider an electric fleet of shared mobility vehicles and treat the locations of Blinkee electric scooters as random variables. Based on the probability distribution of Blinkee scooter, we may predict the time and location where a courier will transfer to a new fully charged Blinkee scooter. This action will be incorporated into the planned routes.

In contrast, **deterministic VRP** has no random information which could be leveraged before the execution of routes and all the given information are

known with certainty. In this thesis, we are focusing on deterministic VRP.

1.2.6 Other Flavours

Dial-a-Ride (DARP) proposed by Wilson et al. [15] in 1971 is a special case of dynamic VRP with pick and deliver. Passengers request a ride at a specified origin and drop location with an optional time window.

Split Delivery VRP [16] is a variant where customers are allowed to be visited more than once. This can be convenient for deliveries of large capacity or stocking fulfillment centers.

Multi Depot VRP is a simplification of the vehicle routing problem with pick and deliver, where pick can happen only on predefined depot locations. This simplification of pickup location is making the problems less complex then vehicle routing problem with pick and deliver (VRPPD).

1.3 VRP in a Real-world

Consumer habits have been shifting towards online and the pandemic situation only accelerated this process. Delivery option is nowadays taken for granted and consumers are demanding a perfect delivery experience. In 2020, there has been shipped over 5.5 billion packages around the world [17].

Solving various flavors of VRP efficiently in a reasonable time plays a crucial role for multiple businesses. For example, urban logistics is an essential part of the delivery process, not only it is the last part of the delivery chain, but frequently the courier interacts with the customer and delivery on time with proper ETA prediction is a must. It is also the most expansive part which makes up about 53% of shipment's total cost[17]. Urban logistics by large benefits from a better and more optimized VRP which increase the delivery efficiency and reduces the delivery cost.

At GoDeliver, we are building an autonomous last-mile delivery system and at the core is a planning system which is solving various VRP. The flavour which we are focusing on is Dynamic Capacitated Vehicle Routing Problem with Time Windows and Pick and Delivery (CVRPDPTW). GoDeliver typical use case is on-demand food delivery with multiple depots (pick and deliver), this means that the system has to be dynamic and flexible because a new customers are ordering stochastically for a chosen time window. Another our common use case shown in 1.3 is grocery delivery with multiple depots, time windows, and capacity for customers.

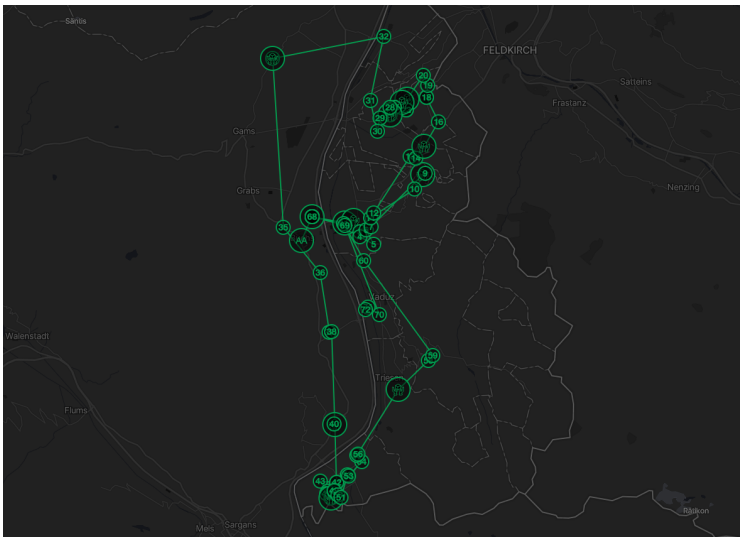

Figure 1.3: Grocery delivery planning with multiple depots from the GoDeliver system

Chapter 2

Theoretical Background

In this chapter, we will be covering the advanced theoretical background to fully understand the solved task of VRPTW using ML.

2.1 Reinforcement Learning

ML can be divided with a little simplification into three categories; supervised learning, unsupervised learning, and reinforcement learning. Supervised learning is the most common where the model is learned from the provided labeled data. Unsupervised learning, on the other hand, is about finding a hidden patterns in a collection of data with no labels. Finally, reinforcement learning has no labeled data but learns by interacting with the environment and getting feedback in the form of rewards as shown in Figure 2.1.

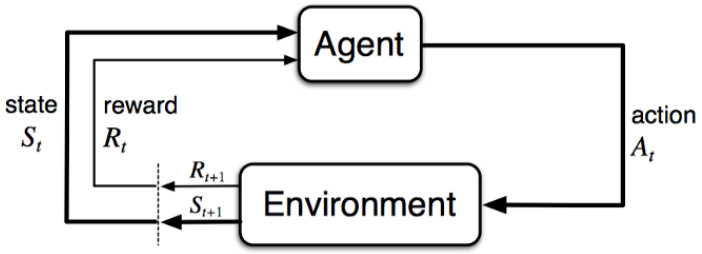

Figure 2.1: Agent feedback loop[3]

The Reinforcement Learning mimics the learning process of humans beings. By experiencing the world and accumulating knowledge, we are learning how to handle novel situations. reinforcement learning (RL) system consists of agent in observed state s_t, the agent interacts with the environment via its actions a_t at discrete time steps t and receives a reward r_{t+1} for given action. The action moves the agent into a new state s_{t+1}. The goal of the agent is to

learn a policy π which chooses the action that maximizes the agent's rewards based on the environment [3].

2.1.1 State and Action Value Functions

Transition to a new state gives us a reward and to maximize it, we need a way to quantify how good a state is. A state-value function $V_\pi(s)$ predicts a future reward for a given state when following the policy π [3].

$$V_\pi(s) = \mathbb{E}[G_t|S_t = s] \tag{2.1}$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

The equation 2.2 calculates G_t, all future rewards, sometimes called as *return* [3]. The $\gamma \in [0, 1]$is a discount factor and penalizes the rewards in the future, incorporating the possible uncertainty and variance of the future rewards.

We will also define action-value $Q_\pi(s, a)$ which is for a similar purpose as state-value function but predicts the reward for action and state following the policy π.

$$Q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a] \tag{2.3}$$

The decomposition of state-value and action-value function replays on Bellman equations [18]. The decomposition of state-value function is

$$V_\pi(s) = \mathbb{E}[G_t|S_t = s] \tag{2.4}$$

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s] \tag{2.5}$$

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots)|S_t = s] \tag{2.6}$$

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \tag{2.7}$$

$$V_\pi(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s] \tag{2.8}$$

Similarly, this method is applicable to action-value function,

$$Q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})|S_t = s, A_t = a] \tag{2.9}$$

$$Q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma \mathop{\mathbb{E}}_{a\sim\pi} Q(S_{t+1}, a)|S_t = s, A_t = a] \tag{2.10}$$

2.1.2 Policy Gradients

Policy Gradient [19] is a method for solving the reinforcement learning problem and learning the policy that maximizes the rewards. We define a set of parameters θ that directly models the policy, $\pi_\theta(a|s)$.

To optimize θ for the best reward, we define an objective function [19] as

$$J(\theta) = \sum_{s \in S} d_{\pi_\theta}(s) V_{\pi_\theta}(s) \tag{2.11}$$

where $d_{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ, the probability of ending in a given state [20].

$$d_{\pi_\theta} = \lim_{t \longrightarrow \infty} P(S_t = s | s_0, \pi_\theta) \tag{2.12}$$

The objective function $J(\theta)$ optimizes the θ parameters via gradient ascent [21].

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \tag{2.13}$$

However, computing $\nabla J(\theta)$ is tricky because it depends on the action selection and the stationary distribution of states [22]. Policy gradient can be simplified using Policy Gradient Theorem by Sutton et al. [19].

The proof of policy gradient theorem is quite long and complicated, but you may go through it in this article [22] which is inspired by Sutton and Barto [3]. Policy gradient is simplified to the form as

$$\nabla J(\theta) = \mathbb{E}[\nabla \ln \pi(a|s, \theta) Q_\theta(s, a)] \tag{2.14}$$

2.1.3 REINFORCE

REINFORCE algorithm, proposed by Williams [23] in 1992, is a policy gradient method to update the policy parameter θ.

Let us define the additional terms required by the REINFORCE algorithm. We define a trajectory τ which is a sequence of states, actions, and rewards. Episode is a trajectory which ends at the terminal state S_t.

$$\tau = (S_0, A_0, R_0, S_1, A_1, R_1, \cdots) \tag{2.15}$$

REINFORCE algorithm computes the policy gradient as follows

$$\nabla J(\theta) = \mathbb{E}[G_t \nabla \ln \pi(A_t|S_t, \theta))] \tag{2.16}$$

It is a simplification of a regular policy gradient because $Q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$ and in REINFORCE algorithm we rely on a full trajectory where

we can estimate G_t based on Monte-Carlo method which is describe in this article [3].

Algorithm 1: REINFORCE algorithm

Result: Updated θ that maximises reward
Initialize *theta* at random;
Generate one episode $S_0, A_0, R_0, \cdots, S_T$;
for $t = 1, 2, \cdots, T$ **do**
    Estimate the the return G_t since the time step t;
    $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla \ln \pi(A_t|S_t, \theta)$
end

In the REINFORCE algorithm, the estimated gradient is highly effected by variance. A technique called baseline $b(S_t)$ is common to be used which subtracts a baseline from the estimated G_t to reduce the variance [24]. The baseline function can be in many forms, but the most common one is to calculate the advantage function $A(s, a) = Q(s, a) - V(s)$ and use it to be subtracted from the gradient.

2.2 Attention

Attention mechanism was first proposed by Bahdanau et al.[25] in 2014 with a problem to help memorize long source sentences in neural machine translation. Not long after that, this concept gave birth to Transformers which dramatically improved many domains, especially Natural Language Processing and brought state-of-the-art results [26].

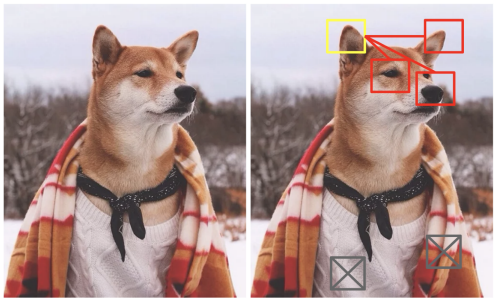

Figure 2.2: A Shiba Inu and what is caughting the network's attention [4], photo credit by @mensweardog.

For humans, visual attention is allowing us to focus on certain regions as visualized on Figure 2.2. Attention mechanism decides on which part of the given source should pay attention to.

2.2.1 Transformer

The Transformer neural network architecture proposed by Vaswani et. al [5] was one of the major breakthroughs in the field of Natural Language processes (NLP). Transformer is getting rid of recurrence [27] in favor of the attention mechanism which allows global dependencies between input and output.

The Transformer architecture is based on encoder-decoder structure as shown on Figure 2.3. Encoder maps the input sequence x to continues representation z which is taken by decoder and decodes it to output sequence y one element at a time. The model is auto-regressive, it takes into account the previously generated output as additional input.

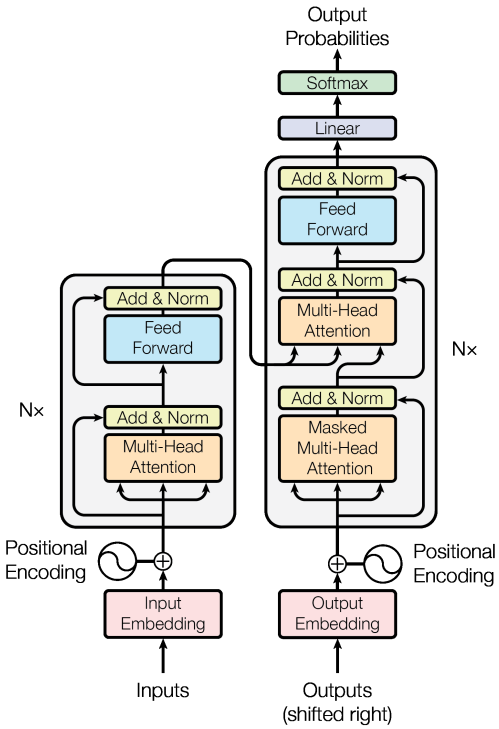

Figure 2.3: The Transformer architecture, encoder on the left and decoder on the right [5].

2.2.1.1 Encoder

The encoder on Figure 2.3 has N identical layers (Vaswani et al. set $N = 6$ [5]). Each layer has two sublayers, the first is multi-head attention 2.2.1.3, and the second is a simple fully connected feed-forward network. Each of the sublayers has a residual connection [28] that sums the input and output of the

sublayer and normalize it [29].

$$OutputSubLayer = Norm(x + SubLayer(x)) \tag{2.17}$$

The residual connections are similar to the skip connection which propagates input of the sublayer to the output which helps to avoid exploding or vanishing gradient.

2.2.1.2 Decoder

The decoder on Figure 2.3 is also built from N identical layers and with similar sublayers as the encoder. The first sublayer is masked multi-head attention 2.2.1.3 also with residual connection. The masking mechanism is only there to hide the future information because the Transformers are processing the input one by one. The next two sublayers of the decoder are same as the encoder with the only difference that multi-head attention 2.2.1.3 receives part of the input from the encoder.

2.2.1.3 Multi-Head Attention

The major component of Transformers is Multi-Head Attention (MHA). It runs the input through an attention mechanism h times in parallel. The independent attention outputs are then concatenated and transformed into the expected dimension as shown in Figure 2.4 [4].

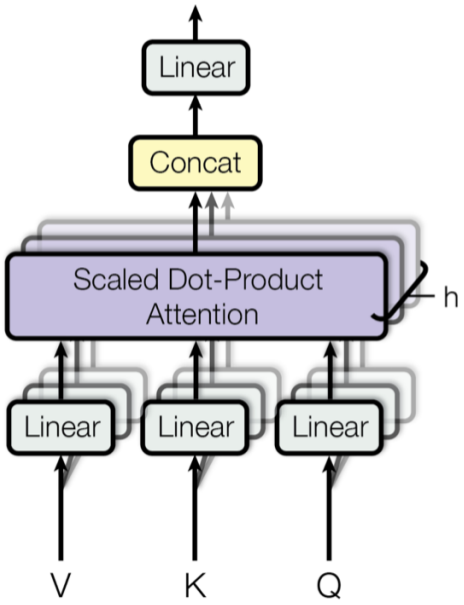

Figure 2.4: Multi-Head Attention component [4]

The Multi-Head Attention takes the embedded input and split the input copy to Key K, Value V and Query Q. They all have the same dimension of input sequence length and embedded vector size d_{model} (Vaswani et al. set $d_{\text{model}} = 512$ [5]).

The matrices K, V, and Q get multiplied for each head $i \in (1, \cdots, \text{h})$ by trainable parameters W_i^K, W_i^V, and W_i^Q, respectively. Then it performs scaled dot-product attention as shown in equations 2.18 and 2.19

$$\text{head}_i = \text{ScaledDotAttention}(QW_i^Q, KW_i^K, VW_i^Q) \tag{2.18}$$

$$\text{ScaledDotAttention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{n}})V \tag{2.19}$$

According to the paper[5], the scaled dot product attention allows the model to jointly attend to information from different representation subspaces at different positions.

Finally, we concatenate all head outputs $head_i$ for $i \in (1, \cdots, \text{h})$ from scaled dot product attention and perform linear transformation by trainable parameter W_0 of Multi-head Attetntion (MHA).

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, \cdots, \text{head}_h)W_0 \tag{2.20}$$

2.2.2 Graph Attention Network

To understand the model used for solving VRPTW we need a basic understanding of Graph Attention Network (GAT), first proposed by Veličković et al. [6] in 2017.

It is a neural network architecture capable of operating on graph-structured data. GAT extend the work on Graph Convolution Networks [30] which has a problem with generalizability because they are structure-dependent [31]. GAT address the shortcoming of Graph Convolution Networks (GCN) by leveraging masked self-attentional layers 2.2.

GAT is a single layer (possibly can be stacked) which takes a set of nodes features $h = \{h_0, h_1, \cdots, h_N\}, h_i \in \mathbb{R}^F$ where N is the number of nodes and F is the number of features for a node. The GAT layer assigning different importance to each node through the attention coefficients.

$$e_{ij}^l = a^l(W^l h_i^l, W^l h_j^l) \tag{2.21}$$

The equation 2.21 is a linear transformation of node features and weight matrix W^l which is used for calculating the attention coefficients via learnable parameter a^l. It is gathering information about the importance of node j's features to node i.

$$\alpha_{ij}^l = \text{softmax}_j(e_{ij}^l) = \frac{exp(e_{ij}^l)}{\sum_{k \in N_i} exp(e_{ik}^l} \tag{2.22}$$

The graph structural data are represented in attention via equation 2.22 which is computed only for all neighbour nodes N_i of node i. The softmax function is used to make the attention scores comparable across different nodes.

$$h_i^{l+1} = \sigma(\sum_{j \in N_i} \alpha_{ij}^l W^l h_i^l) \tag{2.23}$$

The equation 2.23 is aggregating together all attention embeddings of all neighbour nodes. If we perform multi-head attention on the final layer of GAT then computation of the final high level feature embedding is as follows

$$h_i^{l+1} = \sigma(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in N_i} \alpha_{ij}^l W^k h_i^l) \tag{2.24}$$

The aggregation process of a multi-head graph attentional layer for a given node is illustrated on Figure 2.5.

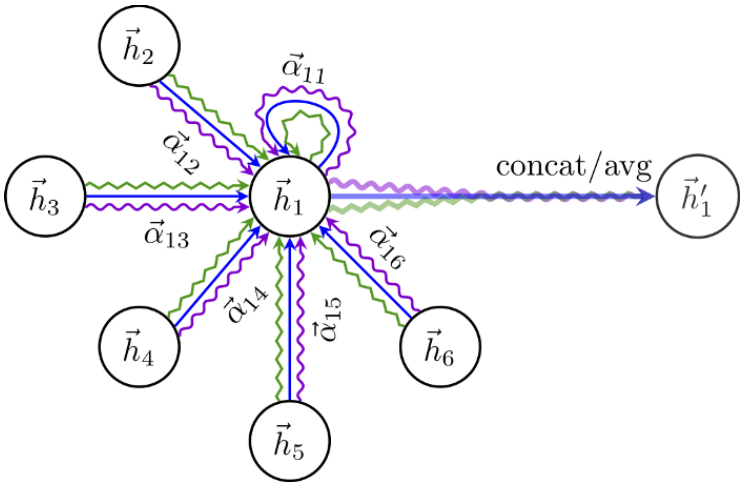

Figure 2.5: Multi-head attention with $K = 3$ heads [6].

Chapter 3

VRPTW via Optimization

In this chapter, we review, multiple approaches for solving VRPTW via optimization techniques with the objective to find suboptimal solutions in a reasonable time. The real-life use cases of VRP are demanding to quickly find a good enough solution for large instances and this is achieved by applying handcrafted heuristics.

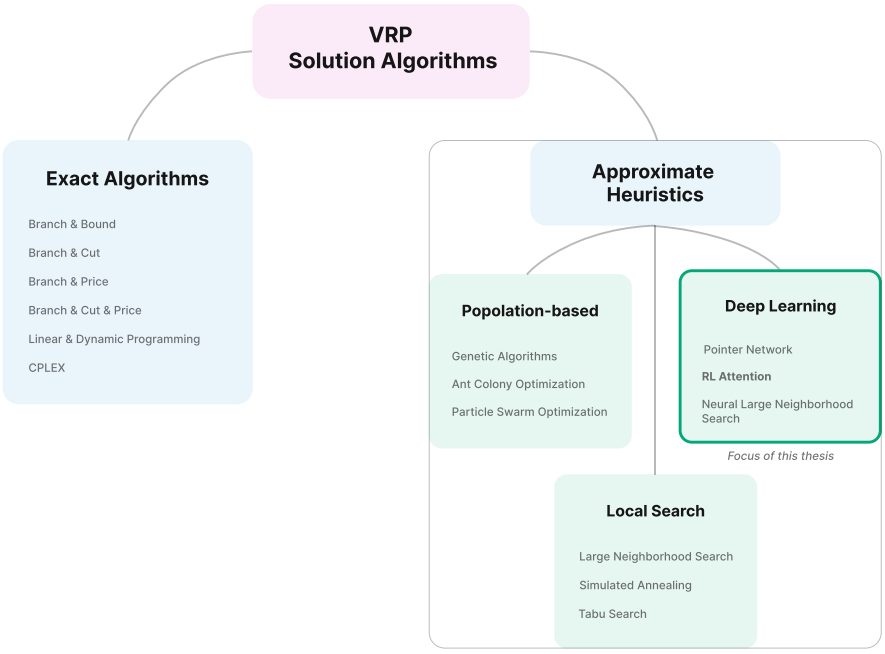

Figure 3.1: Family of algorithms for solving VRP

In the field of operations research, approximate heuristics may be divided

into *constructive heuristics* and *metaheuristics*. Constructive heuristics are used to provide an initial solution in a matter of seconds, which is later optimized. Metaheuristics is end-to-end algorithm that takes a problem instance and calculates its suboptimal solution. Metaheuristics can be further divided into *local search method* which explores the search space by iterativly moving to a better solution, or *population-based methods* that generate a set of solutions that are continually evolving [13].

3.1 Insertion Heuristics

Insertion heuristic is a constructive heuristic first proposed by Jaw et. al [32] in 1986 and later that year extended by Solomon et al. [33] to supporting time windows.

The insertion heuristic works by gradually iterating through all the nodes considered to be planned. For each of the nodes, the algorithm tries to insert it to all possible positions of all plans, then it picks the best insertion based on the calculated penalty. By repeating this process for all the nodes, a fully feasible solution is constructed.

Algorithm 2: Insertion Heuristic

```
for x ∈ X do
    bestPenalty = MAX;
    for r ∈ R do
        let l be a length of route r;
        for i ∈ {1, ··· , l + 1} do
            for j ∈ {i + 1, ··· , l + 2)} do
                Try to insert on π^r_ij the node x;
                Calculate newPenalty of the insert π^r_ij;
                if bestPenalty > newPenalty then
                    bestInsert ← (i, j);
                    bestPenalty ← newPenalty;
                end
            end
        end
        Perform best insert π^r_bestInsert the node x;
    end
end
return π;
```

Insertion heuristic is mainly driven by the cost function that returns the penalty for a given set of plans. The cost function has to consider the distance of a plan but also include the time window slack due to the insertion [34].

It very common with dynamic VRP 1.2.5 to continuously recalculate delivery plans based on triggers such as a new delivery order. The advantage of

this heuristic is that it can leverage on a given previous solution and does not need to construct it from scratch.

3.2 Google OR-Tools

We have noticed that many of the production solutions in the field of logistics use some type of optimization framework. The framework allows to describe the routing problem using their specific notation and their optimization library finds its solution. They usually implement variants of local search algorithms. However, their are fairly flexible and they serve as a great reference point for any benchmarking.

In this thesis, we use Google OR-Tools[1] an open source tool developed to solve optimization problems in vehicle routing, network flows, integer and linear programming, and constraint programming [35]. We have chosen this optimization framework because the operations research community uses OR-Tools as a main benchmark and it supports soft constraint programming.

OR-Tools works by building a graph where the distance callback function assigns a value to graph transitions called arc cost. In the use case of solving VRPTW, the arc cost is supposed to be the traveled duration between two given locations (nodes).

OR-Tools implements a cost function in the form of dimensions representing quantities accumulated at nodes along the vehicle's route. Solving VRPTW via OR-Tools requires to have extra dimensions for time windows that accumulates early and late arrivals. OR-Tools are minimizing the dimensions values by using a chosen metaheuristic.

OR-Tools implements many well-known metaheuristics such as

- AUTOMATIC Lets the solver select the best metaheuristics.
- GREEDY_DESCENT Accepts only cost-reducing solutions using local search of neighbors until a local minimum is reached.
- GUIDED_LOCAL_SEARCH Uses guided local search to escape local minima [36].
- SIMULATED_ANNEALING Uses simulated annealing to escape local minima [37].
- TABU_SEARCH Uses tabu search to escape local minima [38].
- OBJECTIVE_TABU_SEARCH Uses tabu search on the objective value of a solution to escape local minima [39].

[1] `https://developers.google.com/optimization`

3.3 Large Neighborhood Search

Heuristics based on large neighborhood search have shown superior results in solving a wide variety of routing problems. Large neighborhood search was first introduced by Shaw [40] in 1997.

Large neighborhood search is an iterative algorithm that gradually improves its solution by exploring its neighborhoods. The neighborhoods are defined by applying *destroy* and *repair* operator. The destroy operator removes a random part of the solution such that different parts are destroyed in each iteration. The repair operator takes the partial solution and rebuilds into a fully feasible solution [41].

In the implementing of large neighborhood search, the most important is to pick the proper degree of destruction for the destroy operator. If it destroys a small part of the solution, then it leads to ineffective exploration of the search space. In the opposite, when destroying large parts of the solution, the algorithm will keep re-optimizaing, which yields poor quality solutions with higher complexity [41]. The degree of destruction can be either chosen randomly or it can be gradually increased during the execution.

Algorithm 3: Large Neighborhood Search

```
Initialize a feasible solution x_b;
while stopping criteria is met do
    x_t = repair(destroy(x_b));
    if accept(x_t, x)) then
        Accept a new solution x = x_t;
    end
    if cost(x_t) > cost(x_b) then
        Keep the best x_b = x_t;
    end
end
return x_b;
```

The original large neighborhood search algorithm only accepted a superior solution based on the cost function. To achieve better exploration, a new acceptance criterium was used, inspired by the algorithm of simulated annealing [42]. The algorithm accepts the solution x based on probability $e^{(c(x_t)-c(x))/T}$ where T is the parameter for temperature. The temperature is gradually decreasing resulting in accepting fewer deteriorating solutions.

3.3.1 Adaptive Large Neighborhood Search

Adaptive Large Neighborhood Search is an extension of Large Neighborhood Search (LNS) that was proposed by Ropke et. al [43] in 2006.

Adaptive Large Neighborhood Search (ALNS) supports multiple destroy operators and repair methods and adds a component of diversification strategy.

Repair operators are selected based on their past performance during the search and usually by employing a roulette wheel selection process with an adaptive weight adjusting mechanism [43].

3.4 Ant Colony Optimization

The other category of metaheuristics are population-based methods, which take their inspiration from natural concepts such as the evolution of species or the behavior of insects. However, all successful population-based heuristics rely on local search methods to drive the search towards promising areas and to avoid local optima. As a result, the majority of population-based algorithms are naturally hybrid [13].

Ant colony optimization was first applied on VRP by Reimann et. al [44] in 2004, and successfully outperformed many existing solutions. It is inspired by the pheromone mechanism used by ants for coordination. where each ant simulates a solution by traversing the graph along its edges and accumulates the pheromones.

Ant Colony Optimization mainly consists of the iteration of four steps [44]:

- Generation of solutions according pheromone information.
- Application of a local search to the ant's solution.
- Update of the pheromone information.
- Augmentation of the attractiveness list.

The algorithm generates solutions according to pheromone information, which represents how good is the combination of two nodes. The solution is generated by running a decision step for the ants in which they decide where to move based on the pheromone information. Each ant generates k feasible moves and one is picked based on the attractiveness probability [44], evaporating pheromone. The construction process is stopped when no more feasible combinations are possible. After the ants have constructed their solutions, it is improved by applying a local search algorithm like LNS 3.3. Than the pheromone trails are updated based on the achieved local optimum [45] and a new attractiveness values are created based on pheromone information.

Chapter 4

VRPTW via AI

In this chapter, we will describe our end-to-end deep learning method for solving VRPTW.

Machine learning and artificial intelligence have been replacing many hand-engineered algorithms and providing state-of-the-art results. In recent years, reinforcement learning 2.1 and advances in attention models 2.2 has shown great promise to disrupt the field of heuristics algorithms [46, 7, 47]. Heuristics algorithms [48] are incomplete methods that can compute solutions efficiently, but are not able to prove the optimality of a solution. Most of the business challenges do not require the most optimal exact solution [49] but focus on approximation of the optimal solution in a reasonable time.

4.1 Related Work

Regarding the research of solving the VRP, researchers have been mainly concentrating on designing hand-crafted metaheurictis via optimization. However, the great adoption of deep learning is starting to catch up with the field of operations research.

The first relatively successful deep learning model for solving general VRP was proposed by Vinyals et al. [50] in the year of 2015 by introducing Pointer Networks. The model uses attention to output a permutation of the input and the model was trained in the supervised manner by example solutions. In the next year, Bello et al. [51] extended the model of Pointer Networks by adopting the Actor-Critic algorithm that introduced RL and the model was trained on the training samples and did not require labelled data anymore. The reward function was a simple Euclidean length of the routes. The network showed improved performance on larger instances of 50 nodes over the predecessor model using supervised learning. In 2018, Nazari et al. [52] simplified the model of RL-based Pointer Network by omitting the recurrent neural network encoder and replaced it with embedding to D-dimensional vector space. The recurrent neural network is not necessary because the inputs of delivery nodes

are not dependent on order [52]. There was no deterioration in performance and the model also supported the constraint for solving split delivery 1.2.6.

In 2018, Kool et al. [7] proposed a new approach and replaced the Pointer Network with Transformers 2.2.1 using Graph Attention Network 2.2.2 instead of positional encoding, and the Actor-Critic algorithm was changed to REINFORCE algorithm. The model showed superior performance. In this thesis, we will extend this model to support the soft time window constraint.

In 2019, Hottung et al. [53] introduced a novel approach that was inspired by LNS 3.3 called Neural Large Neighborhood Search. This model learns the destroy and repair operators by using graph attention network and attention mechanism, and it is outperforming the standard metahesuristics on capacitated vehicle routing problem. However, it is still iterative algorithm based on the local search which results in much larger runtime then end-to-end deep learning methods for VRP.

In 2021, another paper by Kool et al.[47] was released with a completely new approach. It combines dynamic programming and deep learning to solve VRP and promises much better performance than previous solutions. The method uses deep learning to restrict the dynamic programming search space using a policy derived from graph neural network. In the future, we will explore this method in depth and extend the support of constraints for time windows and pick and deliver problem 1.2.3

4.2 Solution

The end-to-end deep learning method pro solving VRPTW is extension of the work done by Kool et al. [7].

Let us describe the high-level concept behind the method. Consider we have a model as blackbox which takes VRPTW instance as an input and outputs probabilities for all the VRPTW nodes. The probability represents which node should be visited next and by following to the most probable node we get a partial solution which will be considered by the blackbox. We iterate this process until all nodes have been visited and we acquire a feasible plan as shown on Figure 4.1.

4.2.1 Model Architecture

The model architecture [7] leveraging recent advancements in attention mechanism is here extended by the time window constraint. The model is built upon transformers 2.2.1, graph attention network 2.2.2, and reinforcement learning 2.1. The network structure is encoder-decoder that fits well for solving sequential decision problems. The structural input instance is extracted by the encoder 4.2.1.1 and then the solution is incrementally constructed by the decoder 4.2.1.2.

The VRPTW input instance is consisted from:

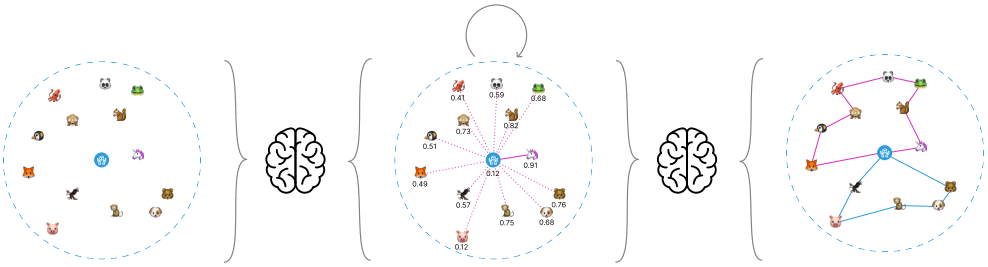

Figure 4.1: High-level concept behind the used method.

- $X = \{x_1, \cdots, x_n\}$ where x_i is two-dimensional coordinates in the euclidean space.
- x_0 is the location of depot.
- $D = \{d_1, \cdots, d_n\})$ is the demand capacity for each of the locations.
- $T = \{(e_1, l_1), \cdots, (e_n, l_n)\})$ is time windows for each of the location where e_i is the beginning and l_i is the end of the considered time window.

The output is the solution of VRPTW instance and is represented as a permutation π of locations $X \cup x_0$.

- $\pi = \{\pi_1, \cdots, \pi_T\} \in \{x_0, \cdots, x_n\})$

4.2.1.1 Encoder

The encoder uses graph attention network 2.2.2 to embed the node features to graph embedding. Then the decoder architecture is the same as the decoder of transformer 2.2.1. Typicaly, the decoder of transformer uses positional encoding [54] to embed the input, but in this case it has been replace with GAT 2.2.2 since we deal with graph-based structure and the input order does not matter.

The first step is to perform the initial embedding of input data 4.2.1 via learned linear projections as in GAT. The h_i^l represents the node embedding of layer $l \in \{0, \cdots, N\}$ (N=3).

$$\widetilde{x} = \text{concat}(X, D, T) \tag{4.1}$$

$$h_i^0 = \begin{cases} W\widetilde{x}_i + b_i & \text{if } i > 0 \\ W\widetilde{x}_0 + b_0 & \text{if } n = 0 \end{cases} \tag{4.2}$$

The node embeddings are updated via N attention layers, each containing multi-head attention 2.2.1.3 (M=8) and a fully connected feed-forward network with normalization. The structure is identical to transformer's encoder 2.2.1 with additional support of graph structure 2.2.2 as shown on Figure 4.3.

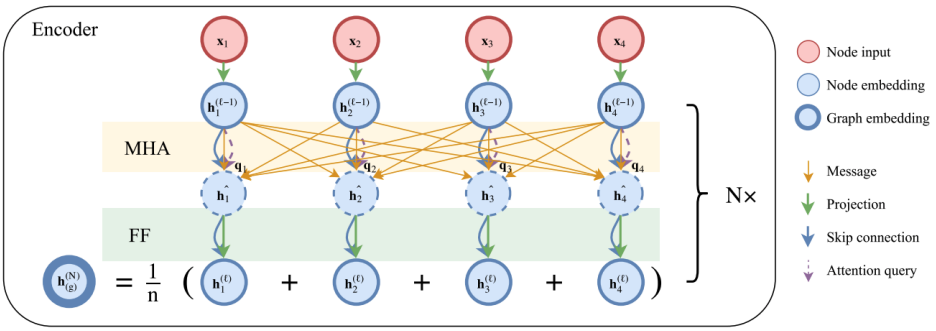

Figure 4.2: Encoder layers [7]

The equation 4.3 calculates the query Q, key K and value V of multi-head attention layer using the node embeddings and weights W_m^Q, W_m^Q, and W_m^Q, respectively. The number of heads is represented by $m \in \{1, \cdots, M\}$ (M=8).

$$\mathbf{q}_{im}^l = W_m^Q h_i^(l-1), \mathbf{k}_{im}^l = W_m^K h_i^(l-1), \mathbf{v}_{im}^l = W_m^V h_i^(l-1) \tag{4.3}$$

The query and key values are used in calculating the compatibility u_{ijm}^l of node i with a node j 4.4. If node i is not adjecnt to node j then they are not compatible and the value is set to a large negative number.

$$u_{ijm}^l = \begin{cases} q_{im}^l k_{jm}^l & \text{if } i \text{ adjacent to } j \\ -\infty & \text{otherwise} \end{cases} \tag{4.4}$$

The attention score $a_{ijm}^l \in [0, 1]$ is calculated using softmax from the compatibility values of nodes 4.19

$$a_{ijm}^l = \frac{e^{u_{ijm}^l}}{\sum_{j'=0}^{n} e^{u_{ij'm}^l}} \tag{4.5}$$

The transformed ${h'_{im}}^l$ 4.6 aggregates all attention scores across neighbour nodes, which is based on GAT 2.2.2.

$${h'_{im}}^l = \sum_{j=0}^{n} a_{ijm}^l v_{jm}^l \tag{4.6}$$

Finally, we may calculate the multi-head attention 2.2.1 for layer l as a function of $\{h_1^{l-1}, \cdots, h_n^{l-1}\}$ through ${h'_{im}}^{l}$.

$$\mathrm{MHA}_i^l(h_1^{l-1}, \cdots, h_n^{l-1}) = \sum_{m=1}^{M} W_m^O {h'_{im}}^{l} \tag{4.7}$$

$$\widetilde{h}_i = \mathrm{BN}^l(h_i^{l-1} + \mathrm{MHA}_i^l(h_1^{l-1}, \cdots, h_n^{l-1}))) \tag{4.8}$$

$$h_i^l = \mathrm{BN}^l(\widetilde{h}_i + \mathrm{FF}^l(\widetilde{h}_i)) \tag{4.9}$$

In the final layer, the encoder computes the aggregated embedding of the input graph as the mean of the final node embeddings.

$$h^N = \frac{1}{n} \sum_{i=1}^{m} h_i^N \tag{4.10}$$

The output of the encoder's final layer is passed to the decoder, which is detailed in the next sections 4.2.1.2.

4.2.1.2 Decoder

Decoder works sequentially through timestamps $t \in \{0, \cdots, n\}$, at each timestamp one node is selected to be visited based on partial route $\pi_{1:t-1}$. It is predicting the probability distribution over nodes according to the node embedding and context vector of the encoder 4.2.1.1.

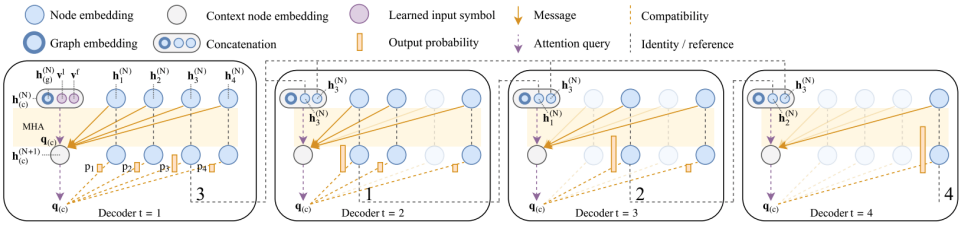

Figure 4.3: Describes the decoder iteration in the construction of a solution. This diagram very nicely visualizes the process and it was used in the paper by Kool et al. [7]

The decoder uses a new context vector $h_c^{'}$ which represents the state 2.1 and it goes as follows:

$$h_c^{'} = \begin{cases} \mathrm{concat}(h_N; h_0^N; D_t) & \text{if } t = 0 \\ \mathrm{concat}(h_N; h_{\pi_{1:t-1}}^N; D_t) & \text{if } t > 0 \end{cases} \tag{4.11}$$

The state of $h_c^{'}$ is concatenation of h_N, the output of the encoder, $h_{\pi_{1:t-1}}^N$, the embedding of previous partial solution, and D_t, the remaining demand capacity of the vehicle.

Due to the fact that the decoder architecture is transformer 2.2.1, the next layers are multi-head attentions which are responsible for choosing the next node to visit. This defines the system action 2.1.

The multi-head attention in the decoder is computed in a similar manner as in the decoder 4.2.1.1 with a little alternation.

$$q_{(c)m} = W_m^Q h_c^{'}, k_{jm} = W_m^K h_j^N, v_{jm} = W_m^V h_j^N \tag{4.12}$$

$$u_{(c)j} = \begin{cases} q_c^T k_j & \text{if } d_j <= D_t \text{ and } x_j \notin \pi_{1:t-1} \\ -\infty & \text{otherwise} \end{cases} \tag{4.13}$$

The equation 4.13 computes the compatibility score and performs a masking mechanism to mask the nodes which have already been visited during the partial route (besides depot x_0) and eliminates nodes where the vehicle capacity would overflow. If we would consider a time window as a hard constraint, the calculation of compatibility would have extended masking mechanism to show only nodes which correspond to the time t.

$$h'_{(c)m} = \sum_{j=0}^{n} softmax(u_{(c)j}) v_j m \tag{4.14}$$

$$h_c = \text{MHA}(h_c^{'}) = \sum_{m=1}^{M} W_m^O h'_{(c)m} \tag{4.15}$$

In order to calculate the desired probability $p_\theta(\pi_t | X, \pi_{1:t-1})$, a logit layer. The final layer is a single-head attention.

$$q = W^Q h_c, k_j = W^K h_j^N \tag{4.16}$$

$$u_j = \begin{cases} C.\tanh(q^T k_c) & \text{if } d_j <= D_t \text{ and } x_j \notin \pi_{1:t-1} \\ -\infty & \text{otherwise} \end{cases} \tag{4.17}$$

$$p_i = p_\theta(\pi_t | X, \pi_{1:t-1}) = \frac{e^{u_j}}{\sum_{j'=0}^{n} e^{u_{j'}}} \tag{4.18}$$

4.2.2 Reinforcement Learning

The model 4.2.1 takes VRPTW instance and outputs probability distribution over nodes $p_\theta(\pi|X)$ which is used to sample a full feasible route as a solution π. The instance of VRPTW S is defined as concatenation of locations, demand capacity and time windows for each node, $S = [X, D, T]$.

To train the model, we have to define a reward, respectively, a cost function. The model is trained using REINFORCE algorithm 2.1.3 as proposed by Kool et al. [7]. The algorithm is based on the computation of the policy gradient, which is defined as

$$\nabla_\theta \mathcal{L}(\theta|X) = \mathbb{E}[\mathcal{L}(\pi|X) - b(X)) \nabla \ln \pi(\pi|X))] \tag{4.19}$$

4.2.2.1 VRPTW Cost

For effectively solving VRPTW, the cost function is an integral part of a successfully trained model. In this thesis, we propose a new cost function to solve the vehicle routing problem with soft constrained time windows and demand capacity for each node.

The cost function is ranking the given solution of VRPTW instance. It penalizes the solution based on the length of the routes, early and late visits, and unequal distribution of nodes across vehicles.

For a given VRPTW instance S and its solution π, we propose the cost function as follows:

$$\mathcal{L}(\pi|S) = dis_p(\pi, S) + t_p(\pi, S) + bal_p(\pi, S) \tag{4.20}$$

$$dis_p(\pi, S) = \sum_{i=0}^{N} \left\| x_{\pi(i)} - x_{\pi(i+1)} \right\|_2 \tag{4.21}$$

The equation 4.21 calculates the length of all routes in Euclidean space.

$$t_p(\pi, S) = \sum_{i=0}^{N} (I_{e_i > \widetilde{t}_i}(e_i - \widetilde{t}_i)p_e + I_{l_i < \widetilde{t}_i}(\widetilde{t}_i - l_i)p_l) \tag{4.22}$$

The equation 4.22 calculates the penalty for early or late arrival. The time of the visit for a given node i is defined by vector $\widetilde{t}_i$. We assume that the travel speed is always identical and we approximate that one unit of distance equals to one unit of time. The vector I behaves as a mask $I \in (0, 1)^n$ which represents if either early or late arrival occurred. The penalty for late arrival p_l should be greater than for early arrival p_e and finding the proper penalties will be empirically determined as a part of experiment chapter **??**.

$$bal_p(\pi, S) = \sigma([|R_0|, \cdots, |R_k|]) \tag{4.23}$$

The last subpart of the cost function is calculating balance cost 4.23 that aims to evenly distribute the number of nodes in a route R_i by minimizing its standard deviation. In logistics, we expect to utilize couriers evenly.

4.2.2.2 Training loop

Pseudocode of the RL system training loop is as follows

Algorithm 4: REINFORCE algorithm

Input: Number of epoch E, steps per epoch T, batch size B
Result: Updated θ that maximises reward
Initialize θ at random;
for *epoche* $= 1, 2, \cdots, E$ **do**
  for *steps* $= 1, 2, \cdots, T$ **do**
    Compute context embedding h^N via decoder (4.2.1.1);
    for $t = 1, 2, \cdots, N$ **do**
      Calculate $p_\theta(\pi_t|X, \pi_{1:t-1})$ via encoder for t (4.2.1.2);
      Pick an action based on probability distribution;
      Update the state by visiting a new node;
    Compute reward $\mathcal{L}(\pi|S)$ (4.20);
    $\nabla_\theta \mathcal{L} \leftarrow \mathbb{E}[\mathcal{L}(\pi|X)\nabla \ln \pi(\pi|X))]$;
    $\theta \leftarrow \text{Adam}(\theta, \nabla_\theta \mathcal{L})$;

4.2.3 Integrating Duration Matrix

Real-world application of VRP require to obtain the distance and duration matrix which represents the weighted transition between graph nodes. The duration between two locations is typically defined by the infrastructure and speed limits on a given route. Such a duration matrix is calculated using map data such as OpenStreetMap [55].

The neural network solving VRPTW learns to approximate the Euclidean distance between two given points. However, if we would integrate the duration matrix into the cost function of the model, the network would have to derive the duration between two points, which is an impossible task with the given model architecture. Moreover, the planning would be fixed to the location (city) on which the model was trained on.

We propose an indirect integration of the duration matrix for the input instance. We may project the duration matrix into the node locations using multidimensional scaling [56] which would embed the duration information in a given 2D space.

Chapter 5

Planning System

In this chapter, take a look at how the GoDeliver[2] system works and the integration of the VRPTW AI planner.

5.1 GoDeliver System

The GoDeliver system is consisted of multiple services, each responsible for a given subproblem of the delivery process. The core of the system is GoDeliver service, which implements all system APIs and performs the basic CRUD[3] operation on our NoSQL database. The database stores information about business, delivery orders, couriers, and delivery plans. In the Figure 5.1 is visualized the simplified architecture of GoDeliver system, it is especially oriented to show how the planning process works.

5.1.1 Planning process

The planning process is a complex operation which requires to work asynchronously because the planning of delivery orders is a time-consuming operation. The modern delivery planners have to support dynamic rescheduling and autonomously act upon the constantly changing environment.

The first part of the planning process is the creation of a delivery order, which is a request for delivery via API. The delivery order is saved in the database by GoDeliver Service with additional meta-data about the delivery state, etc. The database is monitored for a so-called *trigger changes* which fires a replanning job via a distributed task queue Celery[4]. The *trigger changes* are a list of actions such as creation of a new delivery order, changes in courier capacity, or a significant delay of delivery.

[2] `https://godeliver.co/`
[3] Create, read, update and delete
[4] `https://github.com/celery/celery`

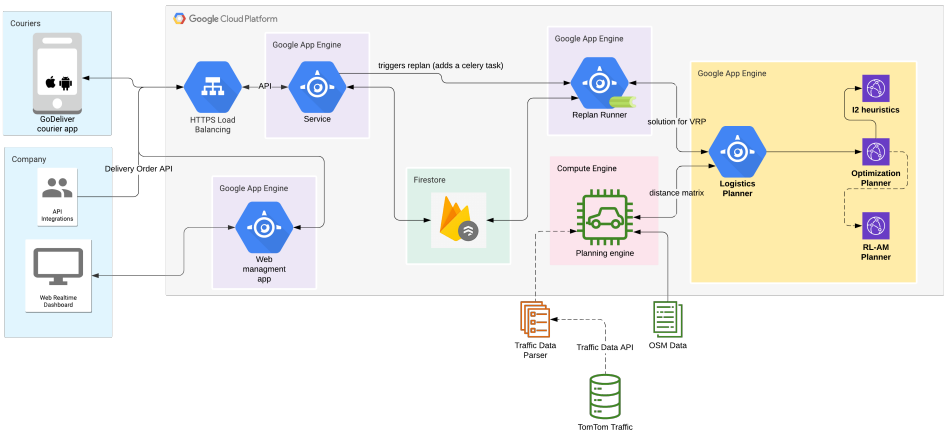

Figure 5.1: GoDeliver System Architecture

If such a replanning job is created, it is saved in Celery queue which is processed by GoDeliver Replanning Service 5.1. The replanning service loads a business configuration, delivery orders, and delivery plans from the database. It transforms the data into a generic structure which is accepted by our another service logistics planners that are solving the vehicle routing problem. The generic plan structure for the logistics planner freezes some delivery points which should not be considered by the planner since we do not want to change the current in-progress delivery points. This process is enabling us to perform the dynamic vehicle routing problem 1.2.5.

Based on the business config, the desired vehicle routing solver is invoked via the logistics planner API. Usually, the planner takes the previous delivery plan and performs a heuristic algorithm like insertion heuristic which outputs an extended feasible plan. This plan is then improved by a local search algorithm to improve its cost function. Then the solution is processed by GoDeliver Replanning Service and saves the new delivery plans into the database.

In the future, the proposed VRPTW AI planner will be used instead of insertion heuristics because we expect the AI planner will outperform the insertion heuristics. The downside is that AI planner is not able to leverage on the previous solution, which can lead to drastic changes in the solution structure of the delivery plan. However, this side effect is not a blocker since couriers only see one current delivery point from the delivery plan via GoDeliver mobile app 5.2.

5.1.2 Planning Requirements

The GoDeliver objective is to provide the most advanced and versatile urban logistics system. The logistics cases are always different for individual busi-

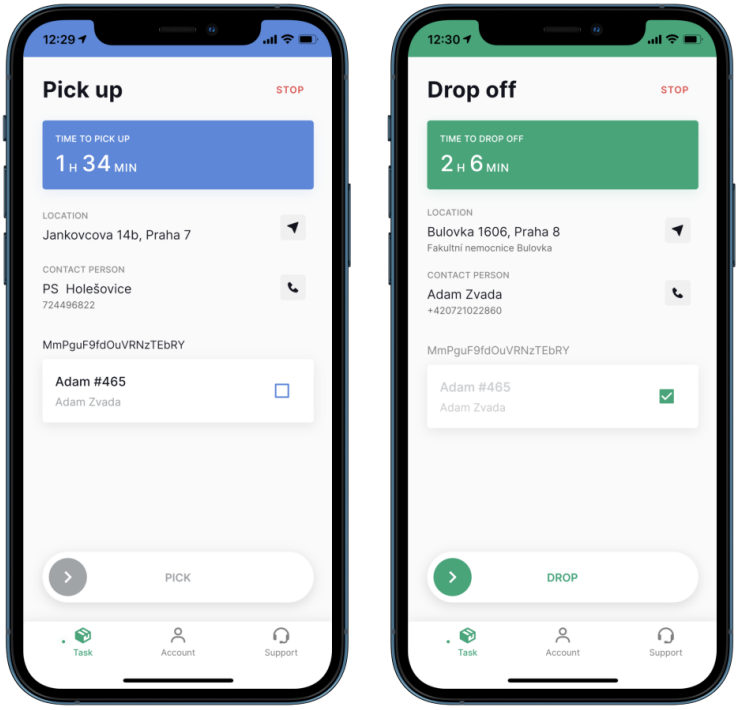

Figure 5.2: GoDeliver Driver App

nesses and in order to cover them all we require a flexible but yet powerful planning system. For a new planner to be successfully used in the production environment, we have defined the planner requirements which have to be supported 5.1.

In the table 5.1, we have summarized the supported features of our proposed VRPTW AI It does not support features such as Pick and Deliver, predefined number of vehicles, and distance matrix. The Pick and Delivery is possible to support by extending the model to support a heterogeneous fleet based on this paper by J. Li et. al [57] which extends the masking mechanism and RL state and action to support the pick and deliver constraints. Distance matrix could be indirectly supported by applying multidimensional scaling [58] to project the distance matrix into Euclidean space which is supported by the model. To define a fixed number of vehicles for a given VRPTW instance is surprisingly more complicated, but we propose that it can be achieved by implementing the support of multiagent reinforcement learning [59] where each agent is one vehicle.

Feature	Is supported?
Time windows	✓
Soft constraint	✓
Distance matrix	indirectly[5]
Pick and Deliver	-
Demand Capacity	✓
Balanced load across vehicles	✓
Predefined number of vehicles	-

Table 5.1: VRPTW AI support of planner requirements

5.2 Tech Stack - VRPTW via Optimization

The planners based on optimization heuristics are implemented in programming languages Python[6] and Go[7] and our implemented metaheuristic uses the optimization framework OR-Tools.

5.3 Tech Stack - VRPTW via AI

The VRPTW planner via AI is implemented in the programming language Python which is the most favorite programming language for any AI-related project [60]. Besides it is easy language to get start with, it has many amazing AI and data libraries such as Pandas, Numpy, Tensorflow or PyTorch.

Production implementation of neural network is developed with the use of deep learning frameworks. TensorFlow[8] and PyTorch[9] are the two most popular frameworks. It is an important decision which deep learning framework to choose if you plan to develop a production ready ML pipeline.

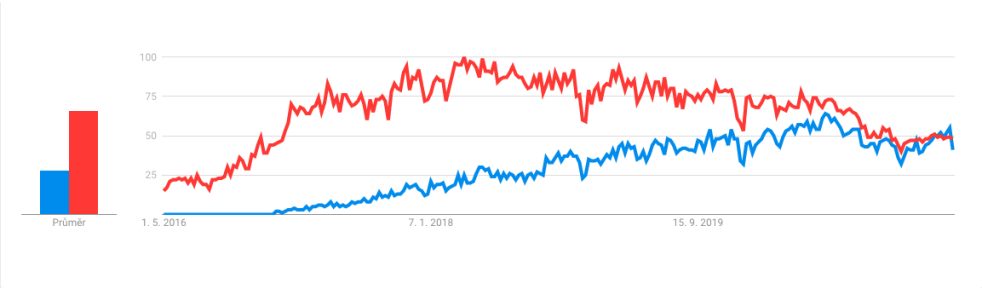

Figure 5.3: Google Trend of PyTorch (blue) vs Tensorflow (red)

[6] https://python.org/
[7] https://golang.org/
[8] https://pytorch.org/
[9] https://pytorch.org/

The Google Trends graph 5.3 shows how TensorFlow was more popular in the past, but lately they have similar amount of search results. However, the majority of the new research papers are implemented via PyTorch [61]. It is due to the fact that PyTorch has a great and intuitive API based on Numpy[10] operations and it is even slightly faster than TensorFlow.

In this thesis, we have decided to use PyTorch as our main deep learning framework.

[10] `https://numpy.org/`

Chapter 6

Evaluation

In this chapter, we will evaluate our proposed planning model based on deep reinforcement learning built upon Transformer 2.2.1 architecture utilizing Graph Attention Network 2.2.2 and benchmark the performance against constructive heuristics and metaheuristics.

6.1 Dataset

The dataset used for training and evaluation was generated on the fly via a uniform probability distribution within a given range. The reason we decided to generate the data is that learning reinforcement policy requires a large amount of training data. Since it learns by interacting with the environment, we do not need labeled datasets and generating them seems as the best approach. Nevertheless, in further work, the model will require other kinds of distribution to simulate a real-world demand.

In real-world routing applications, the geographic coordinate system is typically used for specifying locations. However, our proposed model requires locations between $[0, 1]$ to achieve model convergence. The locations of depots and delivery nodes were generated via uniform distribution within a range of $[0, 1] \times [0, 1]$.

The data for the demand capacity of a delivery node is a discrete number $\{1, \cdots, 9\}$ chosen uniformly at random with assumption that the depot has a demand capacity of zero.

Lastly, each location has assigned time windows that work as a soft constraint at which time a vehicle is supposed to visit the node. We generate different lengths of time windows based on the problem size (20, 50, 100). For the problem size of 20 nodes, the time window value occurs in a range of $\{0, \cdots, 10\}$ with the condition that the length of the time window is less than four. The problem size of 50 nodes has the upper bound set to 20 with a maximal length of 6 and the case of 100 nodes, the upper bound is 40 and the

maximal window length is 9. Similarly, the start and end of a time window are generated with uniform distribution.

6.2 Sample Solutions

To better imagine the complexity and variability of solving VRPTW, in Figure 6.1 and 6.2 we illustrate a random solution predicted by our proposed deep learning model. On the left, we visualize the customer's time windows using Gantt diagram and on the right are shown the routes for each vehicle. Each color represents a vehicle which servers the customers represented by a node with a generated time window.

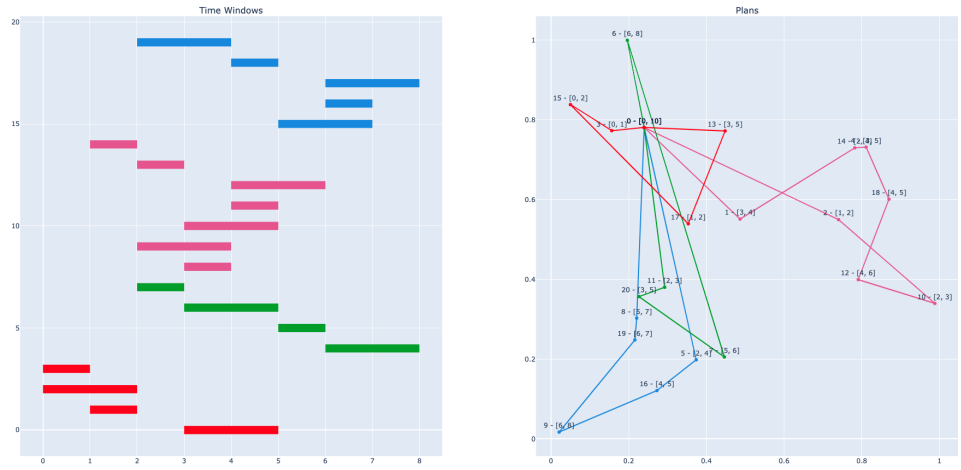

Figure 6.1: Sample solution of random VRPTW instance for problem size 20.

The AI model has to optimize the route path with focus to minimize the travelled distance, but simultaneously early and late arrivals should be avoided. Our model is able to solve VRPTW instance considers both distance and time window constrains. The windows are soft constrained, which results in a much larger combinatorial space of possible solutions than considering just hard constrained time windows. As illustrated on Figure 6.2 the model sometimes sacrifices the time of a served customer in order to optimize the route distance and vice versa.

The model has successfully learned the policy how to pick the next node to visit, the time windows are almost cascadingly sorted, and the routes are sub-optimally optimized based on distance. However, it is clearly not an optimal solution, but it proves that the problem of vehicle routing with soft constrained time windows is possible to be solved with ML and the research is on the right path to outperform regular metaheuristics. In the section 6.4 we compare the model with our implemented constructive heuristics and metaheuristics

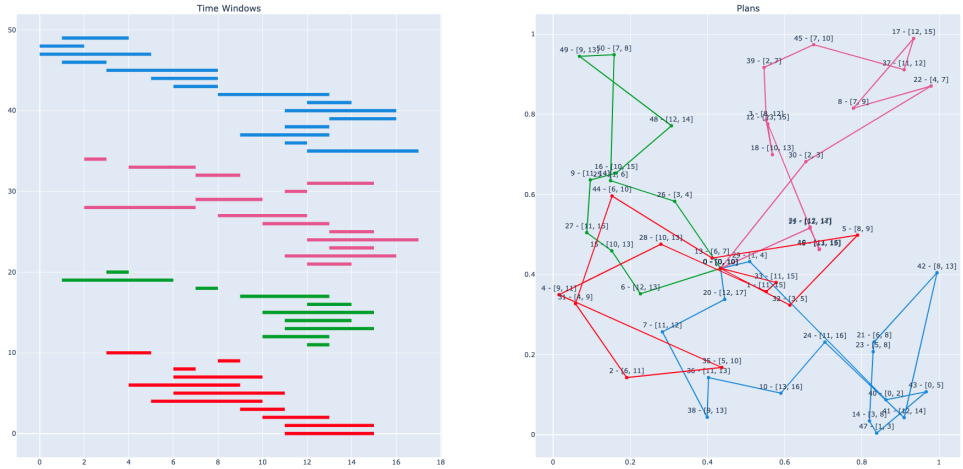

Figure 6.2: Sample solution of random VRPTW instance for problem size 50.

6.3 Experiments

6.3.1 Time Windows

The main goal of this thesis was to integrate the soft constraint of time windows to VRP model introduced by Kool et al.[7]. The model is extended by our proposed cost function 4.20 that penalizes early and late arrivals. The early arrival is not as crucial as delayed visit, so we have decided that the penalty for early arrival is always less than late visit $p_e < p_l$.

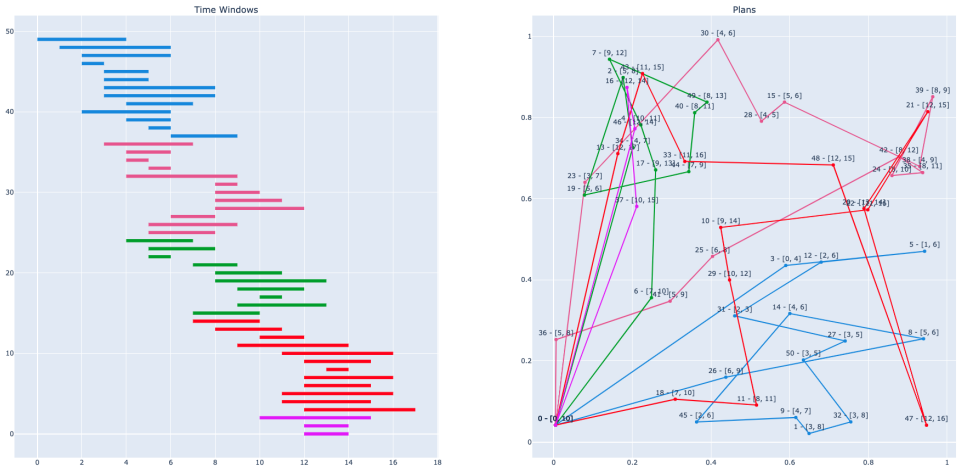

Figure 6.3: Low penalty for early visit of node results in poor performance.

Finding a proper penalty values p_e and p_l for time windows is one of the

most important steps because an incorrect penalty highly influences the model in choosing the next node to visit.

The Figure 6.3 illustrates how the performance is degraded by choosing an incorrect penalty value for early arrival p_e. If the early penalty is too low, the model will strictly focus on minimizing the late arrival and Gantt diagram of time windows looks like a plan for just a single vehicle.

We have empirically tested various kinds of penalties and updated them based on our observations. After the experiments, we have chosen $p_e = 0,25$ and $p_l = 0,5$ which resulted in predicting a feasible plans with properly distributed time windows across the vehicles as shown in Figure 6.4.

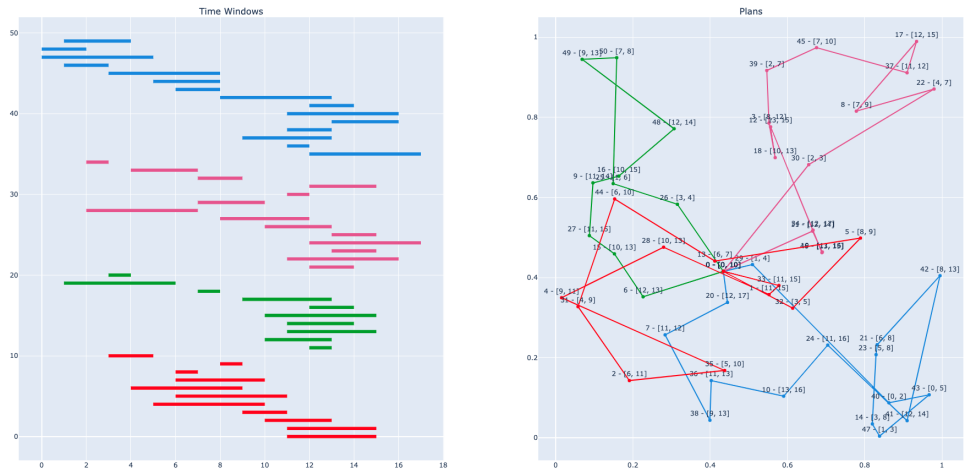

Figure 6.4: Properly distributed time windows across the vehicles.

6.3.2 Balancing Plans

In real-world solution of VRP, we aim to balance the number of served customers across vehicles. It means the number of nodes in a given route is supposed to be uniformly distributed across all routes. However, we have noticed that the model was constantly adding additional routes of size one as illustrated on Figure 6.5.

Therefore, we have decided to penalize the delivery plans that are not balanced by including an unbalance penalty. We calculate the standard deviation of plans with the objective to be minimized using our cost function. This approach helped in creating balanced delivery plans.

6.3.3 Generalization

A great downside of this proposed model is its sensitivity to data input, which results in poor model generalization. If we introduce to a model a distribution

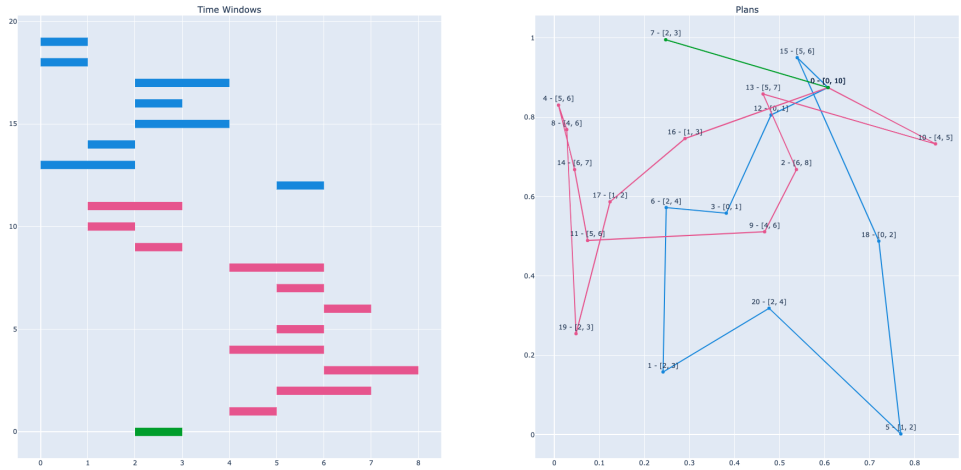

Figure 6.5: Unbalanced delivery plans.

of data which has not been seen before, it is not possible to predict sensible delivery plans and the model just serves each of the nodes by a single vehicle. This behavior is shown in Figure 6.6. The model has even problems with generalization to different input data distributions in a single dataset.

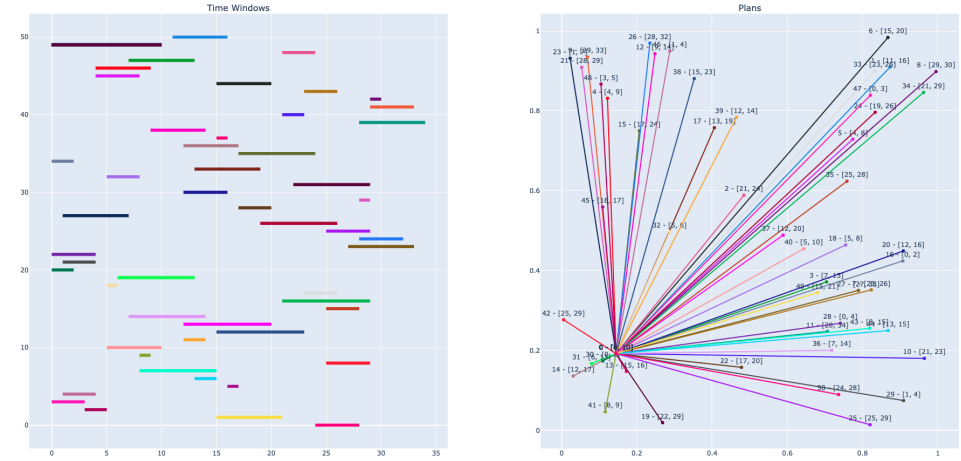

Figure 6.6: If the model does not know how to solve the instance, it just sends each vehicle to serve one node.

Another model sensitivity is to the instance problem size. If the model is trained on the problem size of 20 nodes, the model performance is continuously degraded by a larger difference of the problem size on which it was trained on. We tried training the model on a dataset of different problem sizes, but the network was not effectively learning. The model has to be trained on a

problem size with a minimal difference.

The solution is to train multiple models for each of the problem sizes and data distribution. However, this seems as a great obstacle in using this model in real-world environment and thus a new model with improved model generalization needs to be proposed by researches.

6.3.4 Training Process

The training process of such a network is very time consuming and with our limited available hardware, the VRPTW model for solving 50 nodes requires at least 5 days to be fully trained. Therefore, experimenting with a different values of network hyperparameters is not part of this thesis. We only focused on proving if even such a problem like VRPTW can be sub-optimally solved via ML. We have used the same hyperparameters as Kool et al. [7] in their research.

During the training process, we have observed multiple values of the cost function 4.20 to understand if the network was successfully learning to solve the problem.

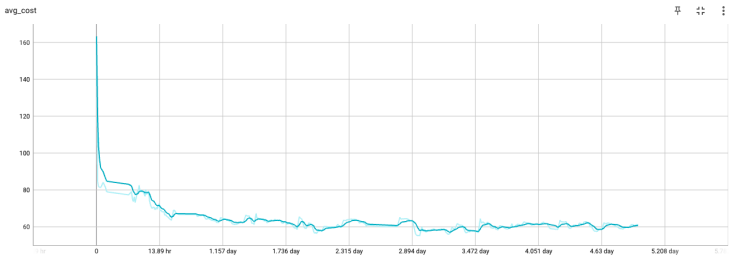

Figure 6.7: Average cost (reward) per epoch on training data.

In Figure 6.7 we observe the average cost of the predicted solutions per epoch. It fairly quickly plateaued, but if we look at the average cost on the validation dataset 6.8, the value was still decreasing. The model was still improving just on different parts of the cost function.

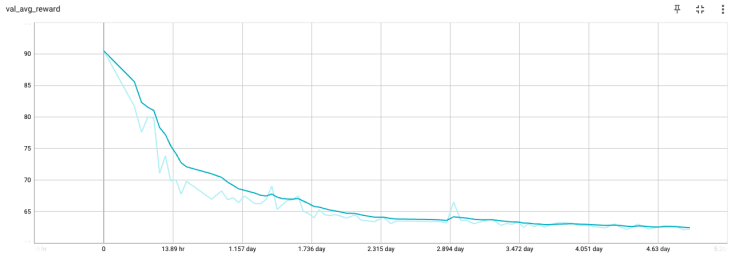

Figure 6.8: Average cost (reward) per epoch on validation data.

If we would experiment with hyperparameter tuning, in this case, we could try to use the decaying learning rate which we assume would further more decreased the cost function.

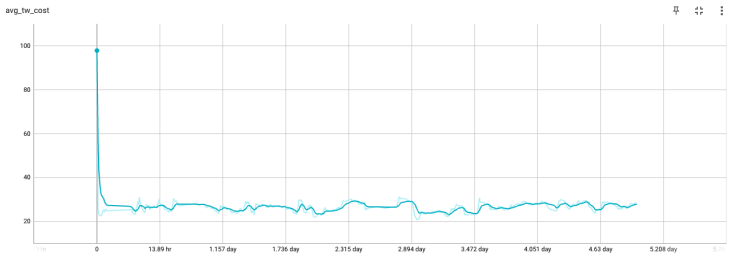

Figure 6.9: The average cost for time windows per epoch.

In Figure 6.9 we observe the average penalty for early and late arrivals of the predicted routes per epoch. The model quickly learns the policy strategy how to predict the next node to visit based on the time constraint. We assume that the model first learns the strategy for minimizing the time window based on the drastic drop of the time window cost function and then it learns to optimize the distance of routes. In Figure 6.10 we observe the average total distance of predicted routes per epoch and the cost is constantly decreasing while the model learns to optimize the distance.

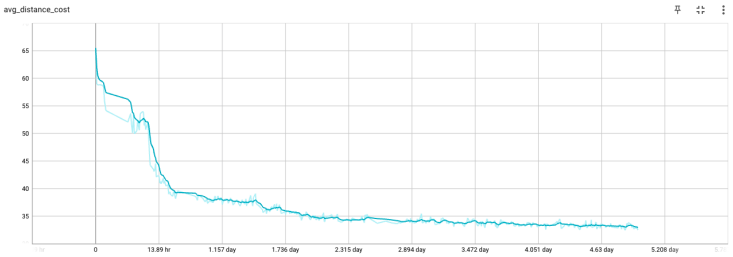

Figure 6.10: The average distance cost per epoch.

In Figure 6.11 is the last observed part of the cost function, the unbalance penalty, which is slowly decreasing while the model learns to distribute the nodes uniformly across the vehicles.

The cost function is used as a reward function for RL. It is important to note that the model can never achieve a value of the cost function which would converge to zero.

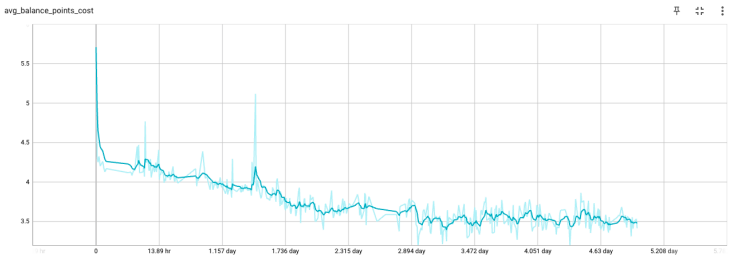

Figure 6.11: The average cost of balanced plans.

6.4 Benchmarking

We know that our deep reinforcement learning model for sub-optimally solving VRPTW can predict a solution for VRPTW instances 6.2. However, to evaluate the model performance, we need to compare it with other reliable methods like constructive heuristics and metaheuristics.

The first benchmark is insertion heuristics 3.1 which produces a feasible solution in a matter of seconds. It is frequently used to generate an initial solution which is then optimized by a local search algorithm or population-based method. The second typical benchmark used in research papers is OR-Tools framework 3.2 and its is widely used for many optimization tasks. Lastly, the third benchmark combines insertion heuristics as an initial solution and OR-Tools planner as a metaheuristic.

In tables 6.1, 6.2 and 6.3 are measured the average cost of our implemented planners on 64 random VRPTW instances. We evaluate on all the essential subparts of the cost function where the *AggCost* is the same as being used in our model architecture 4.20. Additionally, we have tracked the computation time per each instance to emphasize on the great advantage of our model.

Our proposed deep reinforcement learning model is RL_AM_PLANNER. Additionally, we have implemented RL_AM_OR_TOOLS_PLANNER, a planner that uses the predicted solution of our model as a constructive heuristic which is then optimized by OR-Tools.

Model	DistanceCost Km	DelayCost min.	EarlinessCost min.	ComputationTime sec	**AggCost**
INSERTION_HEURISTIC	981	0	847	1	**1,193**
RL_AM_PLANNER	610	38	871	0.5	**847**
OR_TOOLS	482	4	814	120	**687**
OR_TOOLS_INSERTION	488	1	798	121	**688**
RL_AM_OR_TOOLS_PLANNER	482	4	814	120	**688**

Table 6.1: Benchmarking of VRPTW solvers for problem size of 20 nodes.

Our proposed model RL_AM_PLANNER is outperforming the constructive heuristic as expected, but OR-Tools as our metaheristics, clearly outperforms the deep learning model. The RL_AM_PLANNER model for the problem

Model	DistanceCost Km	DelayCost min.	EarlinessCost min.	ComputationTime sec	**AggCost**
INSERTION_HEURISTIC	2,306	0	4,355	39	**3,395**
RL_AM_PLANNER	1,297	100	3,634	2	**2,255**
OR_TOOLS	757	539	6,180	154	**2,571**
OR_TOOLS_INSERTION	2,007	3	3,967	192	**3,000**
RL_AM_OR_TOOLS_PLANNER	759	537	6,161	156	**2,568**

Table 6.2: Benchmarking of VRPTW solvers for problem size of 50 nodes.

Model	DistanceCost Km	DelayCost min.	EarlinessCost min.	ComputationTime sec	**AggCost**
INSERTION_HEURISTIC	4,654	0	17,548	750	**9,041**
RL_AM_PLANNER	$5,852^*$	1^*	$15,805^*$	16	$9,804^*$
OR_TOOLS	1,034	2,885	23,719	199	**8,406**
OR_TOOLS_INSERTION	4,478	0	17,109	807	**8,755**
RL_AM_OR_TOOLS_PLANNER	$1,034^*$	$2,885^*$	$23,719^*$	219^*	$8,406^*$

Table 6.3: Benchmarking of VRPTW solvers for problem size of 100 nodes.

size of 100 was not fully trained and the data may be biased (marked with x^*). However, it is important to appreciate the negligible computational time which the deep learning model requires to predict a feasible but still highly suboptimal solution in compression with metaheuristics solvers.

Conclusion

The thesis objective was to explore solutions for the vehicle routing problem with soft constrained time windows (VRPTW) using optimization heuristics (metaheuristics) and primarily machine learning. Even though the general vehicle routing problem has seen novel proposed approaches in recent years, the problem of vehicle routing with soft constrained time windows has received almost no attention. Based on this realization, we have focused on proposing a solution for vehicle routing problem with soft constrained time windows using machine learning techniques and comparing it with metaheuristic solvers.

In this thesis, we have formally defined the problem of vehicle routing and described its other flavours such as time windows. The thesis provides the necessary theoretical background of reinforcement learning, attention mechanism, Transformer network, and Graph Attention Network to fully understand the implemented solution of vehicle routing with soft constrained time windows. It covers various methods for solving our problem using constructive heuristics and metaheuristics. We also described how a production ready planner should look like.

This thesis describes a new method for solving a vehicle routing problem with soft constrained time windows using deep reinforcement learning. The model is built upon Transformer architecture utilizing Graph Attention Network for embedding the input instance. The model uses a newly proposed reward function that incorporates the time window constraint.

We have developed and trained our proposed deep reinforcement learning using PyTorch and additionally implemented insertion heuristics as a constructive heuristic benchmark and OR-Tools solver as a metaheuristic benchmark. The model was successfully trained on a generated dataset of VRPTW instances because it requires a large amount of training unlabeled data.

We have evaluated the performance of our model against our implemented constructive heuristic and metaheuristic. The model significantly outperforms the constructive heuristics but is still behind the OR-Tools metaheuristics. We have assumed that the model could be a great alternative to constructive

heuristics, but because of its poor generalization, it is impossible to use it as a production ready planner.

In the future work, we will focus on a little different approach and adopt a new model architecture of Kool et al. [47] which we plan to extend to support soft constrained time windows and pick and deliver.

Bibliography

[1] Available from: https://neo.lcc.uma.es/vrp/vehicle-routing-problem/

[2] Bono, G. Deep multi-agent reinforcement learning for dynamic and stochastic vehicle routing problems. , no. 2020LYSEI096, Oct. 2020. Available from: https://tel.archives-ouvertes.fr/tel-03098433

[3] Sutton, R. S.; Barto, A. G. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018, ISBN 0262039249.

[4] Weng, L. Attention? Attention! *lilianweng.github.io/lil-log*, 2018. Available from: http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

[5] Vaswani, A.; Shazeer, N.; et al. Attention Is All You Need. *CoRR*, volume abs/1706.03762, 2017, 1706.03762. Available from: http://arxiv.org/abs/1706.03762

[6] Veličković, P.; Cucurull, G.; et al. Graph Attention Networks. 2018, 1710.10903.

[7] Kool, W.; van Hoof, H.; et al. Attention, Learn to Solve Routing Problems! 2019, 1803.08475.

[8] Lenstra, J. K.; Kan, A. H. G. R. Complexity of vehicle routing and scheduling problems. *Networks*, volume 11, no. 2, 1981: pp. 221–227, doi:https://doi.org/10.1002/net.3230110211, https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230110211. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230110211

[9] Dantzig, G. B.; Ramser, J. H. The Truck Dispatching Problem. *Management Science*, volume 6, no. 1, 1959: pp. 80–91. Available from: https://EconPapers.repec.org/RePEc:inm:ormnsc:v:6:y:1959:i:1:p:80-91

[10] Solomon, M. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Oper. Res.*, volume 35, 1987: pp. 254–265.

[11] Psaraftis, H. A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem. *Transportation Science*, volume 14, 05 1980: pp. 130–154, doi:10.1287/trsc.14.2.130.

[12] Bertsimas, D.; Jaillet, P.; et al. A Priori Optimization. *Operations Research*, volume 38, 02 1989, doi:10.1287/opre.38.6.1019.

[13] Toth, P.; Vigo, D.; et al. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. USA: Society for Industrial and Applied Mathematics, 2014, ISBN 1611973589.

[14] Gendreau, M.; Laporte, G.; et al. Stochastic vehicle routing. *European Journal of Operational Research*, volume 88, no. 1, 1996: pp. 3–12, ISSN 0377-2217, doi:https://doi.org/10.1016/0377-2217(95)00050-X. Available from: https://www.sciencedirect.com/science/article/pii/037722179500050X

[15] STEIN, D. M. Scheduling Dial-a-Ride Transportation Systems. *Transportation Science*, volume 12, no. 3, 1978: pp. 232–249, ISSN 00411655, 15265447. Available from: http://www.jstor.org/stable/25767916

[16] DROR, M.; TRUDEAU, P. Savings by Split Delivery Routing. *Transportation Science*, volume 23, no. 2, 1989: pp. 141–145, ISSN 00411655, 15265447. Available from: http://www.jstor.org/stable/25768367

[17] Last-mile logistics worldwide - Statistics Facts. Available from: https://www.statista.com/topics/4383/last-mile-delivery/#dossierSummary

[18] Bellman, R. E. *The Theory of Dynamic Programming*. Santa Monica, CA: RAND Corporation, 1954.

[19] Thomas, P. S.; Brunskill, E. Policy Gradient Methods for Reinforcement Learning with Function Approximation and Action-Dependent Baselines. 2017, 1706.06643.

[20] j2kun. Markov Chain Monte Carlo Without all the Bullshit. Sep 2016. Available from: https://jeremykun.com/2015/04/06/markov-chain-monte-carlo-without-all-the-bullshit/

[21] Ng, A. CS229 Lecture notes - Supervised learning, 2012.

[22] Weng, L. Policy Gradient Algorithms. *lilianweng.github.io/lil-log*, 2018. Available from: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

[23] Williams, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, volume 8, 1992: pp. 229–256.

[24] Available from: https://danieltakeshi.github.io/2017/03/28/going-deeper-into-reinforcement-learning-fundamentals-of-policy-gradients/

[25] Bahdanau, D.; Cho, K.; et al. Neural Machine Translation by Jointly Learning to Align and Translate. 2016, 1409.0473.

[26] Radford, A.; Wu, J.; et al. Language Models are Unsupervised Multitask Learners. 2019.

[27] Hochreiter, S.; Schmidhuber, J. Long Short-term Memory. *Neural computation*, volume 9, 12 1997: pp. 1735–80, doi:10.1162/neco.1997.9.8.1735.

[28] He, K.; Zhang, X.; et al. Deep Residual Learning for Image Recognition. 2015, 1512.03385.

[29] Ba, J. L.; Kiros, J. R.; et al. Layer Normalization. 2016, 1607.06450.

[30] Kipf, T. N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. 2017, 1609.02907.

[31] Graph attention network. Available from: https://docs.dgl.ai/en/0.4.x/tutorials/models/1_gnn/9_gat.html

[32] Jaw, J.-J.; Odoni, A. R.; et al. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, volume 20, no. 3, 1986: pp. 243–257, ISSN 0191-2615, doi:https://doi.org/10.1016/0191-2615(86)90020-2. Available from: https://www.sciencedirect.com/science/article/pii/0191261586900202

[33] Solomon, M. M. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, volume 35, no. 2, 1987: pp. 254–265. Available from: https://EconPapers.repec.org/RePEc:inm:oropre:v:35:y:1987:i:2:p:254-265

[34] Lu, Q.; Dessouky, M. M. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, volume 175, no. 2, 2006: pp. 672–687, ISSN 0377-2217, doi:https://doi.org/10.1016/j.ejor.2005.05.012. Available from: https://www.sciencedirect.com/science/article/pii/S0377221705004698

[35] Perron, L.; Furnon, V. OR-Tools. Available from: https://developers.google.com/optimization/

[36] Voudouris, C.; Tsang, E.; et al. *Guided Local Search*. 09 2010, pp. 321–361, doi:10.1007/978-1-4419-1665-5_11.

[37] Kirkpatrick, S.; Gelatt, C.; et al. Optimization by Simulated Annealing. *Science (New York, N.Y.)*, volume 220, 06 1983: pp. 671–80, doi:10.1126/science.220.4598.671.

[38] Glover, F. Future paths for integer programming and links to artificial intelligence. *Computers Operations Research*, volume 13, no. 5, 1986: pp. 533–549, ISSN 0305-0548, doi:https://doi.org/10.1016/0305-0548(86)90048-1, applications of Integer Programming. Available from: https://www.sciencedirect.com/science/article/pii/0305054886900481

[39] Pacheco, J.; Martí, R. Tabu search for a multi-objective routing problem. *Journal of the Operational Research Society*, volume 57, no. 1, 2006: pp. 29–37, doi:10.1057/palgrave.jors.2601917, https://doi.org/10.1057/palgrave.jors.2601917. Available from: https://doi.org/10.1057/palgrave.jors.2601917

[40] Shaw, P. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. 1998.

[41] Pisinger, D.; Ropke, S. *Large Neighborhood Search*. 09 2010, pp. 399–419, doi:10.1007/978-1-4419-1665-5_13.

[42] Schrimpf, G.; Schneider, J.; et al. Record Breaking Optimization Results Using the Ruin and Recreate Principle. *Journal of Computational Physics*, volume 159, 04 2000: pp. 139–171, doi:10.1006/jcph.1999.6413.

[43] Ropke, S.; Pisinger, D. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, volume 40, 11 2006: pp. 455–472, doi:10.1287/trsc.1050.0135.

[44] Reimann, M.; Doerner, K.; et al. D-ants: Savings Based Ants Divide and Conquer the Vehicle Routing Problem. *Computers Operations Research*, volume 31, 04 2004: pp. 563–591, doi:10.1016/S0305-0548(03)00014-5.

[45] Bullnheimer, B.; Hartl, R.; et al. A New Rank Based Version of the Ant System - A Computational Study. *Central European Journal of Operations Research*, volume 7, 01 1999: pp. 25–38.

[46] Cappart, Q.; Moisan, T.; et al. Combining Reinforcement Learning and Constraint Programming for Combinatorial Optimization. 2020, 2006.01610.

[47] Kool, W.; van Hoof, H.; et al. Deep Policy Dynamic Programming for Vehicle Routing Problems. 2021, 2102.11756.

[48] *Local Search in Combinatorial Optimization*. Princeton University Press, 2003. Available from: http://www.jstor.org/stable/j.ctv346t9c

[49] Lawler, E. L.; Wood, D. E. Branch-And-Bound Methods: A Survey. *Operations Research*, volume 14, no. 4, 1966: pp. 699–719, ISSN 0030364X, 15265463. Available from: http://www.jstor.org/stable/168733

[50] Vinyals, O.; Fortunato, M.; et al. Pointer Networks. 2017, 1506.03134.

[51] Bello, I.; Pham, H.; et al. Neural Combinatorial Optimization with Reinforcement Learning. 2017, 1611.09940.

[52] Nazari, M.; Oroojlooy, A.; et al. Reinforcement Learning for Solving the Vehicle Routing Problem. 2018, 1802.04240.

[53] Hottung, A.; Tierney, K. Neural Large Neighborhood Search for the Capacitated Vehicle Routing Problem. 08 2020.

[54] Transformer Architecture: The Positional Encoding. Available from: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

[55] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org, 2017.

[56] Borg, I.; Groenen, P. Modern Multidimensional Scaling: Theory and Applications. *Journal of Educational Measurement*, volume 40, 06 2006: pp. 277 – 280, doi:10.1111/j.1745-3984.2003.tb01108.x.

[57] Li, J.; Xin, L.; et al. Heterogeneous Attentions for Solving Pickup and Delivery Problem via Deep Reinforcement Learning. *IEEE Transactions on Intelligent Transportation Systems*, 2021: pp. 1–10, doi: 10.1109/TITS.2021.3056120.

[58] Davison, M.; Sireci, S. Multidimensional Scaling. 10 2012, doi:10.1016/B978-012691360-6/50013-6.

[59] Zhang, K.; Yang, Z.; et al. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. 2019, 1911.10635.

[60] Stack Overflow Developer Survey 2020. Available from: https://insights.stackoverflow.com/survey/2020

[61] He, H. The State of Machine Learning Frameworks in 2019. *The Gradient*, 2019.

Appendix A

Acronyms

AI artificial intelligence

ALNS Adaptive Large Neighborhood Search

CVRP capacitated vehicle routing problem

GAT Graph Attention Network

GCN Graph Convolution Networks

LNS Large Neighborhood Search

MHA Multi-head Attetntion

ML machine learning

PDP Pick and Deliver

RL reinforcement learning

TSP traveling salesman problem

VRP vehicle routing problem

VRPPD vehicle routing problem with pick and deliver

VRPTW vehicle routing problem with time windows

Appendix B

Media contents

- `readme.txt` ..... introductionary instructions
- `rl-am-vrptw` ..... the directory of VRPTW using deep learning
  - `README.md` ..... the directory with example sequence from dataset
  - `rl-am-vrptw` ..... source code of network
- `godeliver-planner` ..... directory of optimization planners
  - `godeliver-planner` ..... source code of planers
  - `benchmarking` ..... source code for running benchmarks
- `text` ..... the thesis text directory
  - `thesis` ..... the directory of LaTeX source codes of the thesis
  - `thesis.pdf` ..... the Diploma thesis in PDF format