Xi'an Jiaotong-Liverpool University

西交利物浦大学

Learning-based Heuristics for Solving Capacitated Vehicle Routing Problems

基于机器学习的启发式算法求解有能力约束的车辆路径调度问题

Author: Yixuan Chen

ID: 1822444

Supervisor: Prof. Jianjun Chen

Department of Applied Mathematics

This dissertation is submitted for the degree of
BSc Applied Mathematics

School of Science May 2022

Abstract

The purpose of this paper is to discuss how to solve combinatorial optimization issues, specifically the capacitated vehicle routing problem (CVRP). Recent research in deep learning for routing problems has concentrated on building heuristics, the solutions to which are still far from ideal. By iteratively refining a solution, improvement heuristics have a tremendous deal of promise to close this gap. On the other hand, traditional improvement heuristics control their operators with manually specified criteria, which may limit their effectiveness. For the purpose of figuring out better heuristics for solving routing problems, I've developed a framework for using deep reinforcement learning. I created a policy network based on self-attention to guide the selection of the next operator. The learnt policies can be further strengthened through simple diversification measures. Additionally, the rules are highly generalizable to a variety of issue sizes, initial solutions, and even real-world datasets.

Keyword: Capacitated Vehicle Routing Problem (CVRP), Deep Reinforcement Learning, Heuristics

本文的目的是讨论如何解决组合优化问题，特别是有容量限制的车辆路径问题。目前对于路径问题的深度学习研究主要集中在直接构建路径上，但其解决方案仍不理想。通过迭代完善解决方案，改进启发式在缩小这一差个方面有巨大的前景。另一方面，传统的改进启发式方法用人工指定的标准来控制其算子，这可能会限制其有效性。为了弄清解决路径问题的更好的启发式方法，我开发了一个使用深度强化学习的框架。我创建了一个基于自我关注的政策网络来控制下一个算子的选择。学习到的政策可以通过简单的多样化措施进一步加强。此外，这些规则对各种问题规模、初始解决方案、甚至真实世界的数据集都具有高度的通用性。

Acknowledgements

Many individuals, including my supervisor, family, and friends, have provided me with a great deal of assistance and support throughout the course of my final year project.

First and foremost, I want to express my gratitude to my supervisor, Dr. Jianjun Chen. Dr. Jianjun Chen provided me with a lot of advice and help during this endeavor. He also provided me with a lot of encouragement, which gave me the confidence I needed to finish this job. It may be claimed that without Dr. Jianjun Chen's supervision and support, this final year research and thesis would not have been completed.

Secondly, I want to express my gratitude to my family and friends. It was they who supported me and gave me the guts to persist in accomplishing this final year project when I was uncertain about it.

Finally, I am very grateful to myself for choosing this direction of research and earnestly completing the preliminary exploration in this field, which has laid a good foundation for my postgraduate career. Through reading a lot of papers in this direction, I have gained a preliminary understanding of this field and look forward to making some achievements in it in the future.

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

A	Action space
a	Action
C	The Capability of the Vehicle
c	Travelling cost
e	Effect of the action
N	Number of customers
r	One route in the solution
S	Solution set
s	State
T	Total step
V	The sets of customers

Subscripts

i	index of customer node
j	index of customer node
m	number of routes in the solution
t	current step

Acronyms / Abbreviations

CVRP (Capacitated Vehicle Routing Problem

DL Deep Learning

OR Operational Research

RF1 Reward Function 1

RF2 Reward Function 2

RL Reinforcement Learning

TSP Traveling Salesman Problem

VRP Vehicle Routing Problem

Chapter 1

Introduction

1.1 Motivation, Aims and Objective

The Vehicle Routing Problem (VRP) is a well-studied topic in Transportation and Operations Research [26]. This is largely due to the VRP's straightforward fundamental structure [31] and extensive applicability to real-world challenges. The VRP versions have developed in several spheres of human life [16, 9], most notably in the logistic business, which is thriving as a result of globalization. Effective route scheduling may result in significant cost savings (usually between 5% and 20%) [28], which is critical for boosting distribution efficiency [1] and increasing organizations' economic advantages. Capacitated Vehicle Routing Problem (CVRP) is an example of a class of routing issues that fall under the purview of combinatorial optimization. Due to their NP-hardness [29], even though we solve them on a daily basis, attaining satisfying results is still a challenge. Classical approaches to routing issues can be divided into three categories [28]: 1)Exact method, based on the branch-and-bound framework, which theoretically ensures finding the best solution but is actually confined to tiny instances due to their exponential complexity in the worst case, are frequently used to build precise procedures [18, 23]; 2)Suboptimal solutions can be found in polynomial time using approximation methods, however, they may only exist for certain situations and still have poor approximation ratios [1, 7]. 3)Heuristics are the most popular method for addressing routing difficulties in the real world. Despite the lack of theoretical guarantees about the quality of the answer, heuristics frequently discover suitable solutions in an acceptable amount of time [11]. Heuristics, however, must be developed via a lot of trial and error, as well as rely heavily on the intuition and expertise of human specialists [15].

Recently, there has been an increasing interest in using deep learning (DL) to develop heuristic methods for solving routing issues automatically. The basic logic is twofold: 1)

Problems may have identical structures and simply differ in data distributions; 2)DL models can find these patterns to produce new algorithms that are superior to the ones designed by humans through the application of supervised or reinforcement learning [3]. Some solutions use the sequence-to-sequence (Seq2Seq) models and treat routing difficulties as if they were a sequence generating problem [33]. These approaches are capable of learning high-quality heuristics based on intricately constructed deep structures such as recurrent neural network (RNN) [32, 2, 25], and attention mechanism [17, 14, 8].

1.1.1 Motivation

It has been demonstrated that learning-based approaches may produce solutions that are relatively excellent in quality in a short period of time after being taught on a large number of issue situations. However, when evaluated against the same benchmark examples, these learning-based approaches are unable to beat the current state-of-the-art method, LKHR3 [13], which is a penalty function-based version of the traditional Lin-Kernighan heuristic [20, 12]. I was inspired by this line of research to investigate a framework that blends the strength of Operations Research (OR) heuristics with the learning capabilities of machine learning (RL in particular).

1.1.2 Aims and Objective

When test examples are created from the same distribution as training instances, machine learning may rapidly acquire the ability to solve a class of problem situations. While search algorithms and other conventional techniques might be fairly effective, they may need substantial processing, which can be rather time-consuming. Hyper-heuristics is a related field of research that is defined as "an automated search method or learning mechanism for selecting or developing heuristics for solving computational search issues" [5]. Rather than building a high-level approach that does not need knowledge of the complexities of low-level heuristics, I am particularly interested in developing an integrated system that maximally harnesses the power of OR operators and learning capabilities.

1.2 Literature Review

Over the last 60 years, research on the VRP has advanced at a breakneck pace. Since 1973, numerous versions of VRP have emerged to address more complicated operational difficulties, including location routing problems, cyclical VRP, dynamic VRP, VRP with time windows, inventory routing problems, fuzzy VRP, open VRP, multi-storehouse VRP, and

shared VRP.[4] Over the last two decades, scholars have tended toward a greater emphasis on sustainable VRP, which takes economic, environmental, and social factors into account.[10] Since 2000, other variations of sustainable VRP have evolved, including green VRP, electric vehicle VRP, pollution routing problems, drone routing problems, and crowdsourced delivery[19]. However, because these solutions to these variants are predicated on increasing the efficiency of the methods used to solve the underlying CVRP, there is continued interest in CVRP algorithms and the incorporation of new approaches from recent years.

The use of deep learning and RL to handle combinatorial optimization issues has been widely studied in recent years. A number of publications have been dedicated to the study of routing issues, particularly TSPs and VRPs [27, 24, 21, 22]. With the exception of Chen & Tian's study [6], the majority of these studies take an end-to-end strategy, which entails explicitly creating a solution from start. Sequence-to-sequence models motivated Vinyals et al. to first present the Pointer Network to solve TSP [32]. They employ an attention model to supervised learn the order of various nodes. Bello et al. suggested developing a reinforcement learning technique for training the Pointer Network without the necessity for supervised solutions, in which the framework learns the best policy from issue cases [2]. Nonetheless, the RNN-based encoder in the Pointer Network embeds the input sequence's sequential information, which the output sequence should be indifferent to. Thus, Nazari et al. enhance the Pointer Network by linearly translating each node's information to a high-dimensional space with common parameters, invariant with regard to the input sequence, and extending it to solve VRP [25]. Kool et al., inspired by the Transformer design [30], replaced the RNN-based sequential structures in Seq2Seq models with attention modules in both the encoder and decoder, achieving improved performance on both TSP and CVRP[17].

All of the approaches above learn construction heuristics that output just one solution and are capable of significantly outperforming standard non-learning based construction heuristics. However, the quality of the solution is still far from perfect. Chen and Tian propose a paradigm for VRP called NeuRewriter [6]. They establish a rewriting rule set and train two policy networks to achieve the next state: a region-picking policy and a rule-picking policy. The objective is to develop a cost-effective sequence of steps toward the solution given an initial solution. It outperforms Kool's model, the most effective technique for learning construction heuristics, on CVRP. However, it requires two policies to decide on the rewrite region and solution selection independently and depends on complicated node attributes and specialized local operations.

Chapter 2

Methodology

2.1 Preliminaries

CVRP consists of a depot and a collection of N consumers $V = \{1, \ldots, N\}$. Each customer $i \in V$, has a demand d_i that must be met. A vehicle that originates and terminates at the depot can service a group of clients as long as the overall consumer demand does not exceed the vehicle's capacity. The travelling cost $c_{i,j}$ is the cost of transporting a vehicle from node i to node j, where $i,\ j \in \{0,\ 1,\ \ldots,\ N\}$. (where the depot is denoted by node 0 for convenience). The purpose is to determine the least expensive routing plan that serves all consumers while adhering to vehicle capacity limits C. Toth and Vigo [28] provided below an integer programming formulation for CVRP:

$$\min_{x_{i,j}} \quad \sum_{i \in V} \sum_{j \in V} c_{i,j} x_{i,j},\ \ x_{i,j} \in 0,1,\ \forall i, j \in V$$

$$s.t. \quad \sum_{i \in V} x_{i,0} \ = \ K \tag{2.1}$$

$$\sum_{i \in V} x_{i,0} \ = \ K \tag{2.2}$$

$$\sum_{i \in V} x_{i,j} = \ 1,\ \forall j \in V \setminus \{0\} \tag{2.3}$$

$$\sum_{j \in V} x_{i,j} = \ 1,\ \forall i \in V \setminus \{0\} \tag{2.4}$$

$$u_i - u_i + C x_{i,j} \leq C - d_j,\ \forall i, j \ \in V \setminus \{0\},\ i \neq j, s.t.\ d_i + d_j \leq C \tag{2.5}$$

$$d_i \leq u_i \leq C,\ \forall i \in V \setminus \{0\} \tag{2.6}$$

where K is the number of vehicles available (for the CVRP under consideration, $K = N$). Constraints (2.1) and (2.2) state the depot's in and out degrees, whereas constraints (2.3) and (2.4) state that each client is only visited once, respectively. Vehicle capacity needs are imposed by constraints (2.5) and (2.6).

Route : A route r is a collection of nodes visited in succession. Each route starts from node 0 and visits a subset of customers $I \subseteq V$ in sequence. For instance, $[0,2,5,3,1,4,0]$ is a travelling path that begins at the depot, proceeds sequentially to nodes 2, 5, 3, 1, and 4, and then returns to the depot.

Solution : A solution s is a set of routes $\{r_1, \ldots, r_m\}, m > 1$ that ensures each client gets visited precisely once and that the overall demand along each route is less than the vehicle's capacity.

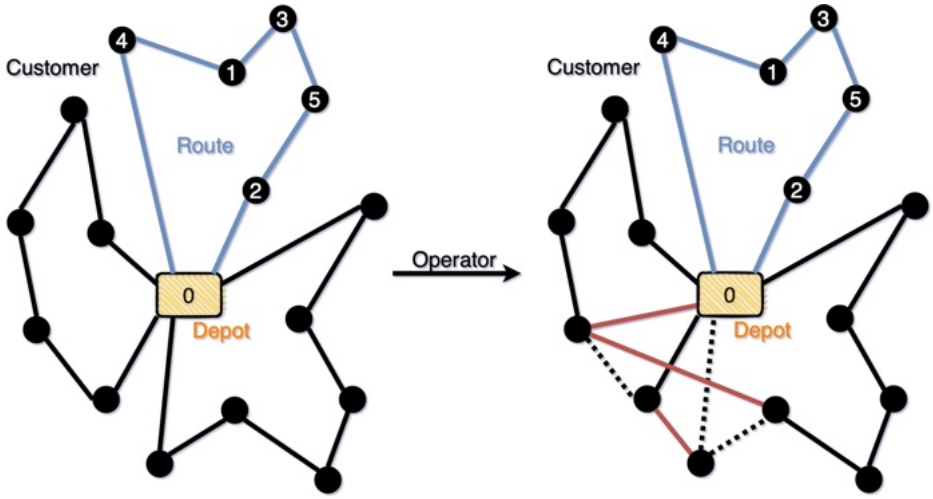

Figure 2.1 An illustration of CVRP

Operator : A mapping from one solution to another is referred to as an operator. Rather than starting from scratch, we use operators to iteratively enhance or disrupt the solution in this study. All operators are described in detail in 2.1.1 and 2.1.2. As shown in the Figure 2.1, the solution to this problem instance consists of the three routes, with the blue one being the sample path. Red lines replace dashed lines in a new solution once an operator is applied.

2.1.1 Improvement operators

- **Intra-route**

  2-Opt: Remove two edges and reconnect their endpoints.

  Swap: Exchange two customers in the route.

  Relocate: Move a customer in the route to a new location.

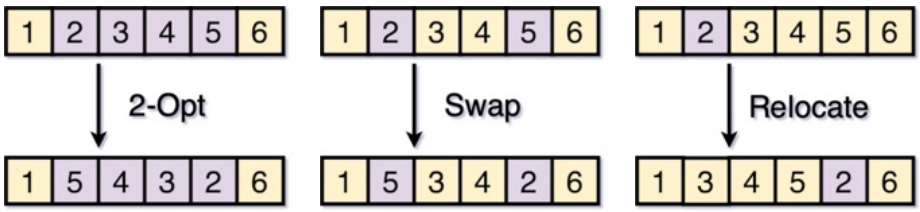

Figure 2.2 2-Opt Figure 2.3 Swap Figure 2.4 Relocate

- **Inter-route**

  Cross: Exchange two customers in two routes.

  Swap2: Exchange 2 or 3 consecutive customers between two routes.

  Relocate2: Move segments of length l $(l = 1,\ 2,\ 3)$ from one route to another.

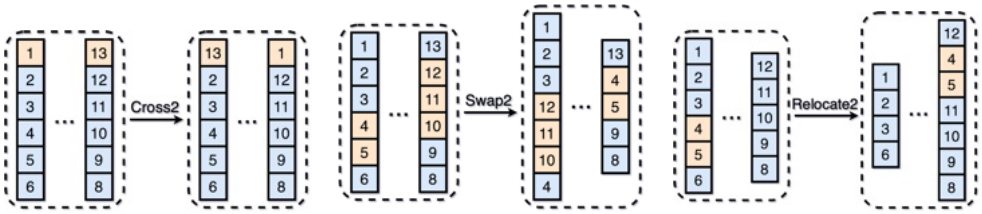

Figure 2.5 Cross Figure 2.6 Swap2 Figure 2.7 Relocate2

2.1.2 Perturbation operators

Random-permute: Dismantle m routes randomly and reconstruct them by visiting impacted consumers in a random sequence as Figure 2.1.

Random-swap: Randomly exchange m pairs of nearby customers between two routes.

2.2 RL Formulation

In this section, I will formally introduce the RL framework I use, including the main components of the system and the design ideas for each part. Figure 2.8 illustrates the entire framework. Using the Capacitated Vehicle Routing Problem as an environment, the Agent makes a choice of improvement operator based on the state derived from observing the environment.

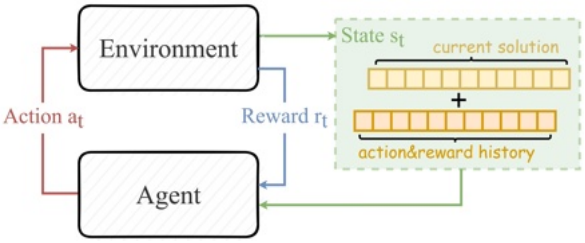

Figure 2.8 RL framework

For an initial feasible solution generated at the beginning, the framework improves the solution through continuous iteration. In the process, all constraints are satisfied. After T (pre-defined parameters) steps, the algorithm stops and the solution with the minimum distance is selected as my final solution. Since the final converged solution may be a locally optimal solution during the training process, I artificially set the rule of calling the perturbation operator, i.e., perturbing it when all operators reach convergence. This allows the exploration of all feasible solution spaces while ensuring the feasibility of the solutions.

State : At time step t, the state s_t represents a solution to an instance. Each state contains information from the solution and the running history. The location and demand of each customer are instance-related parameters, thus they are constant across solutions. The specific features of the solution are determined by the current trip itinerary. A customer's nearby nodes, as well as relevant distances, are computed for each customer given a solution. The running history contains recent actions as well as their effects.

Action : Intra-route operators and inter-route operators are two categories into which the acts fall. When an intra-route operator focuses on decreasing one route's overall travel distance, inter-route operators focus on shifting consumers across many routes to minimize total trip distance. Consider that the same operator may be used with various arguments and yet be called an action a_t. The same operator with various parameters may perform differently for a given issue (or even solution), therefore it's preferable to consider them as independent operations and allow the RL model to figure out how to utilize them most effectively.

Reward : Our ultimate aim is to maximize the improvement of the initial solution within step restriction T. To this end, I have experimented with a number of reward designs, two of which have produced satisfactory results, as well as unique patterns of operator sequences. One (RF1) is that the impact e_t of the action a_t is $+1$ if the action caused the total distance to decrease, and -1 if the action caused the total distance to remain unchanged. Another (RF2) is that based on improved distance, the impact e_t of the action a_t is 10 times improved distance.

Policy network : Starting with step s_0, the stochastic policy selects an action a_t at each step t that will result in step s_{t+1}, until the step limit T is reached. We update the gradient of a policy with a baseline function $b(s)$ using the well-known REINFORCE technique [30].

$$\nabla_\theta J(\theta|s) = \mathbb{E}_{\pi\sim p_\theta(\pi|s)}[(L(\pi|s) - b(s))\nabla_\theta \log p_\theta(\pi|s)]$$

A policy network generates a list of action probabilities for each action described in the state. This procedure is described by the following probability chain rule:

$$P(s_T|s_0) = \prod_{t=0}^{T-1} \pi(a_t|s_t)$$

As shown in the Figure 2.9, the solution-specific properties and running history of the input are translated into a D-dimensional embedding that is supplied into an attention network. The attention network's output is concatenated with a sequence of recent actions and their associated consequences. Finally, the concatenated values are input into a network composed of two fully connected layers, yielding $|S|$ action probabilities for the set of actions A.

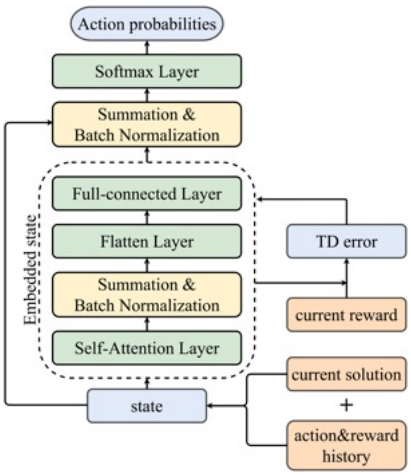

Figure 2.9 Network Structure

2.3 Code Description

This section will describe the implementation of some important parts of the code and their rationale.

2.3.1 Initial Solution

The nearest insertion method is used as the method for generating the initial solution. That is, for customer $i \in V$, the vehicle is visited from depot, starting with customer 1, until the vehicle reaches the capacity limit C at customer j, and then returns to depot. It is then visited from depot, starting with the next customer $i+1$, until the capacity limit C is reached and returns to depot. all customers are visited in this way to generate the initial path.

```
def get_initial_routines(nodes):
    demands = 0
    initial_routines = []
    count = 1
    num_routine = 0
    flag = True
    while flag:
        initial_routines.append([])
        initial_routines[num_routine].append([nodes[0]])
        demands += nodes[count].demand
        while demands <= config().capacity:
            initial_routines[num_routine][0].append(nodes[count])
            count += 1
            if count == len(nodes):
                flag = False
                break
            demands += nodes[count].demand
        if count == len(nodes):
            initial_routines[num_routine][0].append(nodes[0])
            initial_routines[num_routine].append(demands)
        else:
            demands -= nodes[count].demand
            initial_routines[num_routine][0].append(nodes[0])
            initial_routines[num_routine].append(demands)
        num_routine += 1
        demands = 0
    return initial_routines
```

Figure 2.10 Code for Creating Initial Solution

2.3.2 Spectral Normalization

In order to explain spectral normalisation, it is first necessary to review linear algebra in order to broadly explain what a spectral norm is.

$$Av = \lambda v$$

where A is a square matrix, v is the eigenvector, and λ is its eigenvalue.

Eigenvectors are vectors that do not change direction when A is applied to the vector. Instead, they can be scaled by the scalar eigenvalue λ only. There can be more than one eigenvector-eigenvalue pair. The square root of the largest eigenvalue is the spectral norm of the matrix.

For non-square matrices, we would need to use mathematical algorithms (e.g. singular value decomposition (SVD)) to compute the eigenvalues, which can be computationally very expensive. Therefore, the use of power iteration can speed up the computation and make it feasible for neural network training.

Implementation :

1. Initialise the vector w with $N(0,1)$
2. In a for loop, compute the following (the number of iterations is a hyperparameter set to 5).
   - Calculate $v = \left(w^T\right) u$ using matrix transposition and matrix multiplication
   - Normalize v by L_2 normalization, i.e. $v = \frac{v}{\|v\|_2}$
   - Calculate $u = wv$
   - Normalize u by L_2 normalization, i.e. $u = \frac{u}{\|u\|_2}$
3. Compute spectral norm as $u^T wv$
4. Finally, divide the weights by the spectral norm and return, i.e. $\frac{w}{u^T wv}$

```python
class SpectralNorm(tf.keras.constraints.Constraint):
    def __init__(self, n_iter=5):
        self.n_iter = n_iter

    def call(self, input_weights):
        w = tf.reshape(input_weights, (-1, input_weights.shape[-1]))
        u = tf.random.normal((w.shape[0], 1))
        for _ in range(self.n_iter):
            v = tf.matmul(w, u, transpose_a=True)
            v /= tf.norm(v)
            u = tf.matmul(w, v)
            u /= tf.norm(u)
        spec_norm = tf.matmul(u, tf.matmul(w, v), transpose_a=True)
        return input_weights / spec_norm
```

Figure 2.11 Code for Spectral Normalization

2.3.3 Attention Layer

Self-attention can be considered as a feature extraction layer, where given input features $a_1, a_2, \cdots a_n$, fuse each input feature to obtain a new feature $b_1, b_2, \cdots b_n$ after the self-attention layer. As follows:

Let the input features be I and multiply them by the three matrices W^q, W^k, W^v to get Q(query), K(key) and V(value) respectively; Next, use the product of matrices Q and K to get the attention matrix A and normalize it to get $\hat{A}$; finally, multiply the normalized attention matrix $\hat{A}$ by V to get the final output feature O.

In the above self-attention, each input feature a^i multiplied by the matrices W^q, W^k and W^v yields a vector q^i, k^i and v^i respectively, called the single-headed self-attention mechanism. If these vectors q^i, k^i and v^i are split into n, the n-headed self-attentive mechanism is obtained. It is accepted that the multi-headed self-attentive mechanism is better than the single-headed one because the former can capture more dimensions of information.

Implementation :

1. Let the hyperparameter *num_heads* be the number of heads of the self-attentive mechanism so that the dimension *head_size* of each head is calculated.
2. Define W^q, W^k and W^v three matrices.
3. Multiplying the input features by W^q, W^k and W^v and the output tensor does not distinguish between multiple heads. The dimensionality changes as: input$(batch, n, input_size)$ to layers$(batch, n, hidden_size)$
4. Cut into num_heads heads and transform the dimensions. The dimensionality changes as: layers$(batch, n, hidden_size)$ to layers$(batch, num_heads, n, head_size)$
5. The matrices Q and K are multiplied together to obtain the attention matrix and divided by the open square of the dimension of the vector to prevent the attention fraction from increasing with dimensionality. The dimensionality changes as: layers$(batch, num_heads, n, head_size)$ to attention_scores$(batch, num_heads, n, n)$
6. Attention matrix is normalized and then multiplied by matrix V. The dimensionality changes as: attention_scores$(batch, num_heads, n, n)$ multiplied by value_layer$(batch, num_heads, n, head_size)$ to context_layer$(batch, num_heads, n, head_size)$

7. Putting together the results of the heads. The dimensionality changes as: context_layer$(batch, num_heads, n, head_size)$ to context_layer$(batch, n, hidden_size)$

```
class SelfAttention(keras.layers.Layer):
    def __init__(self):
        super(SelfAttention, self).__init__()

    def build(self):
        self.num_heads = 4
        self.hidden_size = 8
        self.head_size = self.hidden_size // self.num_heads

        self.query = keras.layers.Dense(self.hidden_size , activation=None,
                                        kernel_constraint=SpectralNorm(), name='Dense_query')
        self.key = keras.layers.Dense(self.hidden_size , activation=None,
                                      kernel_constraint=SpectralNorm(), name='Dense_key')
        self.value = keras.layers.Dense(self.hidden_size ,activation=None,
                                        kernel_constraint=SpectralNorm(), name='Dense_value')
        self.attn = keras.layers.Dense(self.hidden_size ,activation=None,
                                       kernel_constraint=SpectralNorm(), name='Dense_AttnG')
        self.sigma = self.add_weight(shape=[1], initializer='zeros', trainable=True, name='sigma')
    def transpose(self, x):
        new_shape = tuple(x.get_shape().as_list())[:-1] + (self.num_heads, self.head_size)
        x = tf.reshape(x, new_shape)
        return tf.transpose(x, [0, 2, 1, 3])

    def call(self, x):
        q = self.query(x)
        k = self.key(x)
        v = self.value(x)

        q = self.transpose(q)
        k = self.transpose(k)
        v = self.transpose(v)

        attn = tf.matmul(q, k, transpose_b=True)
        attn = attn/math.sqrt(self.head_size)
        attn = tf.nn.softmax(attn)

        context = tf.matmul(attn, g=v)
        context = tf.transpose(attn_g,[0,2,1,3])
        new_shape = tuple(attn_g.get_shape().as_list())[:-2]+(self.hidden_size,)
        context = tf.reshape(attn_g, new_shape)
        context = self.dense_attn_g(attn_g)
        output = attn_g*self.sigma
        return output
```

Figure 2.12 Code for Attention Layer

2.3.4 Operator

The algorithm uses First improve, which means that it is executed whenever it encounters an improvement that can be made. In order to improve the efficiency of the operation, the method of evaluation before execution is used.

The reason for not using best improve is that the iteration time for each round of best improve is too long, which would greatly reduce the computational efficiency of the whole algorithm.

```python
def two_opt(routine, fro, to):
    """
    Remove two edges and reconnect their endpoints
    :param routine: old routine
    :param fro: the beginning edge
    :param to: the end edge
    :return: new routine
    """
    new_routine = copy.deepcopy(routine)
    count = (to - 1 - fro) // 2
    for i in range(count):
        new_routine[fro + 1 + i] = routine[to - 1 - i]
        new_routine[to - 1 - i] = routine[fro + 1 + i]
    return new_routine
```

```python
def eva_two_opt(routine, fro, to):
    """
    Calculate the distance change after using 2-opt operator
    """
    depot = routine[0]
    old_distance = depot.distances[routine[fro].index][routine[fro + 1].index] + \
                   depot.distances[routine[to].index][routine[to - 1].index]
    new_distance = depot.distances[routine[fro].index][routine[to - 1].index] + \
                   depot.distances[routine[to].index][routine[fro + 1].index]
    return old_distance - new_distance


def apply_two_opt(routines):
    """
    Run the 2-opt operator if there is optimization
    """
    break_flag = False
    new_routines = copy.deepcopy(routines)
    total_diff = 0
    for k in range(len(routines)):
        for i in range(len(routines[k][0]) - 3):
            for j in range(i + 3, len(routines[k][0])):
                diff = eva_two_opt(routines[k][0], i, j)
                if diff - 0.000001 > 0:
                    new_routines[k][0] = two_opt(routines[k][0], i, j)
                    total_diff += diff
                    break_flag = True
                    break
            if break_flag:
                break
    return new_routines, total_diff
```

Figure 2.13 Code for Operator Sample

Chapter 3

Experiments Results

This section contains the results and analysis of the experiments. First, there is a detailed explanation of the CVRP setup and hyperparameters. Then a comparison of the results with previous neural network methodologies such as Nazari et al. [25], Kool et al. [17], Tian-Chen [6], and traditional state-of-the-art heuristics. After that is an examination of the Framework, which adopts the actor-critic algorithm with an ADAM optimiser whose learning rate is 10^{-5} to train the policy network. The nearest insertion method generates an initial feasible solution to the problem instance unless otherwise stated. Then RL is used to control the operator to iterate $T = 2000$ times to improve the solution. The final solution is the best one during the training for each instance. All reported metrics, such as total distance and execution time, are averaged over a random sample of 1000, unless otherwise indicated. Finally, our method was implemented in Python and Tensorflow2 and experimented with on a 2.3 GHz dual-core Intel Core i5 CPU on MBP.

Hyperparameters and Setup : According to the prior work [25, 17, 6], the training process use the same setup. Two subproblems with $N = 20$ and 50 customers are considered, respectively. Each problem has 1000 instance. The location of each node (x_i, y_i) is randomly sampled inside the unit square $[0,1] \times [0,1]$, and the cost of transit between them is the corresponding Euclidean distance $c_{i,j}$. Each customer's demand d_i uniformly sampled from the discrete set $\{1, \ldots, 9\}$. The vehicle's capacity C is 30 and 40, respectively, for $N = 20$ and 50.

3.1 Comparison

This approach compared with a variety of benchmarks, including. 1) Google OR-Tools[1], a well-established and widely-used meta-heuristic routing problem solver; 2) LKH3 [13], a mature heuristic solver with advanced performance on a variety of routing problems; 3) state-of-the-art DL-based methods by learning to construct, such as the Attention Model (AM) [17] and Nazari's model[25]; 4) DL-based methods by improve heuristics, i.e., NeuRewriter [6]. The fact is that Apple computers could not run machine learning on GPU, resulting in very low training efficiency and a prolonged training speed. Thus, all findings were obtained with a step size of $T = 2000$ due to CPU performance limits. Furthermore, because the execution time slows down significantly as the issue size grows, $N = 1000$ was chosen. This approach, however, achieves a better average distance than most decent algorithms now available, as demonstrated in Table 1. The table shows that the results of this method are optimal for $N = 20$, even surpassing the best traditional heuristic algorithm LKH3. The optimization result for $N = 50$ is not as good as for $N = 20$ but still exceeds most of the current state-of-the-art algorithms. It is possible that the results would have been much better if the running conditions had been better.

Table 3.1 Comparison with those methods mentioned above

	N = 20 Obj.	*N = 50* Obj.
OR-Tools	6.46	11.27
Zazari et al.[25]	6.40	11.15
AM sampling[17]	6.25	10.62
NeuRewriter[6]	6.16	10.51
My(T=2000)	6.12	10.57

It is worth noting that the figures of N=20 in the table were tested with the same 1000 instances, while the figures of N=50 are from their respective articles, as the CPU was too slow to test running 1000 instances of N=50, but the data we use is generated in the same way.

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

3.2 Progress Analysis

In the experimental study, as shown in Figure 3.1, the degree of optimization of a single operator was low, and the rate of optimization was not high. As shown in Figure 3.2, the improvement in the optimization of multiple operators is very significant, and the optimization rate is much faster. This situation proves that the choice to use RL to control multiple operators will be effective.

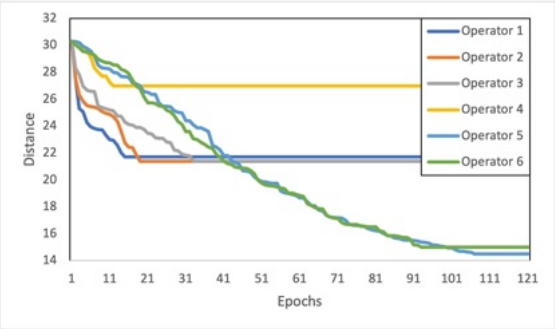

Figure 3.1 Single Operator

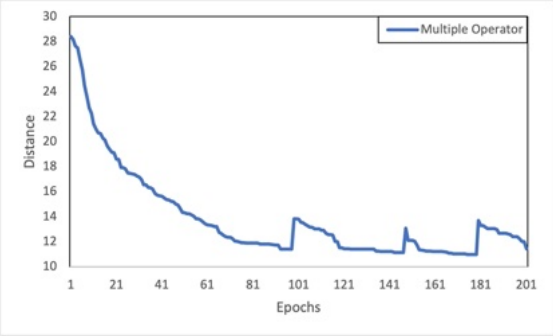

Figure 3.2 Multiple Operator

However, the metaheuristic is also a popular approach for using multiple operators, but it is challenging to do transfer learning. That is to say, the execution of the metaheuristic is a one-time event, and for each new set of problems, the whole optimisation algorithm needs to be run once for each instance. Nevertheless, the optimal solutions to these problems may be somehow strongly correlated in practice. For this case, once a neural-network-based reinforcement learning algorithm has solved one of the problems, it is possible to solve the others quickly. This generalisability of reinforcement learning is analysed in more detail in 3.4 below.

3.3 Analysis of Operator Usages

By observation, the RL model was able to distinguish between operators that were more beneficial to CVRP and those that were less useful. The number of different operators used by the various strategies was recorded in the experiments as the training phase was extended. When using the reward function RF1 as described in Section 2.2, the experimental results show that all these strategies converge to employ a fixed set of improved operators (see Table 3.2 for details).

The subset of operators (Table 3.3) formed by convergence using RF2 is slightly different from the subset obtained by RF1, but is generally consistent.

Table 3.2 Operators mostly used in RF1

Type	Name	Detail
Intra	Swap	Exchange 2 customers in the routine
	2-Opt	Remove two edges and reconnect their endpoints
Inter	Swap2	Exchange 2 or 3 consecutive customers between two routes

Table 3.3 Operators mostly used in RF2

Type	Name	Detail
Intra	Swap	Exchange 2 customers in the routine
Inter	Cross	Exchange two customers in two rou
Inter	Swap2	Exchange 2 or 3 consecutive customers between two routes

The pattern of operator usage, on the other hand, differs by approach. For instance, Figure 3.3 and Figure 3.4 demonstrate the distinct operator use patterns associated with strategy 1 and strategy 2.

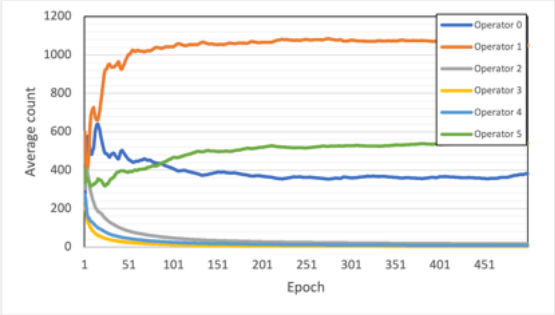

Figure 3.3 Operators usages in RF1

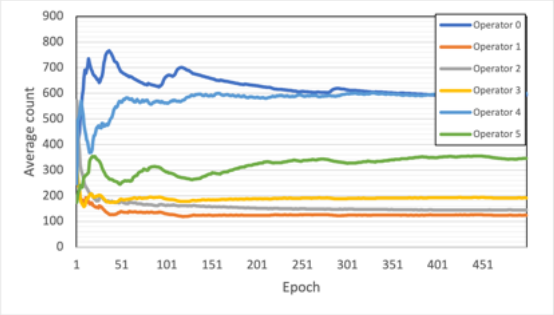

Figure 3.4 Operators usages in RF2

From the two graphs, we can see that the policy operator using RF1 converges faster. The actual results also prove that the policy with RF1 yields better results (for N=20, 6.12 vs 6.24).

3.4 Generalization Analysis

This part will demonstrate that the policies learnt by the model can be extended to approaches not observed in training.

First, the sensitivity of alternative beginning solutions for the same issue size was examined. As mentioned above, the initial training solution was obtained by using the closest insertion approach. There is an alternate way of producing initial solutions: random generation (extremely time intensive for CVRP generation). For the random starting solution, policy also performs at about the same level (10.57 vs 10.60). This illustrates that for the same issue size, the model-trained strategy has great generality for varying quality starting solutions.

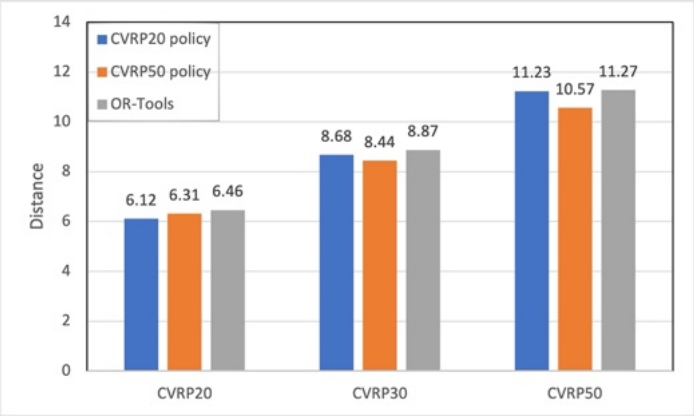

Figure 3.5 Generalization

Secondly, the generalizability of policy was investigated for various issue sizes. As demonstrated in Figure 14, policies learned on CVRP20 and CVRP50 generalised well to various issue sizes, with all training outcomes outperforming OR-Tools. Of the above RL-based CVRP models, only the NeuRewritter model can generalise across different problem sizes, and the generalisation results by this model are all the better than its. This suggests that this model is more generalisable.

3.5 Visualization

This section will give a simple demonstration of the performance of the learning policy for the improvement of the route. In Figure 3.5, Figures (a)-(h) show the training process for an instance of CVRP50. Figure (a) shows the initial solution generated using the nearest insertion method, and figure (h) shows the final trained optimal solution. Figures (b)-(g) visualise the six consecutive states and the five actions used in the training process. The lines and points marked with red fluorescence are the paths that changed after the actions were used. Figure (d) shows an intra-route operator - Relocate, and the other four show

inter routes operators. Figure (c) represents the Cyclic operator, choosing one customer from each of the three lines to exchange in a cycle; Figure (e) and figure (f) represent the Cross operator, choosing one customer from each of the two routes to exchange; Figure (g) represent the Swap2 operator, choosing 2 or 3 consecutive customers from each of the two routes to exchange. Finally, figure (h) shows that the paths have reached convergence and are a reasonable solution.

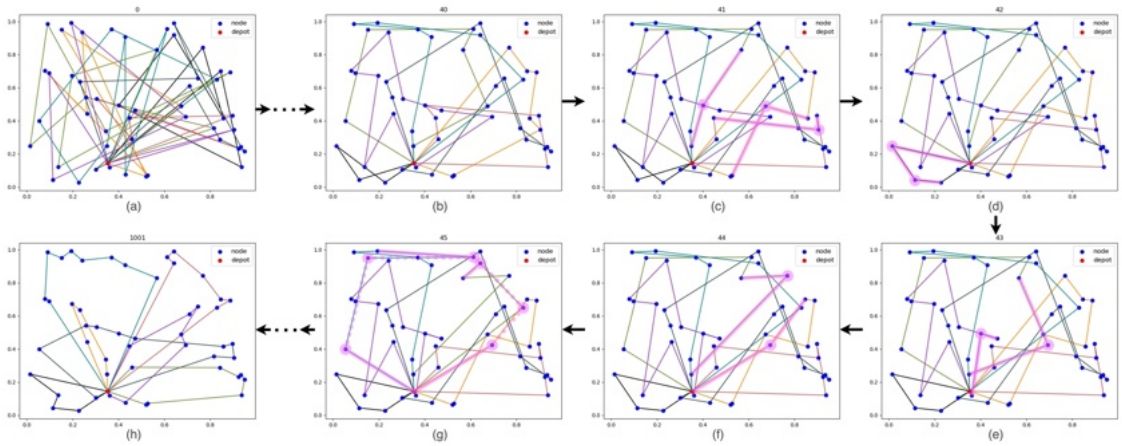

Figure 3.6 Visualization

3.6 Test on Real-World Datasetn

Although trained on random data, this learnt policy performed well on instances of the publicly available CVRPlib[2] benchmark, which comprises real-world issue situations. Note that the distribution of these examples in node placement patterns, client demand, and vehicle capacity may change significantly from the distribution utilized in our training.

Eleven instances on CVRPlib, ranging in size from 101 to 150, were evaluated, each produced using a different warehouse location (central, eccentric, random), customer location (random, clustering), and demand distribution (small or large differences).

Table 3.4demonstrate that our technique outperforms the AM approach by a wide margin. Our technique provides a twofold reduction in the average optimality gap compared to AM and outperforms AM in most (5 of 11) of these CVRPlib situations. This strategy's quality

[2]http://vrp.galgos.inf.puc-rio.br/index.php/en/

Table 3.4 Generalization on CVRPlib

Instance	Opt.	OR-Tools	AM	My
X-n101-k25	27,591	29,405	37,702	**30,673**
X-n106-k14	26,362	27,343	28,473	**28,137**
X-n110-k13	14,971	16,149	**15,443**	16,350
X-n115-k10	12,747	13,320	**13,745**	14,445
X-n120-k6	13,332	14,242	**13,937**	15,486
X-n125-k30	55,539	58,665	75,067	**60,423**
X-n129-k18	28,940	31,361	**30,176**	32,126
X-n134-k13	10,916	13,275	13,619	**12,669**
X-n139-k10	13,590	15,223	**14,215**	16,792
X-n143-k7	15,700	17,470	**17,397**	18,872
X-n148-k46	43,448	46,836	79,514	**56,589**
Avg. Gap	0	8.74%	20.25%	15.13%

Bold means the best among two learning based methods.

does not decrease rapidly on CVRPlib, since the average gap on instances with 101-150 nodes is 15.13 per cent.

Chapter 4

Conclusion and Limitation

4.1 Conclusion

This paper proposes a deep reinforcement learning framework to learn improvement heuristics for solving CVRP automatically. It starts with an initial solution generated by a close-to-insert method and learns how to choose improvement operators to improve that solution based on a self-attentive RL structure. From experimental results, this method outperforms most state-of-the-art depth models for CVRP while closing the gap with highly optimized solvers and even outperforming them for small problem sizes. Furthermore, the model learns highly generalizable strategies for different initial solutions and problem sizes and gives reasonable solutions to real-world datasets.

4.2 Limitation

However, the model in this paper has several obvious limitations. Firstly, it is impossible to determine the actual run rate, as the model is prolonged during training due to the lack of GPU resources and insufficient CPU arithmetic power. Secondly, the slow training speed resulted in the inability to train large problem sizes such as CVRP100 and CVRP200. Since large-scale problems would be closer to real-world problems, testing real-world datasets shows that this model is still far from OR-Tools on real-world problems.

References

[1] Bansal, N., Blum, A., Chawla, S., and Meyerson, A. (2004). Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 166–174.

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

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

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

[5] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724.

[6] Chen, X. and Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32.

[7] Das, A. and Mathieu, C. (2010). A quasi-polynomial time approximation scheme for euclidean capacitated vehicle routing. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 390–403. SIAM.

[8] Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., and Rousseau, L.-M. (2018). Learning heuristics for the tsp by policy gradient. In *International conference on the integration of constraint programming, artificial intelligence, and operations research*, pages 170–181. Springer.

[9] Dorling, K., Heinrichs, J., Messier, G. G., and Magierowski, S. (2016). Vehicle routing problems for drone delivery. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(1):70–85.

[10] Ghorbani, E., Alinaghian, M., Gharehpetian, G., Mohammadi, S., Perboli, G., et al. (2020). A survey on environmentally friendly vehicle routing problem and a proposal of its classification. *Sustainability*, 12(21):9079.

[11] Hassin, R. and Keinan, A. (2008). Greedy heuristics with regret, with application to the cheapest insertion algorithm for the tsp. *Operations Research Letters*, 36(2):243–246.

References

[12] Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130.

[13] Helsgaun, K. (2017). An extension of the lin-kernighan-helsgaun tsp solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University*, pages 24–50.

[14] Kaempfer, Y. and Wolf, L. (2018). Learning the multiple traveling salesmen problem with permutation invariant pooling networks. *arXiv preprint arXiv:1803.09621*.

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

[16] Khan, R. S., Yang, J.-B., and Handl, J. (2015). Interactive multi-objective vehicle routing via ga-based dynamic programming. In *2015 International Conference on Transportation Information and Safety (ICTIS)*, pages 318–322. IEEE.

[17] Kool, W., van Hoof, H., and Welling, M. (2018). Attention, learn to solve routing problems! In *International Conference on Learning Representations*.

[18] Laporte, G. and Nobert, Y. (1983). A branch and bound algorithm for the capacitated vehicle routing problem. *Operations-Research-Spektrum*, 5(2):77–85.

[19] Lin, K., Musa, S. N., and Yap, H. J. (2022). Vehicle routing optimization for pandemic containment: A systematic review on applications and solution approaches. *Sustainability*, 14(4):2053.

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

[21] Lodi, A. and Zarpellon, G. (2017). On learning and branching: a survey. *Top*, 25(2):207–236.

[22] Lombardi, M. and Milano, M. (2018). Boosting combinatorial problem modeling with machine learning. *arXiv preprint arXiv:1807.05517*.

[23] Lysgaard, J., Letchford, A. N., and Eglese, R. W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445.

[24] Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56.

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

[26] Pinto, T., Alves, C., and Valério de Carvalho, J. (2017). Models and advanced optimization algorithms for the integrated management of logistics operations. In *Congress of APDIO, the Portuguese Operational Research Society*, pages 313–324. Springer.

[27] Smith, K. A. (1999). Neural networks for combinatorial optimization: a review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–34.

[28] Toth, P. and Vigo, D. (2002). An overview of vehicle routing problems. *The vehicle routing problem*, pages 1–26.

[29] Toth, P. and Vigo, D. (2014). *Vehicle routing: problems, methods, and applications*. SIAM.

[30] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[31] Vidal, T., Laporte, G., and Matl, P. (2020). A concise guide to existing and emerging vehicle routing problem variants. *European Journal of Operational Research*, 286(2):401–416.

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

[33] Zhang, B., Xiong, D., Xie, J., and Su, J. (2020). Neural machine translation with gru-gated attention model. *IEEE transactions on neural networks and learning systems*, 31(11):4688–4698.

Appendix A

All code for the Final Year Project

All of the Python code has already been uploaded to the XJLU Box by me. All of the source code for this final year project can be downloaded from XJLU Box.

URL:https://box.xjtlu.edu.cn/smart-link/cce43b70-36c3-44d0-b7f1-e25fd0ef19a6/