Universiteit Leiden

Master Computer Science

A Reinforcement Learning-based Hierarchical Method for Route Planning

Name: Yuan Lin
Student ID: 2945142
Date: 20/07/2023

Specialisation: Computer Science: Data Science

1st supervisor: Yingjie Fan
2nd supervisor: Mitra Baratchi
3rd supervisor: Thomas Bäck

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The Travelling Salesman Problem (TSP), a well-known combinatorial challenge, has broad applications across various fields, including the last-mile problem. In 2019, Amazon launched a last-mile challenge, releasing a set of real-world data for public exploration.

To address the issue at hand, we have used a hierarchical architecture in our work. First of all, we use cutting-edge neural networks and reinforcement learning strategies for high-level sequence prediction. This enables us to produce a starting series of zones. The detailed sequence is then refined and generated at a lower level using a traditional local search technique. By combining these methods, we are able to take advantage of the benefits of neural networks and reinforcement learning for high-level sequence prediction as well as the speed and precision of local search strategies for adjusting the sub-level sequence creation.

Although the result obtained may not have met the originally anticipated quality level, it is closely compared to the results obtained using OR-Tools. This suggests that the proposed framework, with its robust features, shows potential for effectively addressing the problem. Expanding on the work, it offers intriguing insights that could contribute to future research efforts in this field.

KEYWORDS: Travelling Salesman Problem, Routing problem, Reinforcement learning, Pointer network

Contents

Chapter 1

Introduction

The routing problem has been studied for a long time and is used in a variety of fields such as logistics, supply chain management, transportation, and more. The logistics industry has reached a phenomenal height in recent years, especially with the growth of e-commerce. E-commerce revenue increased significantly over the past few years, reaching 1.42 trillion USD in 2017, 3.26 trillion USD in 2021, and is expected to reach 5.47 trillion USD by 2027(Statista). Walmart, Amazon, and Apple are the top three significant market players, in that order. In 2021, they generated 1.43 trillion in revenue collectively, which represents 43% of the market.

Without question, Amazon's business relies significantly on retail, which also happens to be its primary source of income. The goal of Amazon's 2021 Last Mile Routing Research Challenge, which uses historical driver routing data, is to investigate efficient methods for addressing real-world routing problems. Participants are given access to the stop coordinate, travel time to other stops, and information about each route's packages through the dataset. This thesis aims to correctly anticipate the trajectory, exploring to closely resemble the actual route taken by the driver. Instead of measuring accuracy, the similarity is used to evaluate the predicted route's quality.

The heuristic method is frequently employed for route planning as a combinatorial optimization task. For the given task, it has been shown that Simulated Annealing, Genetic Algorithms, LKH-3, and numerous other heuristic techniques perform effectively. From a different perspective, it is possible to discover underlying patterns in experienced drivers' implicit knowledge by employing machine learning, more specifically the deep learning method.

This idea and the nature of the routing data then lead to the framework being

divided into two parts. An internal zone label, developed by Amazon based on the physical planning area, is given to each stop along the route. To forecast the zone sequence, the high-level approach initially employs a neural network and reinforcement learning framework. The zone predicted plays a crucial role in determining the outcome. To balance the extensive computational labor in the first stage, a local search strategy is used in the second stage of the process to optimize the sequence of stops inside each zone, requiring less computational effort.

Therefore, the focus of our research is to determine whether the use of a hybrid technique to assess the effectiveness of a particular framework could produce better results. The outcome demonstrates that this framework might attain a quality that is comparable to the Google solver, but there is still potential for improvement compared to the method suggested by the team that placed tops in the competition.

The rest of the thesis is organized as follows. We discuss earlier research on this subject in chapter 2 of the paper. Chapter 3 introduces the challenge provided by Amazon. The structure of the project is shown in chapter 4. Results from the experiment using the Amazon dataset are shown in chapter 5. The discussion of this work and some suggestions for future work are included in chapter 6.

Chapter 2

Literate Review

Considering each route in the last mile task is assigned to a single driver, the problem is referred to as a simplified Traveling Salesman Problem (TSP). In this section, we cover the TSP problem and its relevant variations. The investigation of innovative techniques in machine learning strategies will thereafter be the main topic, with a particular focus on the Pointer Network. We will also talk about the traditional heuristic technique known as local search.

The Pointer Network will be used for high-level zone prediction, utilizing its abilities to perform higher-level sequence-generating tasks. On the other side, we will employ the local search method to locate the best solutions within certain zones for stop-level ordering.

2.1 Travelling Salesman Problems

The Traveling Salesman Problem (TSP) is a classic optimization problem. It involves determining the best path for a salesman to take in order to visit each city exactly once, return to the starting point, and visit all the cities. The goal of TSP is to reduce the overall cost of the route, which is often determined by distance or time. TSP is classified as an NP-hard issue, implying that it becomes harder to find an ideal solution as the number of cities in the problem instance grows.

The TSP is a symmetric problem for calculating distance in the case of a 2D Euclidean graph. However, in actual situations, when discussing how to calculate travel times, variables like traffic, road conditions, weather, etc., can have a significant effect on how long it takes to get from one place to another. As a result, the issue

becomes an asymmetric TSP (ASTP). In a revolutionary methodology approach given by Jonker and Volgenant (1983), the ATSP instance can be converted into a symmetric TSP instance using a 2-node transformation.

Some TSP alterations add new restrictions that go beyond the scope of the initial issue statement. One of the instances is the Vehicle Routing Problem (VRP), which Dantzig and Ramser (1959) first proposed in 1959. Multiple vehicles are assigned to a single route or location where they can refill without having to go back to the depot when doing a VRP task.

Traveling Salesman Problem with Time Windows (TSPTW) is another extension of TSP, which is sensitive to time constraints (Dumas et al. (1995)). The CTSP, or Capacitated Traveling Salesman Problem, is another variant. This variation takes into consideration additional capacity limits (Toth and Vigo (2014)). Just a few typical TSP types are mentioned above. Numerous versions of the routing problem exist, each with its own special considerations and strategy for the solution. An overview of these variants and solution approaches can be found in the works of Matai et al. (2010) and Applegate et al. (2011).

Meta-Heuristics and Machine Learning are the two main strategies that are frequently used to address these TSP variants in broad terms. Meta-Heuristics are a class of optimization algorithms that efficiently explore the search space to identify approximations of solutions. Machine learning techniques, on the other hand, use data-driven models to identify patterns and make forecasts or assessments based on historical information.

2.2 Pointer Network

We begin by discussing some recent studies. Machine learning techniques have become efficient ways to address combinatorial optimization problems (Bengio et al. (2021)). These techniques provide the ability to use real-life data and learn from patterns and trends to optimize solutions. It can increase the efficacy and efficiency of tackling complicated optimization issues by reducing the reliance on expert heuristics and instead relying on data-driven approaches.

The Pointer Network is a brand-new neural network architecture that Vinyals et al. (2015) proposed as part of their innovative methodological approach to the topic. By extending sequence-to-sequence models, this architecture improves their success in Natural Language Processing (NLP) applications and uses them to solve the routing issue. These tasks are fundamentally similar in that they both center on anticipating

sequential order.

The Pointer Network uses attention processes to handle sequential inputs and enables the output selection of particular elements from the input sequence. Without the requirement for padding, it can also manage outputs of varying lengths. By avoiding artificial assumptions, this approach represents a significant advancement in the field of neural networks. It provides a solid solution to sequential problems without relying on fixed-length representations. Additionally, the ability of the Pointer Network to use the attention mechanism as a pointer creates previously unimaginable options for handling a variety of complex tasks.

Based on the work of Vinyals et al. (2015), Bello et al. (2016) propose a framework to address TSP by employing the power of neural networks and reinforcement learning. The suggested structure, which uses a recurrent neural network (RNN) and the policy gradient method, produces encouraging results on 2D Euclidean graphs with up to 100 nodes. Without utilizing complex engineering or heuristics, the method produces solutions that are close to optimal.

Kool et al. (2018) leverages reinforcement learning as well to address the routing challenge. They present a model with attention layers that has benefits over the Pointer Network. They show methods to train this model using the REINFORCE algorithm and also show how to use a simple baseline that uses deterministic greedy rollout.

As TSP was the focus of the earlier study, Nazari et al. (2018) presents a detailed framework to address VRP as an extension of the Pointer Network. For difficult problems where the system representation evolves dynamically over time, the pointer network is constrictive. The proposed model directly uses the embedded inputs rather than the RNN hidden states since in VRP, the input order is irrelevant. This method uses a single policy model that has been trained to locate nearly optimum solutions for numerous issue situations of comparable size. By paying attention to reward signals and following the principles of feasibility, the model performs better than Google's OR-Tools optimization solver and heuristic techniques.

Mo et al. (2023) introduces a pair-wise attention-based pointer neural network, in contrast to the earlier research, which is based on 2D Euclidean space. Using the Amazon dataset, this network is intended to extract drivers' tacit knowledge. Using data from Amazon, this network is intended to capture drivers' tacit knowledge. The authors suggest a novel pair-wise attention-based pointer neural network in place of the traditional encoder-decoder design. This network utilizes a specific neural network to capture local pair-wise information for each pair of stops. The authors also present an

iterative sequence-generating method that comes after model training. This method identifies the first stop along a route with the lowest operational costs, increasing the total efficiency of the route.

Numerous studies have examined and expanded the applications of graph neural network and reinforcement learning methodologies in addition to the aforementioned works, Deudon et al. (2018), Joshi et al. (2019), and Khalil et al. (2017).

2.3 Local Search

The core concept behind local search is an optimization technique that incrementally enhances a given solution by investigating nearby solutions through minor adjustments. To optimize a certain objective function, it focuses on locating a local optimum inside the solution space. Local search is renowned for its effectiveness, simplicity, and capacity to deal with complex optimization issues, but it does not guarantee the discovery of the global optimum.

The 2-opt algorithm is a simple local search method to solve the TSP that is commonly used in optimization (Flood (1956)). By switching pairs of edges in the tour, the method reduces the overall distance and repeatedly improves a starting solution. The 2-opt algorithm seeks to find a locally optimal solution for TSP by experimenting with various combinations of edge swaps.

Expanding on this concept, Croes (1958) introduces the 3-opt algorithm. Similar to the 2-opt technique, the 3-opt iteratively swaps three edges in the tour to improve a starting solution. 3-opt gives more flexibility than 2-opt in terms of enhancing the effectiveness of a solution to combinatorial optimization issues. The benefit of employing the 3-opt algorithm is that it can explore a bigger solution space, possibly accomplish better gains, get around local optima, and increase the refinement of a TSP solution. However, it is crucial to keep in mind that the 3-opt algorithm incurs a substantially higher computational cost than more straightforward strategies like 2-opt because it requires analyzing a larger number of alternative swaps.

The k-opt concept is proposed by Lin and Kernighan (1973) afterward. Other local search methods included iterated local search (Lourenço et al. (2003)), stochastic local search (Hoos and Stützle (2004)), and guided local search (Voudouris et al. (2010)).

2.4 Technical Proceedings

This section provides an overview of the top three methodologies and conclusions that the Amazon Challenge prize winners reported in their proceedings papers (Winkenbach et al. (2021)).

First, the winning team suggests a simple local-search method based on penalties combined with learned routing constraints. They used the Concorde as the solver after first applying a typical transformation of ATSP to TSP to start the solution process. On top of this foundation, the zone limitation is introduced to produce a better outcome. Following data exploration, three constraints are used for a penalty-based local-search method to further capture the driver's tacit knowledge. While retaining the clustering of stops within each zone, the restrictions, including precedence, path, and neighbor constraints, are at the inter-zone level.

The second-place team introduces a tailored cost matrix and a hierarchical framework for improving TSP. The framework's upper level is focused on figuring out the ideal arrangement of zones, while the lower level is concerned with determining the optimal order of stops inside each zone. To start, they alter the route's time matrix and use it as an input to create an optimal zone sequence. The intra-zonal stop sequence is then resolved using a path-based TSP strategy. Additionally, they observe that, in the majority of situations, the driver makes all deliveries inside a particular zone before moving on to stops in another zone. This insight emphasizes how important it is to correctly determine the zone sequence because it is an essential factor in the process. The authors acknowledge that they were unable to increase the accuracy of predicting the first zone despite their proposed framework yielding encouraging findings. As a result, they continue to run into the problem of route reversal, in which the real route turns out to be the opposite of the estimated one.

Last but not least, the competitors who took third place used a strategy that involved solving the TSP on a transformed graph. They make use of the zone label's significance and alter the amount of time it takes to move between node pairs. The token ηij denotes the dissimilarity between the zone ids of the two given nodes i and j.

They also incorporate three discouragement multipliers, denoted as $a1$, $a2$, and $a3$, where $a1 < a2 < a3$. The parameter $a1$ is designed to discourage stops within the same zone, $a2$ represents the multipliers between two stops (i and j) whose zone IDs share three tokens, and $a3$ discourages all other pairs of stops that do not include the depot.

Chapter 3

Amazon Last Mile Challenge

In this chapter, we discuss the competition that Amazon has raised as well as the dataset's details and the method for assessment used to rate the route's quality. We choose a two-stage structure to address the routing problem by utilizing the unique features of this dataset. A thorough explanation of this framework will be given in the next chapter.

3.1 Dataset

The Last Mile Challenge offers a valuable dataset comprising historical driver routes from five major metropolitan areas in the United States during the year 2018. These areas include Seattle, Los Angeles, Austin, Chicago, and Boston (Merchan et al. (2022)).

The dataset is shown in Table 3.1 with data at the route, stop and package levels. This extensive dataset offers valuable insights on the whole delivery process, including the general routes traveled, the precise stops made, and exact details of individual shipments. There are 6,112 routes that have been labeled overall and are divided into low, medium and high quality. The dataset contains data for 17 depots in total. The majority of the routes start in Los Angeles.

Data field	Description
	Route information
Route ID	Unique and anonymized identifier of each route
Station code	Unique identifier for a delivery station where routes begin
Date	Date of route execution
Departure time	Time when vehicle leaves the station
Executor capacity	Volumetric capacity of vehicle
Stops	Each stop on route
Observed sequence	Sequence in which stops were visited
Route score	Quality of the observed sequence
	Stop information
Stop ID	Unique identifier of each stop on a route
Latitude/Longitude	Obfuscated coordinates of each stop
Type	Type of stop
Zone ID	Geographical planning area
Packages	Packages delivered at each stop
Transit time	Estimated transit time to every other stop on route
	Package information
Package ID	Unique and anonymized identifier of each package
Status	Delivery status of package
Time window	Start and end of time window, when applicable
Planned service time	Time that serving the package is expected to require
Dimensions	Length, width, and height of package

Table 3.1: High-Level Description of Data Fields Provided in the Research Challenge Data Set (Merchan et al. (2022))

3.2 Evaluation

When determining the route quality, Amazon takes into account how closely the submitted route resembles the previous driver's route (Amazon (2021)); this evaluation function is represented by equation 3.1.

It determines how closely the machine-generated route and the experienced driver's delivery route match each other. Both Sequence Deviation and Edit Distance with Real Penalty are used in the score function.

$$score = \frac{SD(A,B) \cdot ERP_{norm}(A,B)}{ERP_e(A,B)} \tag{3.1}$$

where SD stands for Sequence Deviation of A with respect to B, along with A is the

historically sequence and B is the prediction generated. ERP_{norm} denotes the Edit Distance with Real Penalty which with normalized travel times. ERP_e represents the number of edits prescribed by the ERP algorithm.

A score of 0 is given to the predicted sequences that exactly match the real sequence. The scores rise when the predicted sequence deviates further from the real sequence. Scores for completely random shuffles at each stop along the route taken by the driver typically range from 0.8 to 1.2.

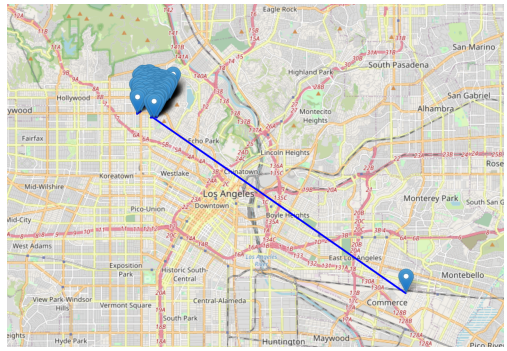
3.3 Preliminary Setting

After thoroughly analyzing the dataset, we made an intriguing discovery about the significant influence of the zoneID label on the generation of sequences. Take routeID $00143bdd - 0a6b - 49ec - bb35 - 36593d303e77$, shown in Figure 3.1, as an example. It has 119 stops in general and a depot located somewhat far from the client locations, as shown in Figure 3.1a. An intriguing pattern may be seen in Figure 3.1b. The driver appears to place more emphasis on finishing stops inside the same ZoneID than precisely adhering to geographic closeness. The color of the stops makes this obvious. (Although there is color repetition in the plot, adjacent zones may be distinguished by their unique colors.)

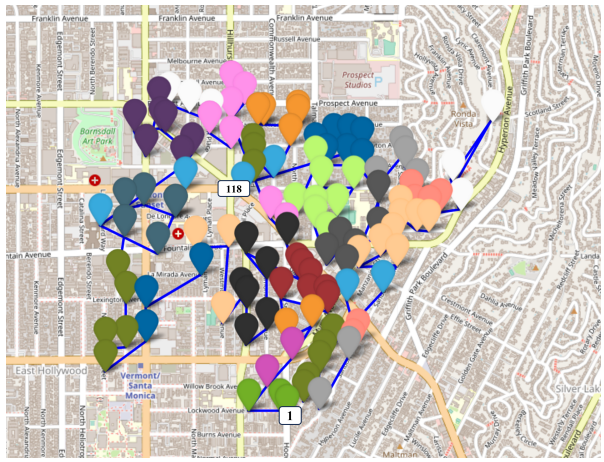
We propose a hierarchical technique to deal with this impact. We first construct the zone sequence, which is the order of the higher level. To develop a more efficient and precise sequence creation approach, we then list the stop order independently within each zone.

The problem would show itself as an asymmetric TSP at the stop level. However, the scope of the problem can be significantly decreased by using a hierarchical strategy. According to Figure 3.2, the number of zones is limited between 6 and 47, but the stop ranges up to 250 stops.

In the higher level, the travel time within each zone pair is calculated as follows `the average time of all stop pairs between two zones.` This problem transforms into a symmetric TSP problem after the stops are grouped according to the ZoneID label.

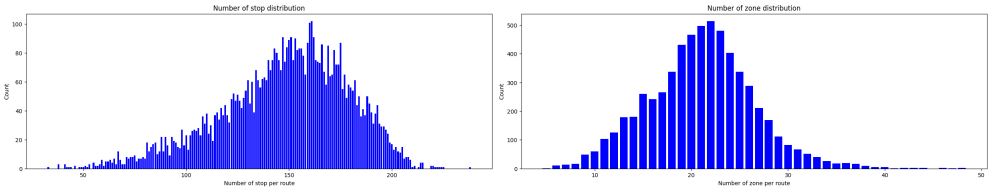


(a) Stops on a geographical map



(b) Stops group by ZoneID

Figure 3.1: RouteID00143$bdd-0a6b-49ec-bb35-36593d303e77$, with 119 stops, include depot



(a) Stop Distribution **(b)** Zone Distribution

Figure 3.2: Distribution of the stop and zone, range from 23 to 250 and 6 to 47, respectively

Chapter 4

Methodology

We introduce a framework for handling the TSP task in this chapter. The technique is divided into two parts to match the unique features of the Amazon dataset. Figure 4.1 displays a general framework flow diagram. We start with a set of arbitrary stops, which are then divided into zones using their corresponding labels, ZoneID. The first phase involves determining the zone sequence utilizing two methods, the Pointer Network and the Actor-Critic method from Reinforcement Learning. A local search strategy is then used to determine the stop sequence inside each zone. The route sequence is generated after the conclusion of these two processing phases. More information on the specifics of the many techniques used in the two stages will be provided in the following sections.

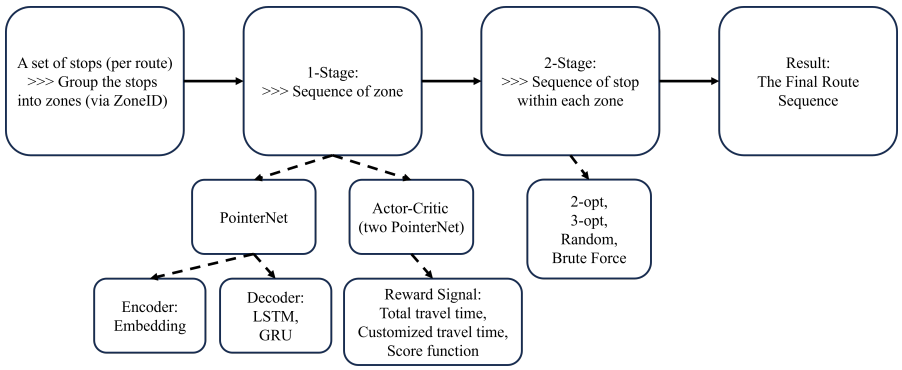


Figure 4.1: The framework flow

4.1 First Stage

The two main approaches we explore during the first phase are the Pointer Network and the Actor-Critic technique, built from policy gradient reinforcement learning.

Additionally, the Pointer Network is categorized as a sequence-to-sequence model that includes the Encoder and the Decoder as two essential parts. While the Decoder uses two different RNN structures, LSTM and GRU, the Encoder contains an embedding layer.

Lastly, in the context of the reinforcement learning approach, the reward signal is established as the objective within the problem. Three distinct reward signals are used and tested in this task to evaluate performance.

4.1.1 Pointer Network

We begin by constructing a sequence-to-sequence (seq2seq) model, consisting of two main components: an encoder and a decoder. In the Amazon TSP scenario, the input sequence has no impact on the outcome, thus the purpose of this model is to convert the input data and generate a route sequence. The model structure is shown in Figure 4.2.

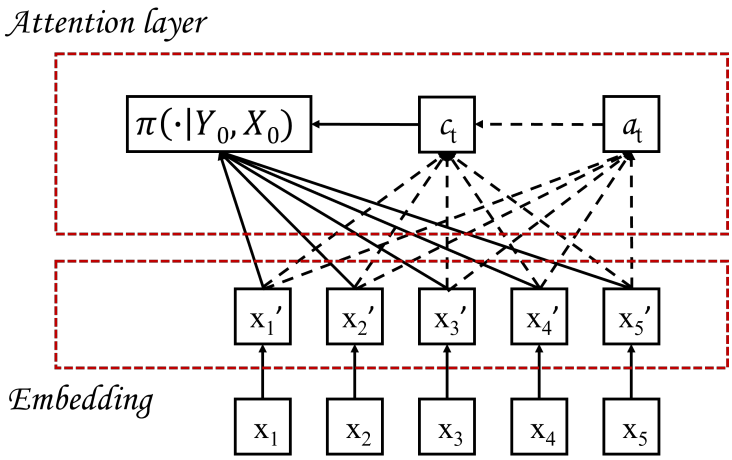


Figure 4.2: The structure of the neural network. (Nazari et al. (2018))

An input sequence is given to the encoder, an embedding layer in this case, and it turns it into an intermediate vector c_t that represents the input sequence. The

decoder then uses the intermediate vector c_t as input and decodes the data to produce the output sequence.

In contrast to the traditional seq2seq model, the incorporation of the Attention Mechanism dramatically enhances the performance of the Pointer Network, producing outstanding results. The role of the Attention Mechanism in this context involves combining or directly adding the encoder's hidden state to the decoder's hidden state, based on specific weights a_t. This additional information helps to improve the overall model's prediction accuracy.

The optimal policy π^* guarantees the generation of the optimal solution with a probability of 1. Our objective is to make π converge to π^* as closely as possible. To achieve this, we employ the probability chain rule to break down the probability of generating the sequence Y as follows:

$$P(Y|X_0) = \prod_{T=0}^{t} \pi(y_{t+1}|Y_t, X_t) \tag{4.1}$$

and

$$X_{t+1} = f(y_{t+1}, X_t) \tag{4.2}$$

is a recursive update of the problem representation with the state transition function f. Each component in the right-hand side of equation 4.1 is computed by the attention mechanism,

$$\pi(\cdot|Y_t, X_t) = softmax(g(h_t, X_t)) \tag{4.3}$$

where g is an affine function that outputs an input-sized vector, and h_t is the state of the RNN decoder that summarizes the information of previously decoded steps $y_0, ...y_t$.

In summary, the standard seq2seq model with an attention mechanism works as follows: the input sequence is encoded using the encoder, the encoded vector is used for attention, and finally, the input sequence is decoded using the decoder to obtain the desired result. On the other hand, Pointer Networks produce a probability distribution known as a pointer as their predictive output. Unlike the traditional seq2seq model with attention, which provides a probability distribution for the output, Pointer Networks provide a probability distribution for the input sequence.

The traditional attention method assigns weights to the components of the input sequence, which is equivalent to highlighting certain spots in the sequence. By selecting the input sequence entry with the highest weight, each projected element can be

located.

Essentially, Pointer Networks use pointers to create a probability distribution for the input text sequence, whereas the traditional attention mechanism of the standard seq2seq model focuses on producing a distribution of this type for the output. The two components that make up the Pointer Network will be thoroughly explained in the following paragraphs.

Encoder: Embedding

First, arbitrary sequences rather than sequential series can be used as the input sequence. The inputs are converted into a high-dimensional vector space using the embedding layer. Convolutional layer embedding is used to enhance training performance. This could be attributed to the method's success in resolving the initial high-dimensional input representation and extracting relevant data (Nazari et al. (2018)).

At decoder step i, we utilize a context-based attention mechanism with a glimpse, which extracts the relevant information from the inputs using a variable-length alignment vector a_t. In other words, a_t indicates the potential relevance of each input data point for the upcoming decoding step t.

Here the greedy strategy is applied, that is to say, at every decoding step, the node with the highest probability is selected as the next destination. Let x'_i be the embedded input i, and $h_t \in \mathbb{R}^D$ be the memory state of the RNN cell at the decoding step t. The alignment vector a_t is then computed as

$$a_t = a_t(x'_t, h_t) = softmax(u_t) \tag{4.4}$$

, where

$$u_t^i = u_a^T tanh(W_a[x_t^{i\prime}; h_t]) \tag{4.5}$$

The two vectors $x_t^{i\prime}$ and h_t are concatenated in the equation. Then the conditional probabilities by combining the context vector c_t, computed as

$$c_t = \sum_{i=1}^{M} a_t^i \bar{x}_t^i \tag{4.6}$$

with the embedded inputs, and then normalizing the values with the softmax function, as follows:

$$\pi(\cdot|Y_t, X_t) = softmax(\tilde{u}_t) \tag{4.7}$$

, where

$$\tilde{u}_t^i = v_c^T tanh(W_c[x_t^{i'}; c_t]) \tag{4.8}$$

In equations 4-8, v_a, v_c, W_a, and W_c are trainable variables.

Decoder: LSTM

Long Short-Term Memory (LSTM) is a specialized type of RNN. The LSTM paper was published in 1997 and was primarily designed to address the problems of vanishing and exploding gradients during training of long sequences (Hochreiter and Schmidhuber (1997)). From the name LSTM, it can be understood as a 'longer' short-term memory. In other words, LSTM improves upon the problem of short-term memory in conventional RNNs, allowing it to retain information from earlier points in time. In simple terms, compared to regular RNNs, LSTM exhibits superior performance when handling longer sequences.

The main difference between LSTM and general RNN lies in the calculation process of the model. The LSTM model performs more complex operations, as shown in Figure 4.3.

The original RNN had only one tanh operation, while LSTM has not only two tanh operations, but also three sigmoid operations σ and a multitude of additions and multiplications. It is precisely the alteration in the model's architecture that brings about changes in the computation process, allowing LSTM to have a longer memory than a typical RNN.

Before delving into the detailed operational steps of the LSTM model, let's first go through its core mechanism - the Cell State. The Cell State stores the model's current state, which signifies what the model has remembered so far.

Neural networks of the RNN type exhibit a recurrent characteristic, where the model's output at a given time step serves as the input for the next time step.

During computation at this time step, the model takes the Cell State C_{t-1} as input, representing its current memory. Following this computation, the model's Cell State changes, giving rise to C_t. C_t also becomes the input for the next time step of the model.

C_{t-1} and C_t can be different or the same, determined entirely by the model itself in terms of what information to retain and what to forget. We can observe that the Cell State C_{t-1} primarily undergoes two operations: multiplication and addition.

The inputs for the multiplication operation originate from two sources: the Cell State and the output of the sigmoid operation, which can be seen if we concentrate

on the multiplication section (depicted by the pink circle with x in the middle).

The sigmoid procedure usually yields a result of 0 or 1. Indicating whether the information should be maintained or forgotten after being element-wise multiplied by the Cell State, 0 implies that it should.

This mechanism determines what information in the Cell State should be forgotten or retained, and is known as the Forget Gate, akin to a large gate that decides who can pass through and who cannot.

In LSTM, there are three types of gates (Forget Gate, Input Gate, and Output Gate), and through these three gates, LSTM extends its memory capacity beyond that of a typical RNN.

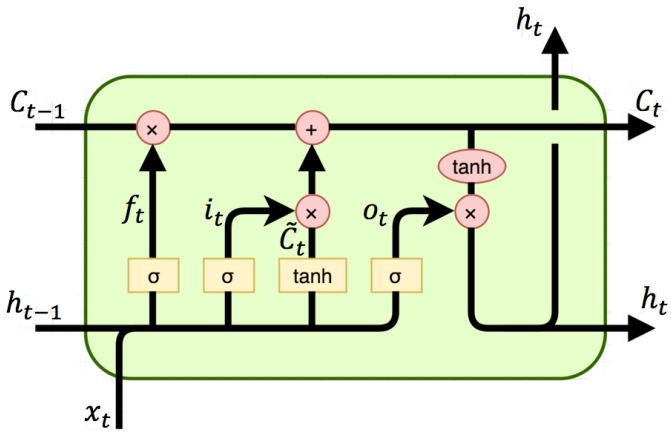


Figure 4.3: The structure of LSTM (Olah). The yellow box represents the Neural Network Layer, and the pink circle means the Pointwise Operation.

Decoder: GRU

To capture the information from the input data, the chosen RNN used in this task is Gate Recurrent Unit (GRU) as the decoder (Cho et al. (2014)). GRU stands among the RNNs and, when compared to LSTM, offers computational efficiency. The structure of GRU is explained in the following section.

Firstly, Figure 4.4 illustrates the GRU structure. It is similar to the input and output structures of standard RNNs. There is a current input x_t, and the hidden state passed down from the previous node, hidden state h_{t-1}, this hidden state contains the relevant information of the previous nodes. Combining x_t and $h_t - 1$ will give the GRU

access to the output of the node that is currently hidden node y_t and the concealed state that passed to the next node h_t.

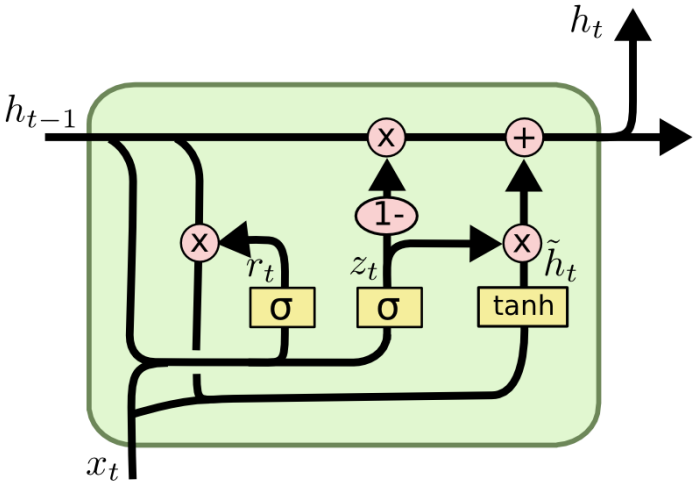


Figure 4.4: The structure of GRU (Olah).

Within GRU, the state of the previous transfer h_{t-1} and the input of the current node x_t get the two gating states.

$$r = \sigma(W_r[x_t; h_{t-1}]) \tag{4.9}$$

$$z = \sigma(W_z[x_t; h_{t-1}]) \tag{4.10}$$

, where r controlling the reset gate, z is an update gate for controlling updates. σ is a sigmoid function, through which the data transform into a value in the range of 0-1 to act as a gating signal.

After receiving the gating signal, start using the reset gate to obtain the data $h'_{t-1} = h_{t-1} \odot r$ (as a result of the reset operation). Next, combine h'_{t-1} with the input x_t through splicing. Finally, apply a tanh activation function to scale the data from range -1 to 1, as depicted in Equation 4.11.

$$h' = tanh(W[x_t; h'_{t-1}]) \tag{4.11}$$

here h' mainly contains the current input x_t data. targeted to h' adding to the current hidden state, which is equivalent to memorizing the state at the current moment.

Finally, we introduce the most critical step of GRU, which we can call the update

memory stage. At this stage, we simultaneously completed the phases of forgetting and remembering. We used the previously obtained update gate z.

$$h_t = (1 - z) \odot h_{t-1} + z \odot h' \tag{4.12}$$

Here the gating signal z ranges from 0 to 1. As the gating signal approaches 1, more data is retained, whereas as it approaches 0, more information is forgotten. One unique aspect is the simultaneous control of the forget and choose memories with a single gate z.

$(1-z) \odot h_{t-1}$: Indicates the selective $forgetting$ of the original hidden state. Here $1 - z$ can be thought of as a forget gate, forget (h_{t-1}) some unimportant information in the dimension. $z \odot h'$: Indicates the information about the current node h' selective memory. Similar to above, here the $(1 - z)$ empathy will forget h' some unimportant information in the dimension. Or, here we should regard it as the right h' dimensions to select certain information. $h_t = (1 - z) \odot h_{t-1} + z \odot h'$: Combined with the above, the operation of this step is to forget to pass down some dimensional information in h^{t-1}, and add some dimensional information input by the current node.

Select $(1 - z)$ and forgetting here z are related. That is to say, we will selectively forget the dimension information provided and to determine how much weight is forgotten (z), we will use the current input (h'). To keep the state consistent, adjust the weights in $(1 - z)$ to match.

4.1.2 Actor-Critic

After implementing the Pointer Network, we utilize the classic policy gradient method where the parameter θ represents the random strategy π. Usually, the policy gradient algorithm consists of two networks: actor and critic. The former predicts the probability distribution of the next action at each decision step, while the latter estimates the reward for a given state in any strength of the problem. The training target is the travel time, given the training set s, defined as follows:

$$J(\theta|s) = E_{\pi \sim p_\theta(\cdot|s)} L(\pi|s) \tag{4.13}$$

During training, the dataset is drawn from the distribution sampled. That is to say, the overall training target is obtained from the sampled samples.

$$J(\theta) = E_{s \sim S} J(\pi|s) \tag{4.14}$$

First Stage

We use the stochastic gradient descent method of policy gradient to optimize the model parameters. $b(s)$ represent baseline function, no dependent on π, is the estimation of expected travel time. The goal is to minimize the variance. Use a parameterized baseline to estimate the expected route length, E_π, which can improve the effect of the model.

Algorithm 1 actor critic training process (Bello et al. (2016))

Input/params: training set S, number of training steps T, batch size B
initialize: pointer network params θ, critic network params θ_v
for t = 1 to T **do**
  $s_i \sim SAMPLEINPUT(S)$ for $i \in \{1, ..., B\}$
  $\pi_i \sim SAMPLESOLUTION(p_\theta(\cdot|s_i))$ for $i \in \{1, ..., B\}$
  $b_i \leftarrow b_{\theta_v}(s_i)$ for $i \in \{1, ..., B\}$
  $g_\theta \leftarrow \frac{1}{B}\sum_{i=1}^{B}(L(\pi_i|s_i) - b_i)\nabla_\theta log p_\theta(\pi_i|s_i)$
  $\mathcal{L}_v \leftarrow \frac{1}{B}\sum_{i=1}^{B}||b_i - L(\pi_i)||_2^2$
  $\theta \leftarrow ADAM(\theta, g_\theta)$
  $\theta_v \leftarrow ADAM(\theta_v, \nabla_\theta, \mathcal{L}_v)$
end for
return θ

Reward signals

Three different reward signals are tested as part of the actor-critic approach implementation.

The first reward function is rather straightforward: it seeks to determine the route with the least amount of time spent traveling.

The second reward signal revolves around a customized total travel time. Here, the intention is to highlight the importance of the ZoneID label by adjusting the travel time using a weight that reflects the distance between each pair of zones.

The labeling rule, which may be thought of as having a macro and micro component, is used to compute the weight. Take one stop AD on routeID $00143bdd - 0a6b - 49ec - bb35 - 36593d303e77$ as an example, the stop information is shown in Table 4.1. The ZoneID of stop AD is $P - 12.3C$; the period is used as a separator, and the prior $P - 12$ and later $3C$ represent the macro and micro parts, respectively. As shown in Figure 4.5. If the two zone labels have a different alphabet letter, w_{macro1} denotes 1, otherwise 0. The alphabet letter in the macro section represents a more global representation of the geographical region. Additionally, form w_{macro2} using the variation in the number following the alphabet letter. The weight in macro is then

determined using the formula $w_{macro} = w_{macro1} + w_{macro2}$. When calculating the micro weight $w_{micro} = w_{micro1} + w_{micro2}$, first add up the difference of the integer before the alphabetic character w_micro1, and then use the Unicode code point for that character $w_micro2 = ord(a) - ord(b)$. The weight is finally determined using the formula $w = w_{macro} + w_{micro}$.

Field	Info
lat	34.099611
lng	-118.283062
type	Dropoff
zone_id	P-12.3C

Table 4.1: Stop-level information for stop AD in routeID $00143bdd - 0a6b - 49ec - bb35 - 36593d303e77$

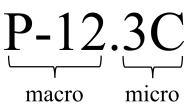


Figure 4.5: The naming structure of zoneID

The last one is the score function, which rates the level of quality of the route. This is regarded as one of the objective functions because the main goal of the thesis is to find a technique capable of producing a route that closely resembles actual driving behavior. Equation 3.1 provides a detailed description of the calculations.

4.2 Second Stage

The second stage uses four distinct ways to decide the order of stops within each zone after the zone sequence in the first stage is set. The final sequence for the complete route is generated after the second stage of the process is performed.

2-opt

To begin with, we employ a straightforward local search technique known as the 2-opt method. This method commences with a collection of zones and an initial random sequence.

Second Stage

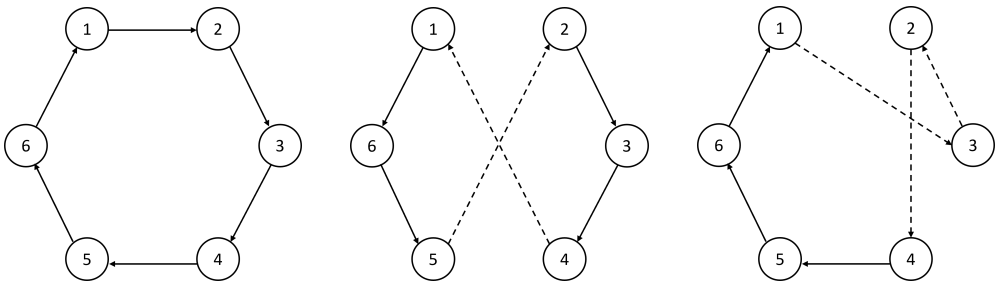
The algorithm employs iterative steps to perform local swaps, aiming to enhance the tour. Each possible pair of edges undergoes a systematic assessment to determine whether interchanging them would lead to a shorter total travel time. Refer to Figure 4.6b for a visual representation of the swap process.

Edges are swapped, and the tour is adjusted as required whenever such a swap would genuinely decrease the overall duration. If this is not the case, the particular pair of edges remains unaltered. This process is repeated until the termination condition is satisfied. Edges are swapped and the tour is adjusted as required whenever such a swap would genuinely decrease the overall duration. If this is not the case, the particular pair of edges remains unaltered. This process is repeated until the termination condition is satisfied.

3-opt

The second approach is an extension of the 2-opt, the 3-opt algorithm. It starts with an initial sequence and calculates its cost, representing the total travel time. It systematically explores all possible combinations of three edges in the tour. As shown in Figure 4.6c, a 3-opt swap operation is carried out for each triplet to provide a new candidate sequence.

The travel time of the new candidate is then assessed using the cost function. The new candidate becomes the new best candidate if its trip time is less than that of the previous best candidate. This process is repeated for all triplets until no further improvements are made. Finally, the algorithm returns the best sequence and its corresponding travel time, representing an optimized solution with a reduced travel time.



(a) Route with six stop **(b)** 2-opt swap movement **(c)** 3-opt swap movement

Figure 4.6: Example of 2-opt and 3-opt

Random order

In addition to the local search, we employ the random assignment method for stops within the zone for comparison. The rationale behind this is exemplified by routeID $00143bdd - 0a6b - 49ec - bb35 - 36593d303e77$, where the number of stops in a zone ranges from 1 to 7, with 3 stops being the most common count. Consequently, the assumption is that the order of stops has a minimal impact on the final result since it affects only a small number of stops.

Exhaustive search

The concept of brute force is straightforward. Given the relatively small scale of the problem, employing the brute force approach becomes viable as it allows us to explore all possible scenarios.

Chapter 5

Experiment

In this chapter, several experiments are carried out within the computational framework using real-life examples. The two-stage architecture is implemented, and the benchmarks are used to evaluate the model's efficacy. The experiment code can be found in GitHub [1]

5.1 Experiment Design

We utilize a policy-based Actor-Critic technique in the first phase, with the embedding layer as an encoder component of the neural network throughout. For the decoder part, we experiment with two different types of RNNs: LSTM and GRU.

Out of the four methods considered for the second stage, contrary to expectations, preliminary testing on a small dataset highlighted that even with limited scope, brute force calculations could lead to significant computational explosiveness when applied to stop-ordering in the second step. Both of the local search techniques significantly outperformed random selection among the remaining three strategies. Based on the explanation in the previous section and the experimental results, 3-opt demonstrated a superior ability to overcome local optima compared to 2-opt. Consequently, the final implementation of all second-stage operations utilized the 3-opt technique.

According to Figure 3.1a, the depot looks to be somewhat distant from consumers, and neither the first stop nor the final stop is the closest to it. Because of this, the depot was not taken into account while building the sequence; instead, we simply looked at the client's location.

[1] https://github.com/annalin25/Thesis.git

As part of the experiment, we also look at the route's quality label, the *Route score* in Table 3.1's Route-level information, which is denoted by a low, medium, or high rating and is utilized as a weight for determining the reward. This characteristic is left out of the final experiment because it barely affects the results.

At the close, the differences in the first stage - specifically, the pairings of LSTM, GRU, and the three different reward signals - were the main focus of our comparison research.

5.2 Benchmark

Three benchmarks are used for comparison to assess the model's performance.

OR-Tools

The first one is OR-Tools, which is an open-source software developed by Google for combinatorial optimization. It provides solvers for many problems, such as Vehicle routing, Constraint programming, Linear and mixed-integer programming, and Graph algorithms. We employ the vehicle routing solver in this project.

The outcome provides both the route and the associated time spent, utilizing the cost matrix (depicting travel time between each pair of stops) and considering the depot's location. Subsequently, the route is passed through the scoring function to assess its performance.

Label-based approach

As mentioned in the previous section, it is found that ZoneID significantly affects the way people drive. The second criterion is hence the presumption that the label naming requirements are strictly adhered to along the route. The driver will complete the zone with the same macro sign as the micro group, as shown in the structure in Figure 4.5. Within the relevant macro- and micro-groups, the visiting order can be either ascending or descending.

Simplest form of the Two-stage Framework

Finally, we create the simplest form as a baseline using the same two-stage architecture. The first step makes use of a Pointer Network, an embedding layer encoder, and an LSTM cell as a decoder. The stops are randomly arranged in the second stage.

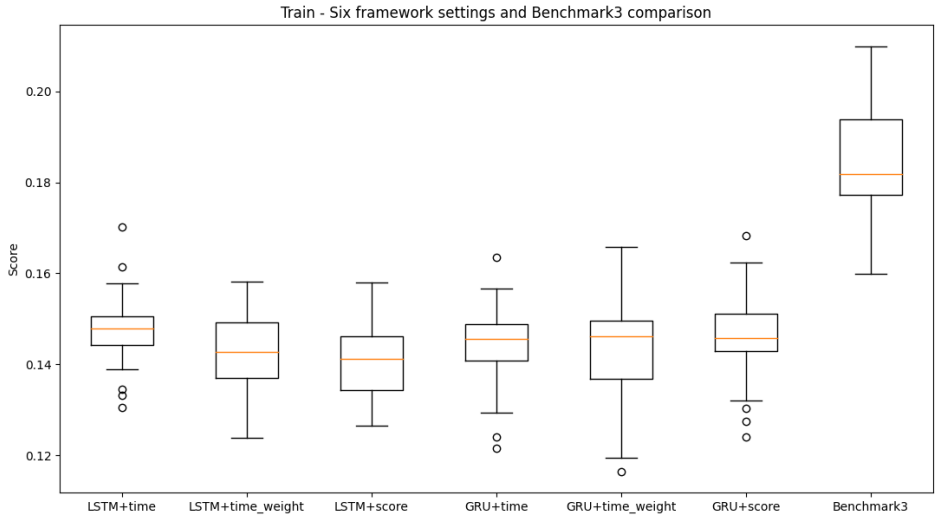
5.3 Result

In this section, we present the results of combining the framework in various ways with three benchmarks. We randomly select 4890 routes for model training and 1222 routes are used to evaluate performance. The parameters of the neural network are shared by several combinations of models, with hidden sizes of 256, 2 layers, dropout of 0.2, and batch size of 128. We train the model using the Adam optimizer with a learning rate of 0.0001 and 30 training epochs.

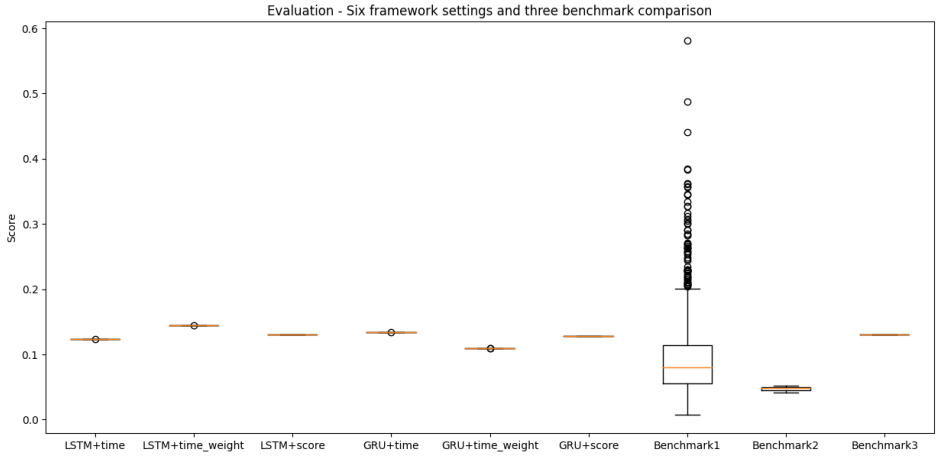
In the training phase, we compare several settings in the first stage, using 3-opt for the stop sequence in the second stage. We use the reinforcement learning approach in the first step, which includes two Pointer Networks. Three reward signals are examined, and two RNN cells are employed in the Pointer Network's Decoder component. As shown in Figure 5.1a, $LSTM + time$ denotes that the neural network employs an LSTM cell and with the goal of reducing the overall routing time, the travel time serves as a reward signal. GRU means the other use type of the RNN, $time\_weight$ stand for the second reward signal in Chapter 4.1.2, the travel time with customized weight. $score$ is the last reward function mentioned. In total, two RNN cells and three reward signals generate six different combinations of the framework; these are compared with $Benckmark3$, the simplest form of the two-stage method, while the other two benchmarks do not require training progress, will be demonstrated in the evaluation section only.

Following is the evaluation phase. There are still six model configurations and three benchmarks, making a total of nine comparison methods. The two additional benchmarks $Benchmark1$ and $Benchmark2$ in Figure 5.1b represent the OR-Tools and the label-based approach, respectively.

Comparing the result, the performance in the training and testing phases exhibits some differences. In the first, the combination of $LSTM + score$ achieves the best performance, while in the second, GRU together with cus_time demonstrates the superior performance. As expected, GRU outperforms LSTM and is relatively simpler in terms of computation. Regarding the three different reward signals, they yield diverse outcomes with different RNNs. However, among the six variants, $GRU + time\_weight$ delivers the best performance, closely resembling the results of $Benchmark1$, Google OR-Tools. Although the results still fall short of matching the label-based approach, it once again underscores the significance of label prediction. All six methods surpass $Benchmark3$ and perform well even in the fundamental two-stage setup.



(a) Train



(b) Evaluation

Figure 5.1: The result of train and evaluation on the models

Chapter 6

Conclusion

In this thesis, we present a two-stage framework that integrates machine learning and heuristic methods for tackling the TSP. We utilize the Pointer Network to address the sequence of higher-level zones, as defined by the zone label. Subsequently, at the stop level, we employ a classical local search method known as 3-opt.

Two findings emerged from the model adjustments. First, contrary to previous studies that employed LSTM, we found that GRU outperformed in this particular task. The simplicity structure of GRU may be a contributing factor, making it well-suited for handling simple inputs. Secondly, the evaluation function did not exhibit a significant impact on the model adjustments. Initially, the assumption was that the evaluation function would define the quality and guide the model's learning. However, in this scenario, the travel time emerged as the more crucial factor.

During the experiment, it was observed that the zone label remains the most influential factor in determining driver behavior. Tacit knowledge, on the other hand, appeared to be more of an outlier in the dataset. Furthermore, having close to 5,000 data points might still fall short in terms of training an effective model. The initial assumption was to use the reward signal to teach the model to learn driver behavior with a small amount of data.

Another potential reason for the model's performance could be attributed to the depot. Preliminary findings indicate that the depot is situated at a considerable distance from all the stops and is not necessarily connected to the closest stop or zone. As a result, it is excluded from the training process. However, this can lead to a scenario where the sequence of zones is correctly formed into a close loop, but the model incorrectly incorporates the depot as a connection to one of the zones.

Consequently, the final route sequence may end up being divided into two parts, with the depot acting as a separation point between them.

In summary, the framework appears to possess substantial interpretability, given that OR-Tools display a certain degree of instability, as depicted in Figure 5.1b. Nonetheless, within the context of the two-stage framework, the poorest performance would still be only less than 0.3, signifying relative robustness. Consequently, with the identification of the suitable configuration, confidence is held in the framework's potential to deliver impressive results.

For further real-life TSP studies, explore multilayer approaches, designing architectures with abstraction for complex routing patterns. Furthermore, combine heuristics with neural networks or reinforcement learning for better solutions. Heuristics offer initial efficiency, neural networks refine, combining for robust TSP outcomes.

Bibliography

Amazon. Amazon routing challenge scoring, 2021.

D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. The traveling salesman problem. In *The Traveling Salesman Problem*. Princeton university press, 2011.

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

Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

G. A. Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

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

M. Deudon, P. Cournut, A. Lacoste, Y. Adulyasak, and L.-M. Rousseau. Learning heuristics for the tsp by policy gradient. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15*, pages 170–181. Springer, 2018.

Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. *Operations research*, 43(2):367–371, 1995.

M. M. Flood. The traveling-salesman problem. *Operations research*, 4(1):61–75, 1956.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

H. H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

R. Jonker and T. Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161–163, 1983.

C. K. Joshi, T. Laurent, and X. Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

W. Kool, H. Van Hoof, and M. Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

R. Matai, S. P. Singh, and M. L. Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, 1, 2010.

D. Merchan, J. Arora, J. Pachon, K. Konduri, M. Winkenbach, S. Parks, and J. Noszek. 2021 amazon last mile routing research challenge: Data set. *Transportation Science*, 2022.

B. Mo, Q. Y. Wang, X. Guo, M. Winkenbach, and J. Zhao. Predicting drivers' route trajectories in last-mile delivery using a pair-wise attention-based pointer neural network. *arXiv preprint arXiv:2301.03802*, 2023.

M. Nazari, A. Oroojlooy, L. Snyder, and M. Takác. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

C. Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed: 2015-08-27.

OR-Tools. L. perron, v. furnon, or-tools. `https://developers.google.com/optimization/`.

Statista. statista - ecommerce (worldwide). `https://www.statista.com/outlook/dmo/ecommerce/worldwide/`. Accessed: 2023-07.

P. Toth and D. Vigo. *Vehicle routing: problems, methods, and applications*. SIAM, 2014.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

C. Voudouris, E. P. Tsang, and A. Alsheddy. Guided local search. In *Handbook of metaheuristics*, pages 321–361. Springer, 2010.

M. Winkenbach, S. Parks, and J. Noszek. Technical proceedings of the amazon last mile routing research challenge. 2021.