Passenger Assignment For Ridesharing Through Supervised Learning

By

Sania Hamid

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

May 18, 2020

Abstract

Ridesharing systems are transforming urban transportation in an economically feasible, environmentally friendly and socially beneficial way. The Ridesharing problem is an intriguing problem in transportation research and has been studied for several years. The objective is to find efficient assignments for transportation of items through a complex network while respecting the capacity constraints of the available vehicles in the fleet. In recent times, more work has been focused on using data driven and intelligent approaches to solve problems arising in transportation research. In this thesis we aim to solve the ridesharing problem using neural network architectures. Our results indicate that the neural network architecture models generate solutions with reasonable accuracy and consume less computational time.

Acknowledgements

I would like to thank my thesis supervisor Dr. Doron Nussbaum for his support and helpful guidance. His insight and feedback on my research acted as a source of encouragement for me through this journey. Without his guidance, this thesis would not have been possible.

I would also like to thank my friends and family for their support and understanding throughout the process. Specially Atif Hamid and Sabiha Hamid who gave me encouragement and moral support throughout the entire process.

Contents

Abstract **ii**

Acknowledgements **iii**

1 Introduction **1**

1.1 Motivation 1

1.2 Problem definition 3

1.2.1 Capacitated Vehicle Routing Problem 3

1.2.2 Heterogeneous Fleet Vehicle Routing Problem 4

1.3 Contribution summary 5

1.4 Thesis organization 6

2 Background **7**

2.1 Previous Work 7

2.1.1 Vehicle Routing Problem 8

2.1.2 Combinatorial Optimization 8

2.1.3 Variations in Vehicle Routing Problem 10

2.1.3.1 Capacitated Vehicle Routing Problem 10

2.1.3.2 Vehicle Routing Problem with Time Windows 11

2.1.4 Heterogeneous Fleet Vehicle Routing Problem 12

2.1.5 Exact Algorithms 13

2.1.5.1 Linear Programming 13

2.1.5.2 Set Partitioning 14

2.1.5.3 Need for Heuristic and Meta-heuristic Algorithms . . 15

2.1.6 Heuristic Algorithms 15
2.1.6.1 Saving Heuristic 16
2.1.6.2 Sweep Heuristic 16
2.1.6.3 Cluster-First, Route-Second Heuristic 16
2.1.7 Meta-heuristic Algorithms 17
2.1.7.1 Tabu Search 18
2.1.7.2 Simulated Annealing 19
2.1.7.3 Ant Colony Optimization 19
2.1.7.4 Genetic Algorithm 20
2.1.8 Data-Driven Systems 21
2.1.9 Intelligent Systems 23
2.1.9.1 Reinforcement Learning 23
2.1.9.2 Supervised Learning 24
2.1.10 Background for Pointer Network 26
2.1.10.1 Neural Network 26
2.1.10.2 Multi Layer Neural Network 29
2.1.10.3 Recurrent Neural Network 31
2.1.10.4 Long Short-Term Memory Architecture 34
2.1.10.5 Sequence-to-Sequence Models 37
2.1.10.6 Attention-Based Models 38
2.1.10.7 Pointer Network 41

3 Problem Definition and Objective 44
3.1 Problem Definition 45
3.1.1 Capacitated Vehicle Routing Problem 45
3.1.2 Heterogeneous Fleet Vehicle Routing Problem 47
3.2 New York Yellow Taxi Dataset 49
3.3 Solving CVRP 52
3.3.1 Greedy Solution 53
3.3.2 Linear Programming Solutions 54
3.3.2.1 Calculating Distance 56

3.3.3 Solving HFVRP 59
3.3.4 Linear Programming Solutions 59
3.4 Training the Pointer Network 64
3.5 Bidirectional LSTM 65

4 Experimental Results and Analysis 67
4.1 Testing Methodology 67
4.1.1 Architecture and Hyperparameters of Pointer Network 69
4.1.2 Measurement of Results 70
4.2 Experiment Setup 70
4.3 Empirical Results 71
4.3.1 Average Total Distance 71
4.3.2 Optimality Gap 76
4.3.3 Minimum and Maximum Distance 79
4.3.4 Average Total Time 80
4.4 Issues with Pointer Network 86
4.4.1 Mismatch of groups 87
4.4.2 Groups with repeated vertices 90
4.4.3 Repeated vertices in an entire sequence 92
4.4.4 Excluded vertices 93
4.5 Serviceable Matchings 95
4.6 Root Mean Square Percentage Error 97

5 Conclusion 103
5.1 Future Work 104

Bibliography 106

List of Tables

4.1 Average Total Distance for comparison algorithms (CVRP) 71
4.2 Average Total Distance for comparison algorithms (LPAC, HFVRP) . 72
4.3 Average Total Distance for comparison algorithms (LPRC, HFVRP) . 72
4.4 Optimality Gap for 15 customers (CVRP) 76
4.5 Optimality Gap for 15 customers (LPAC, HFVRP) 77
4.6 Optimality Gap on ridesharing for 15 customers (LPRC, HFVRP) . . 77
4.7 Optimality Gap of ridesharing for 30 customers (CVRP) 77
4.8 Optimality Gap of ridesharing for 30 customers (LPAC, HFVRP) . . 77
4.9 Optimality Gap of ridesharing for 30 customers (LPRC, HFVRP) . . 78
4.10 Minimum and Maximum distance generated for CVRP 80
4.11 Minimum and Maximum distance generated by LPAC for HFVRP . . 80
4.12 Minimum and Maximum distance generated by LPRC for HFVRP . . 80
4.13 Average Total Time for solving CVRP 81
4.14 Average Total Time for solving HFVRP using LPAC solutions 81
4.15 Average Total Time for HFVRP using LPRC solutions 82
4.16 Time Comparison for 128 sets of 15 customers (CVRP) 84
4.17 Time Comparison for 128 sets of 15 customers (LPAC, HFVRP) . . . 84
4.18 Time Comparison for 128 sets of 15 customers (LPRC, HFVRP) . . . 84
4.19 Time Comparison for 128 sets of 30 customers (CVRP) 85
4.20 Time Comparison for 128 sets of 30 customers (LPAC, HFVRP) . . . 85
4.21 Time Comparison for 128 sets of 30 customers (LPRC, HFVRP). . . 85
4.22 Average mismatch and difference in distance for 15 customers. 89
4.23 Average mismatch and difference in distance for 30 customers. 89

4.24 Average number of groups with repeated vertices (15 customers) . . . 91
4.25 Average number of groups with repeated vertices (30 customers) . . . 91
4.26 Average number of repeated vertices (15 customer) 92
4.27 Average number of repeated vertices (30 customer) 93
4.28 Average number of excluded vertices (15 customer) 94
4.29 Average number of excluded vertices (30 customer) 94
4.30 Average number of serviceable groups (15 customers) 96
4.31 Average number of serviceable groups (30 customers) 97
4.32 Root Mean Square Percentage Error (CRP) 98
4.33 Root Mean Square Percentage Error (HFVRP, LPAC) 98
4.34 Root Mean Square Percentage Error (HFVRP, LPRC) 98
4.35 Root Mean Square Percentage Error for Greedy solution 101
4.36 Root Mean Square Percentage Error for LPD 101
4.37 Root Mean Square Percentage Error for LPAC25 101
4.38 Root Mean Square Percentage Error for LPAC50 102
4.39 Root Mean Square Percentage Error for LPRC25 102
4.40 Root Mean Square Percentage Error for LPRC50 102

List of Figures

2.1 VRP with four vehicles, starting and ending at depot after visiting each passenger. 9
2.2 Representation of a neuron 26
2.3 Sigmoid activation function 28
2.4 Tanh activation function 29
2.5 Multi-layer Neural Network 30
2.6 Backpropagation 32
2.7 A Recurrent Neuron 33
2.8 Unfolding of a Recurrent Neural Network. 34
2.9 LSTM cell architecture. 35
2.10 Basic Sequence-to-Sequence, Encoder-Decoder model. 37
2.11 Attention mechanism is sequence-to-sequence model 39
2.12 Bahadanau Attention Mechanism 40
2.13 Architecture of Pointer Network model. 41
2.14 Attention in a pointer network model 43

3.1 Pickup and dropoff in New York Yellow Taxi Dataset. 51
3.2 Example of trips originating in a cell. 52
3.3 Possible order of travel for 3 passengers. 57
3.4 Possible order of travel for 3 passengers including the depot. 58
3.5 Structure of a Bidirectional LSTM network 66

4.1 Ridesharing matches for 15 customers (LPRC75). 68

4.2 Average Total Distance for 15 customer requests. Orange indicates CVRP solution, blue indicates HFVRP solution using LPAC and green indicates HFVRP solution using LPRC 72
4.3 Average Total Distance for 30 customer requests. Orange indicates CVRP solution, blue indicates HFVRP solution using LPAC and green indicates HFVRP solution using LPRC 73
4.4 Ridesharing matches generated for 15 customers (LPRC25) 74
4.5 Ridesharing matches generated for 15 customers (LPRC75) 75
4.6 Ridesharing matches for 15 customers (Greedy) 75
4.7 Optimality Gap of ridesharing matches for 15 customers.(a) for CVRP (b) For HFVRP using LPAC (c) for HFVRP using LPRC 78
4.8 Optimality Gap of ridesharing matches for 30 customers.(a) for CVRP (b) For HFVRP using LPAC (c) for HFVRP using LPRC 79
4.9 Average Total Time per 15 customers. Orange indicates CVRP solution, blue indicates HFVRP solution using LPAC and green indicates HFVRP solution using LPRC 82
4.10 Average Total Time per 30 customers. Orange indicates CVRP solution, blue indicates HFVRP solution using LPAC and green indicates HFVRP solution using LPRC 83
4.11 Target-matching for 15 customers (LPAC25). 87
4.12 Predicted-matching for 15 customers (Ptr-Net-LSTM). 88
4.13 Predicted-matching for 15 customers with repeated vertices. 95
4.14 Comparison of RMSPE (15 customers). (a) for CVRP (b) For HFVRP using LPAC (c) for HFVRP using LPRC 99
4.15 Comparison of RMSPE (30 customers). (a) for CVRP (b) For HFVRP using LPAC (c) for HFVRP using LPRC 100

Chapter 1

Introduction

With the rapidly expanding metropolitan cities, ridesharing services have emerged as a convenient and cost-efficient option for commuters to mobilize themselves in cities. Ridesharing is a quick and convenient way to minimize traffic congestion, to reduce air pollution and to save on gas. In recent years advancements in data-driven technologies enabled the development of more advanced algorithms to solve the problem of ridesharing.

1.1 Motivation

Tech companies and startup are currently leading research towards developing fully autonomous vehicles (AVs). Google's driver-less car project maintains a fleet of vehicles that have already driven 3,219,00 km as of December 2016 [72]. Apple started Project Titan for building autonomous vehicles in 2014, and have been testing their

software platform on fleets of vehicles since 2017 [48]. Tesla announced Tesla Autopilot in 2014 with the purpose of developing fully autonomous vehicles [66]. Autonomous vehicles have the capability of being used as a shared mode of transit. These shared autonomous vehicles come with many added benefits, like, reduced emission of green-house gasses reduced traffic congestion and traffic accidents, economic feasibility for passengers, increased road capacity and speed etc. Traditional ridesharing algorithms can be evolved to to avail an efficient use of AVs as a shared mode of on-demand transit.

Rideshairng problem is well studied in the area of transportation research. It involves assigning passengers to vehicles and then routing the vehicles to visit passenger locations. It is a formulation of the Vehicle Routing Problem (VRP) which was first introduced by Dantzig in 1959 [20]. It has been classified as an NP-Hard combinatorial optimization problem by Lenstar in 1981 [44]. Since then, a lot of research has been carried out and a lot of progress and improvements have been made on solving VRP. Several techniques on exact methods, heuristic and meta-heuristic methods with strong formulations have been developed in solving VRP [2, 8, 12, 15, 17, 23, 24, 29, 41, 57] etc. More recent work has been focused on data-driven and intelligent algorithms to solve ridesharing problem and its variants [4, 27, 36, 49, 54, 58]. Vinyals et al. [69] proposed a neural networks architecture called Pointer Network (Ptr-Net) to solve combinatorial optimization problems such as finding planar convex hull, computing Delaunay Triangulation and solving Travelling Salesman Problem. In this thesis we test the applicability of Pointer Network in generating solutions to ridesharing as an assignment problem.

1.2 Problem definition

The Ridesharing problem is a formulation of VRP which is an NP-Hard problem. The problem is divided into two parts first is assigning the customers to vehicles and the second is to routing the vehicle to through customer locations. In this thesis we are concerned only with the assignment of customers to the vehicles hence, throughout the thesis the terms VRP, Capacitated Vehicles Routing Problem (CVRP) and Heterogenous Fleet Vehicle Routing Problem (HFVRP) deals with only assigning the passengers to the vehicles. VRP tries to generate a sequence of deliveries for each vehicle in a fleet of vehicles based on a single depot so that all customers are serviced and the total distance travelled by the fleet is minimized. Each vehicle has a fixed capacity and must leave from and return to the depot. Each customer has a known demand and is serviced by exactly one visit of a single vehicle. CVRP is a subclass of VRP with an additional constraint that every vehicle must have uniform capacity. HFVRP has an added constraint that the vehicles in the fleet have different capacity. The purpose of this thesis is to test the applicability of Pointer Network on generating solutions to the assignment problem of ridesharing where customers are grouped together in order to share a vehicle.

1.2.1 Capacitated Vehicle Routing Problem

The classical formulation on CVRP defines it as a graph theoretical problem [56], where $G = (V, E)$ is a complete undirected graph, where $V = \{v_0, v_1, ..., v_n\}$ is the vertex set of $n+1$ modes where n is the number of customers. $E = \{e_{01}, e_{02}, ..., e_{n-1n}\}$ where $e_{ij} = \{v_i, v_j\}$ and $i \neq j$, for $0 \leq i, j \leq n$ is the edge set. Vertex v_0 denotes

the depot where all vehicles start and end their rides. All the other vertices in $V_c = V \setminus \{v_0\}$ correspond to a single customer's drop-off location with each having a non-negative capacity demand $d_{v_i} \geq 0$. These customers are to be serviced by an unlimited number of k vehicles having identical capacity $C > 0$ such that $d_{v_i} \leq C$. The cost of traveling on edge e_{ij} is represented by $w_{e_{ij}}$ and it is assumed to be symmetric, i.e. $w_{e_{ij}} = w_{e_{ji}}$. Each customer location should be serviced by a single vehicle and the vehicles must start and end at the dept.

The problem is to how to partition V_c into k sets $\{r_1, r_2, ..., r_k\}$, each satisfying, $\sum_{j \in r_i} d_{v_j} \leq C$. It is to find a union of k cycles (Equation 1.1) whose intersection is the depot v_0 (Equation 1.2). Each cycle corresponds to a customer assignment serviced by one of the k vehicle.

$$r_i \bigcup_{i=1}^{k} r_j = V_c \quad , \forall 1 \leq i, j \leq n, i \neq j \tag{1.1}$$

$$r_i \bigcap_{i=1}^{k} r_j = v_0 \quad , \forall 1 \leq i, j \leq n, i \neq j \tag{1.2}$$

1.2.2 Heterogeneous Fleet Vehicle Routing Problem

The Heterogeneous Fleet Vehicle Routing Problem (HFVRP) tries to find the minimal cost assignment for a fleet of vehicles with varying capacity. HFVRP is a generalization of CVRP, where the assumption that all vehicles have same capacity is relaxed. Similar to the definition of CVRP described in section 3.1.1, Let, $G = (V, E)$ be an undirected graph, with $V = \{v_0, v_1, ..., v_n\}$ is the vertex set one $n + 1$ nodes, where n is the number of customer locations. $E = \{e_{01}, e_{02}, ..., e_{n-1n}\}$ where $e_{ij} = \{v_i, v_j\}$

and $i \neq j$, for $0 \leq i, j \leq n$ is the edge set. Vertex v_0 denotes the depot where all vehicles start and end their rides. The vector set $V_c = V \setminus \{v_0\}$ corresponds to customer drop-off locations with non-negative capacity demand $d_{v_i} \geq 0$. It is assumed that there are unlimited number of vehicles in the fleet with have capacity C_s where $s = \{1, 2, ..., T\}$ represents capacity of the vehicle such that $(C_1 < C_2 < ... < C_T)$. The cost of traveling an edge e_{ij} is represented by $w_{e_{ij}}$ and it is assumed to be symmetric, i.e. $w_{e_{ij}} = w_{e_{ji}}$. Each customer must be serviced by a single vehicle and no vehicle can service a set of customers when demand exceeds the capacity of the vehicle i.e. $d_{v_i} \leq C_s$. The vehicles must start from the depot and end at the of the depot.

The problem is to how to partition V_c into s sets $\{r_1, r_2, ..., r_s\}$, each satisfying, $\sum_{j \in r_i} d_{v_j} \leq C$. It is to find a union of s cycles (Equation 1.3) whose intersection is the depot v_0 (Equation 1.4). Each cycle corresponds to a customer assignment serviced by one of the s vehicle.

$$r_i \bigcup_{i=1}^{s} r_j = V_c \quad , \forall 1 \leq i, j \leq n, i \neq j \tag{1.3}$$

$$r_i \bigcap_{i=1}^{s} r_j = v_0 \quad , \forall 1 \leq i, j \leq n, i \neq j \tag{1.4}$$

1.3 Contribution summary

In this thesis, we generate solution for VRP and its variants (CVRP and HFVRP) using neural networks. We first generate sample solutions using a greedy heuristic and linear programming formulations. We collectively call these algorithms as comparison

algorithms because they are used to compare the solutions generated by a neural network model called Pointer Network (Ptr-Net) . We also use the solutions generated by the comparison algorithm to train Ptr-Net. We have proposed an enhanced model of the Ptr-Net described in [69] by using Bidirectional Long Short Term Memory cells (BLSTM). We then compare the performance of the Ptr-Net models. The aim of this study is to gauge what purely data-driven algorithms will be able to achieve with respect to exact solutions. We experimentally find that the Ptr-Net models trained with the results of comparison algorithms applied to the problem are able to predict best ridesharing results with a reasonable degree of accuracy.

1.4 Thesis organization

This thesis is organized as follows: In Chapter 2 we present previous research that are relevant to the research presented in this thesis. It also introduces basic concepts and notations that are used in the text. In Chapter 3 we present the research problem, objective and the solution to the problem. In Chapter 4 we discuss and analyze the experimental results. In Chapter 5 we give a summary of the research work presented here, and point out to open problems and suggest future research.

Chapter 2

Background

In this chapter, we provide the major concepts needed to understand the core of the thesis. We first present the Vehicle Routing Problem (VRP) and provide an overview of work on VRP and its variations. We further discuss various methods adopted by authors to solve the problem. We then discuss the advanced approach of data-driven and intelligent algorithms to solve VRP. We also discuss the basics of supervised learning including Neural Networks and Recurrent Neural Networks. We conclude this chapter by describing Pointer Networks which is a data driven algorithm used for solving optimization problems.

2.1 Previous Work

Taxi ridesharing is an on-demand transport service that caters to customer's demands. Taxi ridesharing aims at matching passengers and vehicles subject to some

constraints. With the advancement of technology, big tech companies like Uber and Lyft are committed towards providing mobility-as-a-service to general public [34]. A large amount of research has been done and many models have been proposed towards the development of effective ridesharing algorithms. These services have a beneficial impact on urban transportation as they provide timely and convenient transportation anywhere and anytime [49].

2.1.1 Vehicle Routing Problem

The ridesharing problem is an optimization problem and a formulation of the Vehicle Routing Problem (VRP) first introduced by Dantzig in 1959 [20]. It is an optimization problem. Optimization is a technique of finding the best solution when more than one solution are available. More specifically, VRP is a combinatorial optimization problem.

2.1.2 Combinatorial Optimization

Combinatorial optimization is the mathematical study of finding an optimal arrangement, grouping, ordering or selection of discrete objects usually finite in number [33]. Formally, a combinatorial optimization problem $\mathbf{A}$ is a quadruple (I, f, m, g), where

- I is a set of instances
- given an instance $x \in I, f(x)$ is the finite set of feasible solutions
- given an instance x and a feasible solution y of x, $m(x, y)$ denotes the measure of y, which is usually a positive real number.

- g is the objective function, and is either **min** or **max**

The goal is then to find for some instance x an optimal solution, that is a feasible solution y with

$$m(x, y) = g\{m(x', y')|y' \in f(x)\} \tag{2.1}$$

Examples of combinatorial optimization problems are: Traveling Salesman Problem, Bin-Packing Problem, Job-Shop scheduling, Vehicle Routing Problem etc.

VRP is a generalization of Travelling Salesman Problem (TSP). The goal of VRP is to find an optimal route for multiple vehicles visiting a set of locations. When there is only one vehicle the problem reduces to TSP. The classical VRP aims to

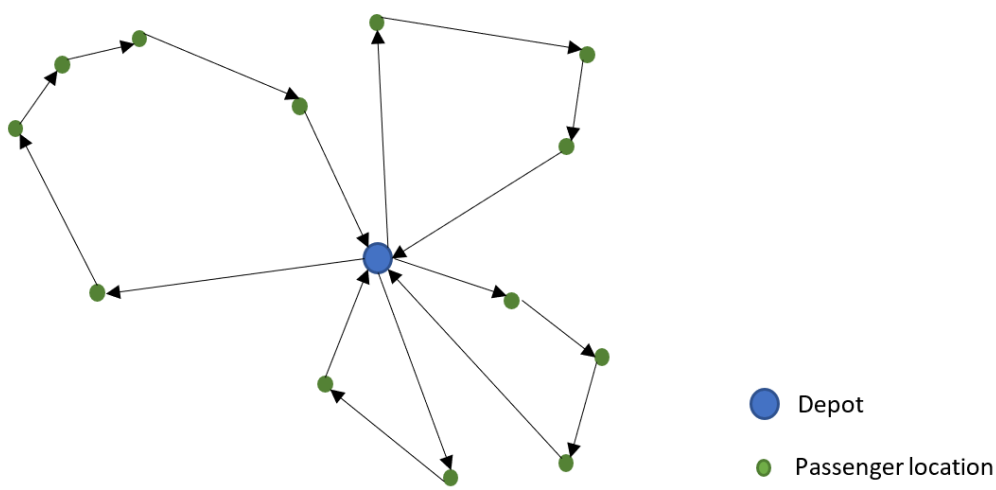

Figure 2.1: VRP with four vehicles, starting and ending at depot after visiting each passenger.

find least-cost routes from a central depot to a set of geographically dispersed points

(passenger pickup/drop-off locations) with various demands using a fleet of vehicles. Each passenger must be served by exactly one vehicle and each vehicle must start and end its route at the depot (Figure 2.1). VRP has been classified as an NP-hard combinatorial optimization problem by Lenstra et al. [44].

2.1.3 Variations in Vehicle Routing Problem

Several variations of VRP exist in literature, beyond the classical formulation. The cases of homogeneous and heterogeneous fleet of vehicles, servicing customer demand while meeting customer constraints gives rise to the Capacitated Vehicle Routing Problem (CVRP) [46, 60, 2] and the Heterogeneous Fleet Vehicle Routing Problem (HFVRP) [14, 31] respectively. Adding time constraints to the problem makes the problem real time compliant. A Vehicle Routing Problem with Time Windows (VRPTW) aims to serve customer requests with hard/soft time intervals [53, 62]. A Dynamic Vehicle Routing Problem (DVRP) is strongly related to VRP when customer requests are known in real-time and must be included in the optimized solution [52].

2.1.3.1 Capacitated Vehicle Routing Problem

A Capacitated Vehicle Routing Problem is defined as follows: given a homogeneous fleet of vehicles with limited capacity, the objective is to assign passengers to taxis while minimizing the total travel distance and meeting capacity constraints and every vehicle must have uniform capacity.

Shin et al. [60] proposed a heuristic algorithm that assigns vehicles to customers based on clusters, where the size of each cluster is the same as the capacity of each vehicle. The time complexity of the algorithm is $\mathcal{O}(n^2)$ where n is the number of passengers. The authors tried to solve instances of CVRP with 15 to 100 passengers. Akhand et al. [2] proposed a sweep based algorithm for grouping passengers in a ride. The algorithm calculates the polar angle of all destination locations of customers and forms clusters based on those angles. They compared the results of a few swarm intelligent algorithms like ant colony optimization [22], particle swarm optimization [70], producer-scrounge method [3] and velocity tentative particle swarm optimization [1] for finding the best solution of optimally routing the vehicles. The authors used a set of benchmark datasets with 200 to 500 customers.

2.1.3.2 Vehicle Routing Problem with Time Windows

In addition to CVRP the objective of Vehicle Routing Problem with Time Windows (VRPTW) is to serve customers within predefined time windows. Here, each customer trip request comes with imposed delivery deadline. The total cost includes distance of travel between customer pickup location and customer drop-off location, time required to travel that distance and the time incurred by the vehicle while waiting for the customer during loading and unloading.

Solomon et al. [62] proposed several heuristics that incorporated distance and time. They proposed nearest-neighbor heuristic, insertion heuristic and a time-oriented sweep heuristic. The primary objective was to minimize the fleet size and the secondary objective was to minimize the travel distance. To simplify the problem they

calculated the distance between customer pickup and drop-off locations as the Euclidean distance.

Ombuki et al. [53] proposed a solution using genetic algorithm (GA) to solve VRPTW. They viewed VRPTW as a multi-objective problem. The two objectives were to minimize vehicle fleet and the total travel distance. They applied a problem specific ranking scheme and a crossover operation known as rout-exchange crossover. Their dataset contained 300 customers and results obtained were at par with the best published results.

2.1.4 Heterogeneous Fleet Vehicle Routing Problem

Heterogeneous Fleet Vehicle Routing Problem differs from VRP in the sense that it deals with a fleet of variable capacity. Depending on the problem, HFVRP can be extended to have variable time constraints. The objective again is to minimize the total cost of routing the vehicles. Golden et al. [31] first introduced HFVRP and they assigned variable cost on each type of vehicle while assuming an unlimited number of vehicles. The objective of the problem was to find the best fleet size which will reduce the cost of travel. They defined several heuristic techniques to solve the problem and compared them against each other. They solved the problem instance with 12 to 100 customers.

Choi et al. [14] introduced a column generation approach to solve HFVRP. The approach finds a set of variables that have the potential to improve the objective function to solve the tight integer programming model of the problem. The cost of routing was taken as the sum of a fixed cost and a variable cost which is incurred in proportion to the distance of travel. The objective was to minimize the cost of

routing. It took them 0.22 secs to solve for 20 customer requests and more than 1000 seconds to solve 100 customer requests at a time.

Since, ridesharing is an instance of VRP, the algorithms used to solve VRP can be applied in the same manner to solve the ridesharing problem. Many techniques like exact algorithms, heuristic algorithms and meta-heuristic algorithms are used to solve VRP and its variations.

2.1.5 Exact Algorithms

The Ridesharing problem is an NP-hard problem and cannot be solved using exact algorithms for a large input size. Exact algorithms only provide optimal solutions to small and medium instances of the problem.

2.1.5.1 Linear Programming

Linear program (LP) is an optimization problem over real numbers. It optimizes a linear cost function of a set of variables $\mathbf{x} = (x_1, x_2, ..., x_n)$ subject to a set of linear equality or inequality constraints. The following is the general representation of an LP problem [26]:

$$\begin{aligned} \min \quad & \mathbf{c^T x} \\ \text{s.t.} \quad & \mathbf{Ax} \leq \mathbf{b} \\ & x \geq 0 \end{aligned} \tag{2.2}$$

where,

- $\mathbf{c}$ and $\mathbf{b}$: are integer vectors.

- $\mathbf{A}$: is a matrix of integers.

Here,

$$\mathbf{c^T x} \equiv [c_1, c_2, ..., c_n] \begin{bmatrix} x_1 \\ . \\ . \\ x_n \end{bmatrix} \tag{2.3}$$

An LP is an optimization problem over real valued variables, they are efficient in practice and can be solved in polynomial time. All combinatorial problems can be written as linear programs provided that all variables $x_i \in \mathbf{x}$ take integer values. The optimal solution obtained by the LP will be the optimal solution to a combinatorial optimization problem.

Dantzig et al. [20] were the first to consider VRP as a truck dispatching problem. The problem was to define a route of trucks from a depot to some delivery points which have specific demands. The trucks were under the constraints of time and capacity. The objective was to minimize the total distance travelled by the entire fleet.

2.1.5.2 Set Partitioning

Balinski et al [8] proposed a set partitioning formulation of CVRP. Set partitioning determines how items in one set can be partitioned into smaller sets such that an item is in only one partition. They used this approach to solve a truck delivery problem. The objective was to minimize the travelling cost. They showed that for a larger dataset the number of variables in LP formulation increases due to feasibility of large number of solutions. The solution generated was appropriate only when the

constraints were tight.

There are many other exact algorithm approaches like branch and bound [15, 23, 24, 41, 57] etc.

These traditional solutions [8, 15, 20, 32] provide an optimal output on a very small subset of the problem and have a very high computational cost usually more than 100 seconds to solve for 100 customer .

2.1.5.3 Need for Heuristic and Meta-heuristic Algorithms

Previous studies have developed fast exact algorithms that solve small instances of the VRP in an optimal manner, but, because VRP is an NP-hard problem and real-life applications require solutions to large instances of the problem, techniques to produce faster results are required. Many heuristic and meta-heuristic algorithms have been developed that provide a trade-off between the quality of a solution and the use of computational resources. These algorithms are more suitable for larger instances of the problem which are more suitable for real-world applications.

2.1.6 Heuristic Algorithms

Heuristic algorithms perform a relatively limited amount of exploration of the search space and produce good quality solutions within modest amount of computational time. They can also be extended to fit to different constraints [40]. Classical heuristics include, saving algorithm [17], sweep algorithms [2, 29], cluster-first route-second algorithms [12, 25, 60], improvement heuristics [67] etc.

2.1.6.1 Saving Heuristic

Clarke et al [17] proposed a heuristic for solving VRP based on the concept of saving. Saving is the estimate of cost production obtained by serving two customers sequentially in the same route, rather than in two different routes. The method starts with vehicle routes containing the depot and one other vertex. At each step, two routes are merged in a single route based on the largest savings generated. They solved the problem instance with 50 to 100 customers.

2.1.6.2 Sweep Heuristic

Gillett et. al [29] popularized the sweep heuristic and applied it on a planar instance of VRP. They divided VRP into two tasks, clustering and routing. Feasible clusters were formed by sweeping a ray around a depot which was the origin of all trips. Customers were chosen at random and they were assigned to a vehicle (if the capacity constraints of the vehicle allow it), as a ray was swept through the customer. The process goes on iteratively until a matching was found for all customers. They solved an instance of 21 customer requests in 0.21 seconds and an instance of 250 customer requests in 9.7 seconds.

2.1.6.3 Cluster-First, Route-Second Heuristic

As the name suggests, this heuristic algorithm is divided into two phases. The first phase was to assign customers to clusters from which they are to be assigned to vehicles. The second phase was to compute the minimum cost route of the vehicle within a cluster. Fisher et al. [25] solved a Generalized Assignment Problem (GAP)

to cluster the customers with the objective of approximating the delivery costs. The algorithm always finds a feasible solution if it exists. Once a cluster was determined, TSP was solved optimally using a constraint relaxation approach. The algorithm showed asymptotically optimal behavior. They solved an instance of 50 customers divided into 5 clusters in 9.33 seconds and an instance of 100 customers divided in 10 clusters in 13.6 seconds.

2.1.7 Meta-heuristic Algorithms

Classical heuristic algorithms find quick feasible solutions. The problem with these feasible solutions is that they may have huge discrepancy with the globally optimal solutions. Meta-heuristic algorithms involve deep exploration of the most promising region of the solution space. They do this by combining sophisticated neighborhood search rules, memory structures and recombination of solutions. Although, the methods applied produce good quality solutions, they require huge computational resources. Meta-heuristics are almost similar to improvement heuristics and can be viewed as natural enhancements to classical heuristics [40].
Meta-heuristics can be classified into two main categories:

- Local Search: This method explores the solution space by iteratively moving from a current solution to a promising neighborhood solution until a criteria for stopping is met. Tabu Search and Simulated Annealing are two of the well known local search meta-heuristic algorithms.

- Population Search: This method maintains a pool of solutions, by selecting a promising off-spring and replacing its parents. The pool contains a set of feasible

solution from which the best solutions are selected for further enhancements until a criteria of stopping is met. Typical methods include Genetic Algorithms, Ant Colony Optimization etc.

2.1.7.1 Tabu Search

Glover et al. [30] introduced tabu search in 1998. It performs a local search in the neighborhood of the present solutions and selects the best from the solutions found until the stopping criterion is satisfied. The current solution may deteriorate from one solution to another. To avoid cycling to a previous solution, solutions that are recently examined are declared to be forbidden or tabu for a number of iterations. The tabu status is removed if the tabu solution is better than the previously seen solution. Over the years tabu search has been applied on VRP and its variants by many authors. Gendreau et al. [28] described a tabu search heuristic called TABUROUT which solved CVRP with restrictions on the length of the route. The algorithm exchanges vertices between routes to produce neighboring solution in an iterative manner. It also accepts infeasible solutions which did not meet the capacity or route length constraints so that it can avoid getting trapped in the local minima. They tested their solution on 50 customer requests that took 6 mins to solve and 200 customer requests took 58.8 mins to solve. Cordeau et al. [18] proposed a parallel TS heuristic solution to VRP. The algorithm generated competitive results as compared to the results produced at that time. They tested their solution on 50 to 420 customer requests and found an improvement of 0.6%.

2.1.7.2 Simulated Annealing

Simulated annealing is a common meta-heuristic approach for solving sophisticated optimization problems. The technique is based on an analogy of the annealing process in solids. It is a randomization algorithm and can be asymptotically viewed as an optimization procedure. Chian et al. [13] developed a simulated annealing approach to solve VRP. In their paper they implemented three different types of simulated annealing algorithms and two different kinds of neighborhood structure and compared the results. They tested their solution on 20 to 100 customer requests. Chez et al. [19] proposed a parallel simulated annealing approach to solve VRPTW. They used their initial solutions as the best solutions proposed by previous studies. Each of the parallel threads carried out annealing searches using the same initial solution and cooling schedule. After a certain time interval, the threads cooperate their best solutions to get a global best solution. They tested their solution on 100 customer requests. It took 2 - 3 seconds to solve for 100 customers. Their results were 11% worse than the optimal solution.

2.1.7.3 Ant Colony Optimization

Ant Colony Optimization simulates the behavior of ant colonies in nature as they forage for food and find the most efficient routes from their nest to food sources. ACO is a sub part of Swarm Intelligence (SI) where the natural behavior of social insects is studied and simulated. ACO is applied on combinatorial optimization problems which use self-organizing methods to solve them. Bell et al. [9] used multiple variations of ACO to solve VRP. Their experiments showed that ACP produces good results

within one percent of the known optimum solutions from small datasets. They tested their results on 100-150 customer requests. For larger problem the authors used a multiple ant colony approach which produced competitive results. While most of the work in literature has been done on static instance of VRP, Montemanni et al. [52] worked on VRP in which passengers' requests are received by the system as time progresses and must be incorporated in vehicles' routing schedule. They divided the entire day into time slices in which the customer orders arrive. For every time bucket the ACO algorithm executed on the known requests. The results produced by ACO were competitive to other meta-heuristic techniques for solving VRP and its variants. They tested their results on 50-100 customer requests. Their results were 10% worse than the best known solution.

2.1.7.4 Genetic Algorithm

Just like ACOs, Genetic Algorithm (GA) is an area of study inspired by nature, where the natural process of evolution, introduced by Darwin is simulated to solve real world problems. GAs are used as a generalized method for search and optimization. In literature most authors describe customized mutation and crossover operations used in GAs and prove their efficiency through elaborate experiments. Baker et al. [7] proposed a GA methodology for solving VRP. They viewed the problem as a single depot problem. The algorithm was tested on benchmark problem instances with ranges from 50 - 200 customer requests. The results generated were 0.5% worse then the best known solution and took around 2000 seconds to run. Similarly, Hanshar et al. [37] proposed a GA technique for solving dynamic VRP. They also divided the dynamic problem into time buckets and solved static VRP. Each time slice takes in

new requests and performs GA based optimization. They ran the algorithm on 50-199 instances and found the results comparable to the results obtained by ACO and TS algorithms. Their results where 7% better than other meta-heuristic solutions.

2.1.8 Data-Driven Systems

The major drawback of the above mentioned solutions is that exact algorithms produce optimal solutions but on very small instance of the problem. Heuristics and meta-heuristics on the other hand try to find the best trade-off between the quality of the solution and the computational time required to generate the solution for larger instance of the problem.

The ridesharing problem in reality has a large search space with a demand of dynamic and real-time solutions. Large scale optimization problems often become computationally infeasible. Keeping in mind the NP-hard nature of the problem, fast and efficient algorithms are needed for real world implementations. More recently authors like Ota, Ma, Alonso and Santi have implemented large scale systems of taxi ridesharing, that handle large datasets. The solutions produced are such that the vehicles can transport several customers at the same time. Alonso-Mora et al. [4] developed a greedy algorithm that improves greedy assignment by constrained optimization which converges to an optimal assignment over time. They have developed an assignment heuristic that dynamically process a large number of trip request and provide ridesharing solutions for upto ten customers. The algorithm was tested on a large number of tax-cab rides in New York City (around 3 million rides). The work showed that 80%of taxi trips in Manhattan can be shared by two customers and 98%

of taxi trips can be shared by four passengers. Santi et. al [58] introduced shareability networks for producing ridesharing solution in real-time. They proposed a graph based model that computes optimal sharing strategies for trips and contains two key parameters: the maximum number of trips that can be shared and the maximum delay the customers are willing to tolerate. They showed that 50% of the trips can be shared by two passengers. For producing ridesharing between three or more customers they solve they found an approximation using a heuristic algorithm and concluded that 66% of the trips can be shared between three passengers. Ota et. al [54] proposed Simulating Taxi Ride Sharing (STaRS), a data driven framework that enables the analysis of wide range of ridesharing scenarios. STaRS is an optimization algorithm that is linear in number of trips and makes use of an efficient indexing scheme, which combined with parallelization makes the approach for solving the problem scalable. The simulation engine aims to derive the best ride-sharing scenario based on a set of input parameters in a data-driven fashion, where pick-up requests are derived from historical data. It operates in an event-driven manner and updates its states when a pickup request is issued. Since, the solution is dynamic, it does not produce globally optimal solutions. They found that for three shared trips, the total saving in the total distance through ridesharing is 29%, while for two shared trips the saving is 18%. Ma et al. [49] studied a dynamic ridesharing problem in a practical setting and designed a system called T-Share. In their paper they combine spatio-temporal database techniques, with optimization algorithms to significantly scale down the size of the problem. This results in producing fast results for processing even thousands of customer trip requests. They ran the simulation on over 30,000 taxis in Beijing and concluded that their service can generate a ridesharing solution in 5ms.

2.1.9 Intelligent Systems

Recently authors have started focusing on using artificially intelligent algorithms in the field of transportation. Intelligent algorithms can be divided into supervised learning, unsupervised learning and reinforcement learning. These algorithms are effective as they produce fast and efficient solutions in a data-driven fashion. These algorithms are naturally scalable to large datasets and can be effectively used with real-time systems.

2.1.9.1 Reinforcement Learning

Reinforcement Learning (RL) [64] is used for learning an optimal policy in dynamic environment. An agent takes an action in state, receives a reward, moves to some next state, and then repeats this procedure until it reaches a terminal state. Initially, the agent randomly picks an action from the action space given a state because it has no knowledge of which action has to be taken in a given state. The agent has a choice between exploiting its knowledge by choosing the action with highest estimated value and exploring its environment by taking any other action. The trade-off between exploration and exploitation is crucial. As time progress the agent learns the best set of actions for each given state of the environment and is said to have learned an optimal policy.

Q-Learning [71] is a widely used RL method because of its computational simplicity. Q-Learning is combined with function approximation techniques like deep neural networks (Deep Q-Networks (DQN)) [51] to handle large state spaces. Recently many

authors have adopted Q-Learning as a method to solve VRP. Jindal et al. [39] developed a Q-Learning based system that learns an efficient policy for carpooling that maximizes transportation efficiency so that fewer cars are required to fulfill the given amount of trip demand. Han et al. [36] wrote a paper on routing autonomous vehicles in a real scenario at the scale of the city of Singapore with pick-up and drop-off events for a fleet of 1000 vehicles. They used DQN with an exploration strategy that follows a greedy policy. Through their work they demonstrated that DQN has the ability to master difficult control policies including traffic control and taxi dispatching. Gao et al. [27] used the Q-Learning algorithm to find optimal locations for taxi cab drivers to find passengers in the city of Beijing.

While RL has been adopted to tackle fleet management problem however, the high dynamics between demand and supply can hardly be modeled accurately by traditional RL approaches [45]. One approach is to use a multi-agent RL setting where each vehicle is considered as an agent. Lin et al. [45] proposed two algorithms namely contextual deep Q-Learning and contextual multi-agent actor critic method to address the management of multiple vehicles in a fleet. Multi-agent RL comes with a crux. Each agent is non-stationary as all the agents are learning and affecting the environment at the same time. Hence, most of the studies allow coordination only amongst a small set of agents due to high computational cost.

2.1.9.2 Supervised Learning

Supervised learning is a machine learning algorithm where there is a input variable x and an output variable Y and the algorithm learns the mapping function f from

input to output.

$$f(x) = Y \tag{2.4}$$

The goal is to approximate the mapping function so, when there is some input data x, the output variable Y for that data can be predicted. There are many supervised learning algorithms such as Naive Bayes, Decision Trees, Linear Regression, Support Vector Machines, Neural Networks etc.

In literature Neural Network based models have been used to predict taxi demands [73]. Xu. et al. [73] proposed a Long Short Term Memory (LSTM) based Recurrent Neural Network (RNN) model to predict future taxi request in each area of New York City. The predictions are based on recent demand and other relevant information. Their model out performed other supervised learning models such as a feed-forward neural network. They achieved an accuracy of 83%. Brebisson et al. [21] proposed the use of recurrent neural networks to predict taxi destination given the beginning of taxi trip GPS traces. They compared the performance of different neural network architecture such as multi-layer perceptron, bidirectional recurrent neural networks and model inspired by memory networks. Ma et al. [50] used LSTM neural network to predict travel speed which is in turn desirable for the traffic prediction problem. They concluded that LSTM neural network produced the best results for long term dependency in data with 89% accuracy. Most of the work in the literature for supervised learning in transportation has been related to customer demand forecasting and traffic prediction.

2.1.10 Background for Pointer Network

In this section we explain what a Pointer Network (Ptr-Net) is and detail the background required to understand Ptr-Net.

2.1.10.1 Neural Network

An artificial Neural Network (NN) is a supervised machine learning algorithm inspired by biological neurons. The basic building block of a NN is also called a neuron. It takes an input and generates an output. We explain this with the help of Figure 2.2. The neuron takes in an input of summation of weighted inputs and a bias which is

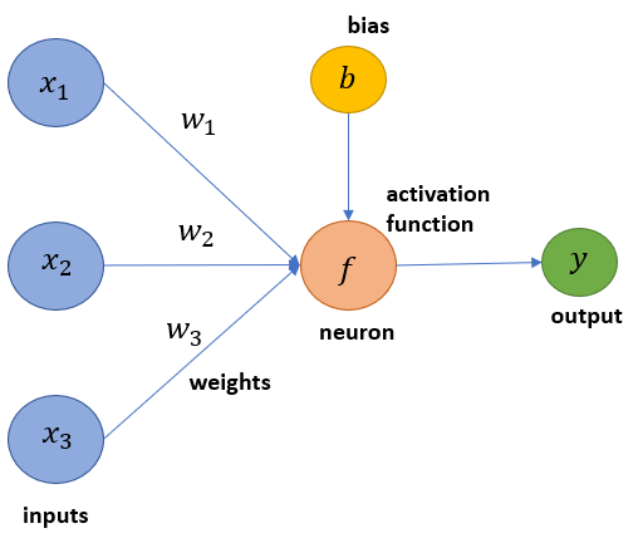

Figure 2.2: Representation of a neuron

then passed through an activation function f. Equation 2.5 and Equation 2.6 shows

the operation of a single neuron.

$$z = \sum_{i=1}^{n} \mathbf{w_i x_i} + \mathbf{b} \tag{2.5}$$

$$f(z) = y \tag{2.6}$$

Here, $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ is a vector of input variables, $\mathbf{w} = \{w_1, w_2, ..., w_n\}$ is a vector of weights where every w_i is associated with x_i, $1 \leq i \leq n$ and b is the bias variable. The function f is known as the activation function and this is the function that learns to map the input with the output.

An activation function of a node defines the output of that node given an input or a set of inputs. It also determines the accuracy and computational efficiency of a NN model. A few types of activation functions are listed as follows:

- **Sigmoid Function**

The sigmoid activation function is a non-linear function that squishes the input within a range of (0,1). It is especially used for NN models to predict the probability of an output. Equation 2.7 represents a sigmoid function and Figure 2.3 depicts its graph.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.7}$$

- **Hyperbolic Tanh Function**

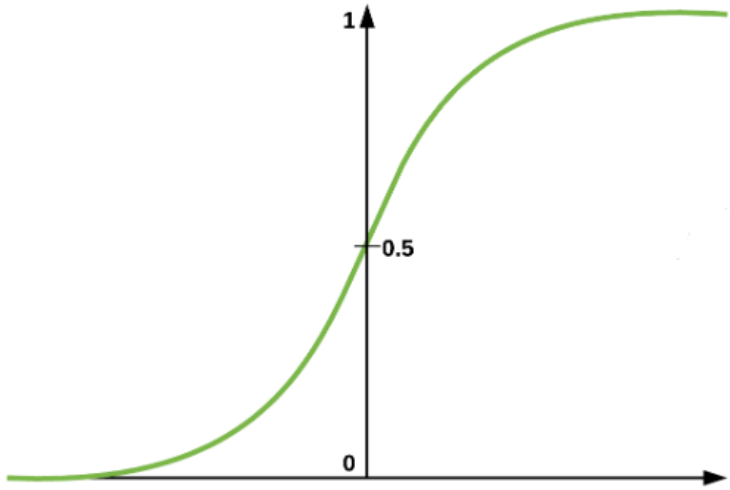

Figure 2.3: Sigmoid activation function

The hyperbolic tanh function is the rescaled version of the sigmoid function. Its output ranges from (-1,1). Equation 2.8 represents a tanh function and Figure 2.4 depicts its graph.

$$tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2.8}$$

- **Softmax Function**

Softmax Function calculates the probability of distribution of events over n different events. The output values range between (0,1). For each vale of input x_i in the input vector $\mathbf{x} = \{x_1, x_2, ..., x_n\}$, the *softmax* value is the exponent of the individual input x_i divided by the sum of exponents of all inputs. Equation 2.9 represents a softmax function.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}, \quad i = 1, .., n \tag{2.9}$$

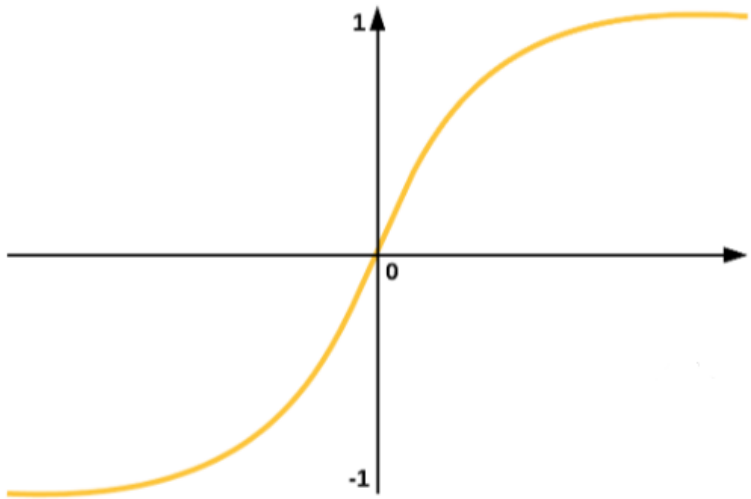

Figure 2.4: Tanh activation function

There are a few other activation functions like: binary step function, ReLU function, Leaky ReLU function etc.

2.1.10.2 Multi Layer Neural Network

So far we have discussed only a single layer NN. In this section we discuss a multi-layer neural network also known as feed-forward neural network. Feed-forward propagation is the process of running the NN to generate the predicted output y. Figure 2.5 represents a multi-layer neural network. A layer is a collection of neurons which take in an input and provide an output. Inputs of each of these neurons are processed through the activation function assigned to the neurons. The NN in Figure 2.5 has three layers. The first layer is the *input layer* because it takes in the inputs. The middle layer is called the *hidden layer* because its values are not observed as the

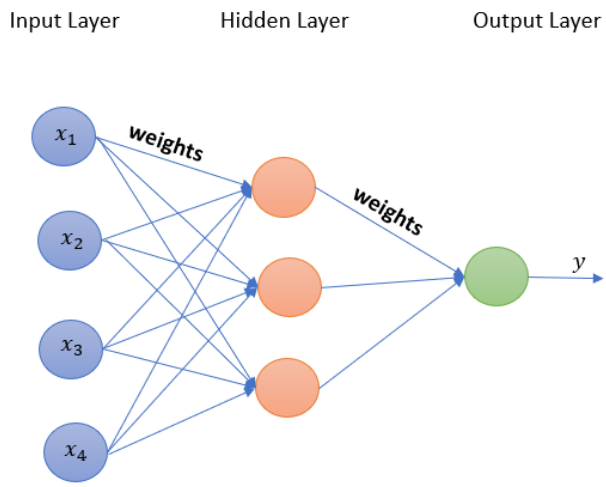

Figure 2.5: Multi-layer Neural Network

output. The third layer is called the *output* as it generates the output.

Training a Neural Network

The output generated by the NN is dependent on the weights and biases of the neurons. These weights and biases are adjusted using backpropagation algorithm (to be discussed). The output of the NN is first evaluated using a cost or a loss function. One of the most commonly used cost functions is the mean squared error cost function (MSE). If y is the predicted output generated by a NN and $\hat{y}$ is the target output then MSE is calculated as follows:

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2 \tag{2.10}$$

Training the network means minimizing the squared loss $(y - \hat{y})^2$. Here, n is the number of inputs.

Backpropagation

Backpropagation is the process by which a NN is trained to improve its prediction accuracy. Training a neural network involves adjusting the weights and biases of the network, this in turn affects the value generated by the activation function within a neuron and consequently updates the predicted output y. The goal of backpropagation is to repeatedly go back and adjust the weights of each preceding layer. Every time the weights are adjusted the network goes through a feed-forward propagation to recalculate the cost function. This process is repeated until the cost function is minimized Figure 2.6. The backpropagation algorithm works by computing the cost function at the output layer. This is done by comparing the predicted output y of the NN and the actual target output $\hat{y}$. It then employs an algorithm called gradient descent [43] (not discussed here because it is beyond the scope of the thesis) to adjust the weights and biases of the neurons present in each successive layer propagating back through the network.

2.1.10.3 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a specialized scheme of NN architecture. RNNs were developed in order to solve learning problems where information about the past (i.e. past events) are directly linked while making future predictions. RNNs make use of sequential information and are used in tasks such as image captioning [10], speech recognition [35], language translation [5], sentiment analysis [65].

The first building block of a RNN is a recurrent neuron Figure 2.7. This neuron

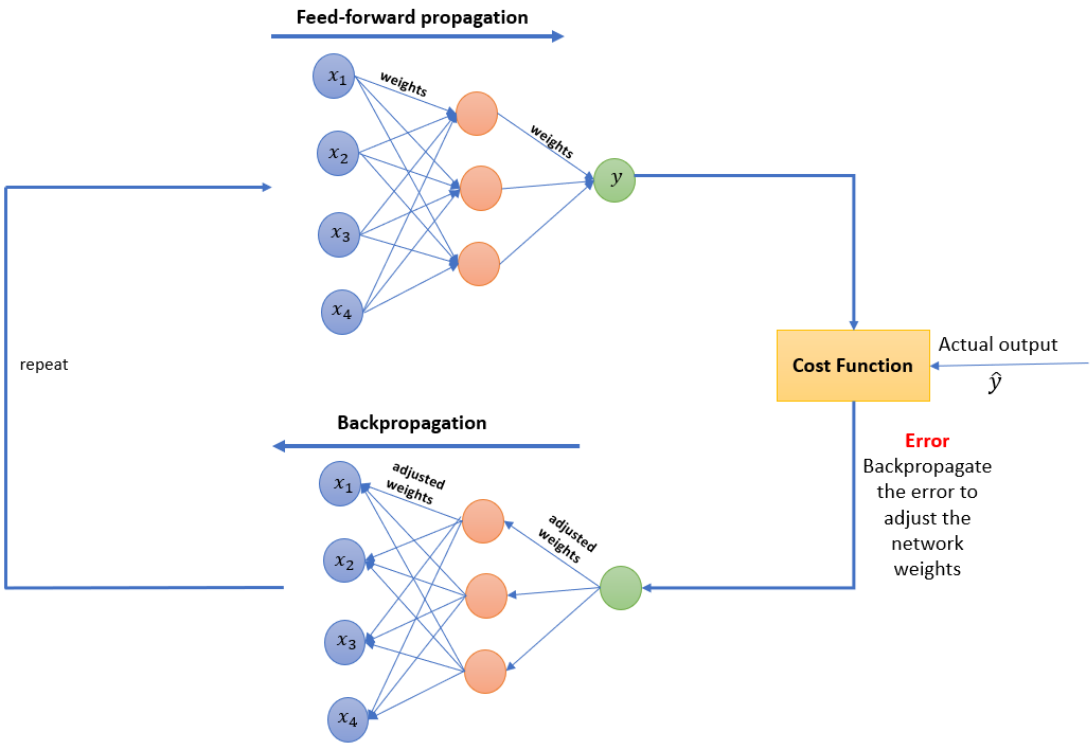

Figure 2.6: Backpropagation

is different from a basic neuron as it maintains a vector of state **h** based on past computations. The states act as memory in the network and helps it learn past events and past sequences. Here, **x** is a vector of input sequences. The recurrent neuron at time t has a state h_t and takes as input the output of the previous state of the neuron $h_{(t-1)}$ which was at time *t-1* in addition to the current input to the neuron x_t. This is better illustrated in Figure 2.8 which shows an unfolding of a RNN. The unfolded RNN shows the flow of information through the recurrent neuron at every time instance. Each time instance is related to each input in the sequence. The output at each time instance (t) is calculated based on the current state $h_{(t)}$ of the recurrent

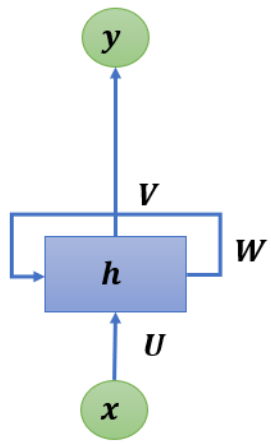

Figure 2.7: A Recurrent Neuron

neuron. This is done with the help of *softmax* activation function to generate output class probabilities. The following equations specify the calculation of hidden states and outputs at each time step during a feed-forward pass of a simple RNN.

$$a_t = \mathbf{W}\mathbf{h}_{(\mathbf{t-1})} + \mathbf{U}\mathbf{x}_\mathbf{t} \tag{2.11}$$

$$h_t = tanh(a_t) \tag{2.12}$$

$$y_t = softmax(\mathbf{V}\mathbf{h}_\mathbf{t}) \tag{2.13}$$

Here, **W** is the matrix of internal weights between hidden layers at adjacent time steps, **U** is the matrix of conventional weights between the input and hidden layer and **V** is the weight matrix between the output layer and the hidden layer. Learning long sequences with RNNs is a difficult problem because of vanishing gradients [11, 38]. In the next section we will talk about Long Short-Term Memory architecture which was designed to handle this problem.

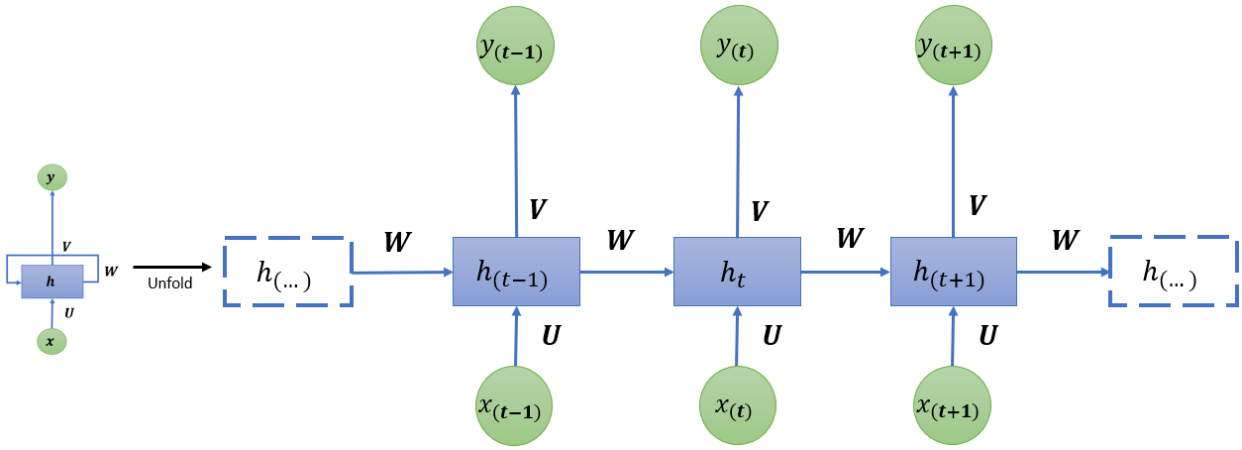

Figure 2.8: Unfolding of a Recurrent Neural Network.

2.1.10.4 Long Short-Term Memory Architecture

Hochreiter et al [38] introduced Long Short-Term Memory (LSTM) architecture for RNNs. They introduced a memory cell (Figure 2.9) which is a unit of computation to replace the traditional recurrent node of previously described networks. LSTMs are very efficient in capturing long term dependencies across a large number of time instances. They do this with the help of a sophisticated memory cell which is made up of an *input gate*, an *internal state*, a *forget gate* and an *output gate*. Information enters through the *input gate*. The *input gate* is responsible for controlling what information gets stored in the memory cell. Next, the *forget gate* regulates how the information from the previous state should persist in the memory cell across time instances. Finally, the *output gate* decides how much information from the memory cell should be generated as an output at any time instance. The detailed description of these states is as follows:

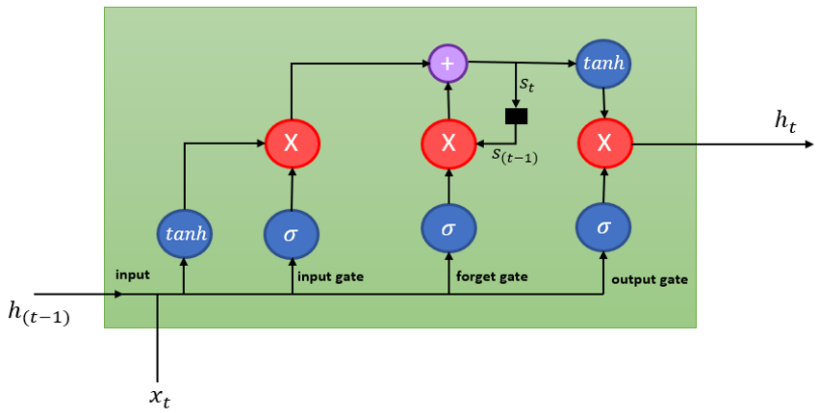

Figure 2.9: LSTM cell architecture.

- **Input gate:** The input gate made up of a *sigmoid* activation function that takes in the current form of input x_t and the hidden state from the previous time step $h_{(t-1)}$:

$$i = \sigma(x_t + h_{(t-1)}) \tag{2.14}$$

The input x_t goes through a *tanh* activation function:

$$g = tanh(x_t + h_{(t-1)}) \tag{2.15}$$

The output g is then element-wise multiplied (denoted by $\odot$) by the output of the input gate i.

$$g \odot i \tag{2.16}$$

This acts as a filter for the gate and helps decide which inputs should be allowed to pass and which should not.

- **Internal state:** The internal state s_t is the memory of the cell. This state is delayed by one time-step and is added to equation 2.18. It provides an internal recurrence loop to learn the relationship between input separated by time.

- **Forget gate:** The forget gate provides a method by which the network can learn to discard the contents of the internal state. It is made up of a *sigmoid* activation function which takes in current input x_t and the hidden state from the previous time step $h_{(t-1)}$:

$$f = \sigma(x_t + h_{(t-1)}) \tag{2.17}$$

The output of the forget gate f is multiplied element-wise by the previous state of he cell $s_{(t-1)}$ to get the new internal state. The forget gate output acts as a weight to the internal state:

$$s_t = f \odot s_{(t-1)} + g \odot i \tag{2.18}$$

- **Output gate:** The final stage of the LSTM cell is the output gate. The output generated is a filtered version of the internal state. The *sigmoid* activation function is fed with the current input x_t and the hidden state from the previous time step $h_{(t-1)}$, to decide which parts of the input are going to be in the output:

$$o = \sigma(x_t + h_{(t-1)}) \tag{2.19}$$

The internal state s_t goes through a *tanh* activation function which is later multiplied element-wise by o to determine which values of the internal state will be in the output of the cell.

$$h_t = tanh(s_t) \odot o \tag{2.20}$$

LSTMs offer a very promising solution to sequence and time-series problems. However, it is difficult to train them. A lot of time and system resources are required to train even a simple model.

2.1.10.5 Sequence-to-Sequence Models

Sequence-to-sequence models were first introduced by [63]. They aim to transform one input sequence (source) to an output sequence (target) where both sequences are of arbitrary lengths. They are used in tasks such as machine translation [47], video captioning [68] etc.

The sequence-to-sequence model uses an encode-decoder architecture. The encoder

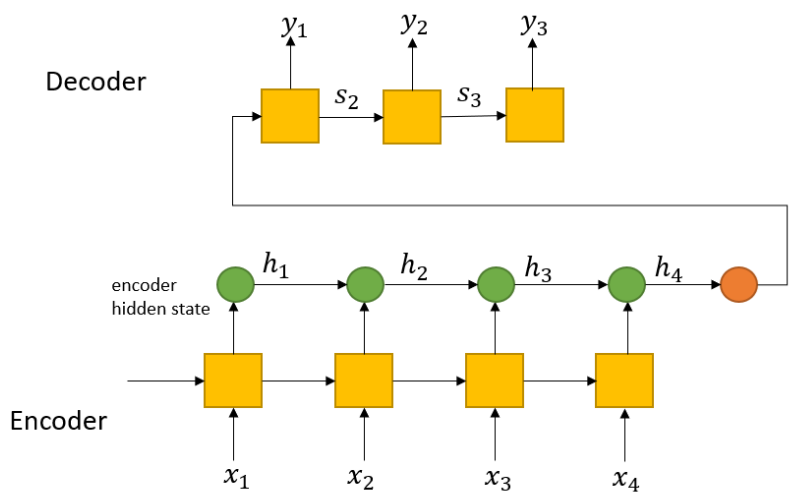

Figure 2.10: Basic Sequence-to-Sequence, Encoder-Decoder model.

takes in an input x_t from the input sequence $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ at time t and produces a hidden state h_t. Thus, the input sequence $\mathbf{x}$ produces an sequence of hidden state

$\mathbf{h} = \{h_1, h_2, ..., h_n\}$ also known as the memory vector. The decoder then uses this memory vector $\mathbf{h}$ to generate the output sequence $\mathbf{y} = \{y_1, y_2, ..., y_m\}$. Both encoder and decoder are made up of RNNs like the LSTM cells. The advantage of this model is that the hidden state of the encoder is fed into the decoder and it captures the summary of the entire input sequence. However, it is not effective for longers sequences and the network becomes lossy (i.e. forgets past dependencies).

2.1.10.6 Attention-Based Models

Attention-based models are variations of sequence-to-sequence encoder-decoder based models that use attention mechanism. Attention mechanism is an interface between the encoder and decoder that provides the decoder with information from every hidden state of the encoder. The model learns longer sequences better than a regular sequence-to-sequence discussed previously. To understand this better, let, $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ be the input sequence being fed into the encoder, $\mathbf{h} = \{h_1, h_2, ..., h_n\}$ be the memory vector generated by the encoder and $\mathbf{y} = \{y_1, y_2, ..., y_m\}$ is the output sequence. The memory vector $\mathbf{h}$ is used to generate a context vector $\mathbf{c} = \{c_1, c_2, ..., c_m\}$. Each context c_t is generated for every time instance in output sequence $\mathbf{y}$. Figure 2.11. shows the way the context vector $\mathbf{c}$ depends on the attention mechanism being used.

Bahdanau Attention Mechanism

One of the most commonly used attention mechanism is called Bahadanau Attention Mechanism [6] (Figure 2.12). Here, similar to $\mathbf{h}$, $\mathbf{s} = \{s_1, s_2, .., s_m\}$ is a vector of hidden states of the decoder. For every time instance of generating an output, the Bahdanau attention mechanism starts by calculating alignment scores using the *tanh*

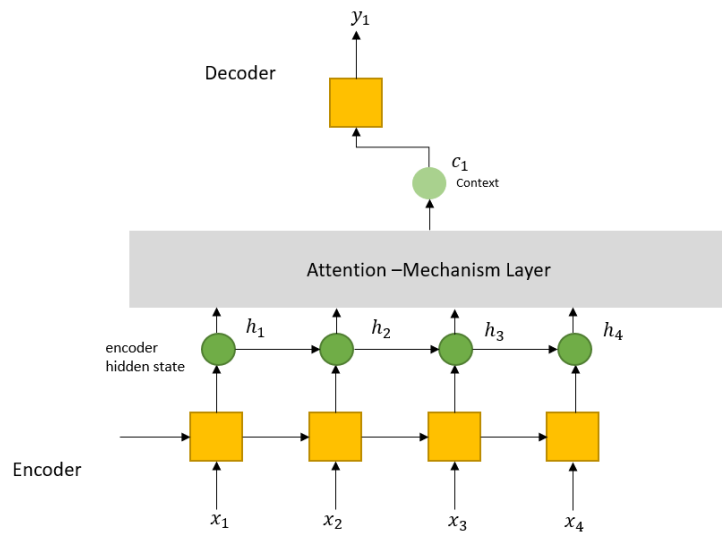

Figure 2.11: Attention mechanism is sequence-to-sequence model

activation function Equation 2.21.

$$\mathbf{a}(s_{(t-1)}, h_j) = tanh(s_{(t-1)} + h_j), \qquad j = 1, ..., n \tag{2.21}$$

Here, $\mathbf{a}$ is a variable that represents the *tanh* operation and $s_{(t-1)}$ is the hidden state of the decoder at the previous time instance i.e. $t-1$. The scores are then fed in to a *softmax* activation function which generates a vector of probabilities of the scores (Equation 2.22). These scores are then element-wise multiplied by the hidden state vector of the encoder $\mathbf{h}$. The values generated are then added together to form the context c_t (Equation 2.23). The context is then concatenated with the previous

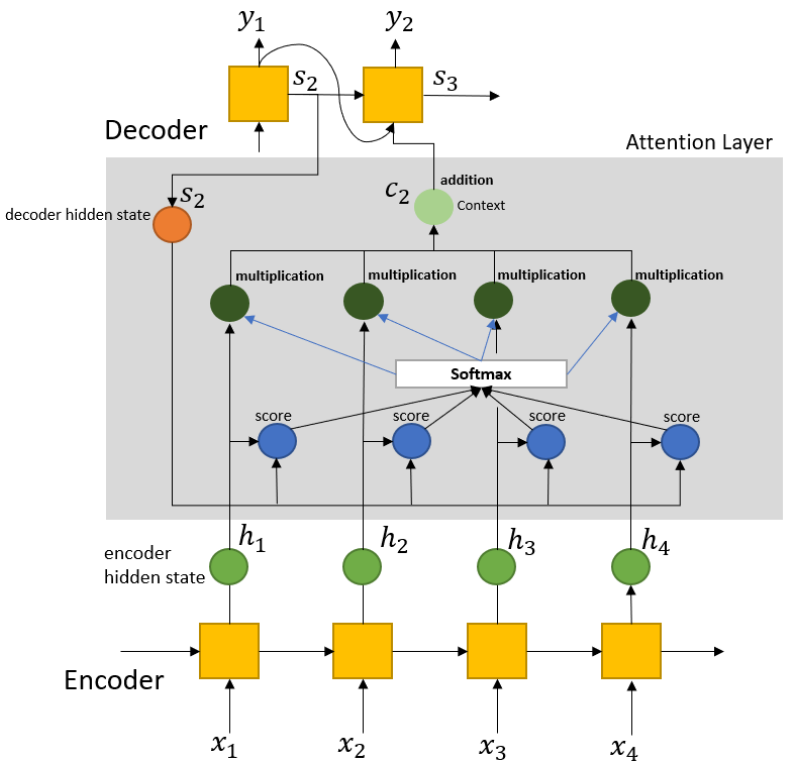

Figure 2.12: Bahadanau Attention Mechanism

decoder outputs to generate the new output.

$$\alpha_{t,j} = \frac{exp(\mathbf{a}(s_{(t-1)}, h_j))}{\sum_{j=1}^{n} exp(\mathbf{a}(s_{(t-1)}, h_j))} \tag{2.22}$$

$$c_t = \sum_j \alpha_{t,j} h_j \tag{2.23}$$

2.1.10.7 Pointer Network

The attention mechanism sequence-to-sequence models have a limitation that the output sequence generated is always of a fixed size. These models cannot be used to solve combinatorial optimization problems as the length of the output sequence depends on the position of the elements and the length of the input sequence. Vinyals et al. [69] proposed Pointer Network (Ptr-Net) to tackle this problems. Ptr-Net is a sequence-to-sequence model in which the output elements corresponds to a position in the input sequence. Instead of using attention to blend hidden states into a context vector (Figure 2.12), the Ptr-Net applies attention over the input elements to pick one as the output at each decoder step (Figure 2.13). Given a training pair $(\mathbf{x}, \mathbf{y})$,

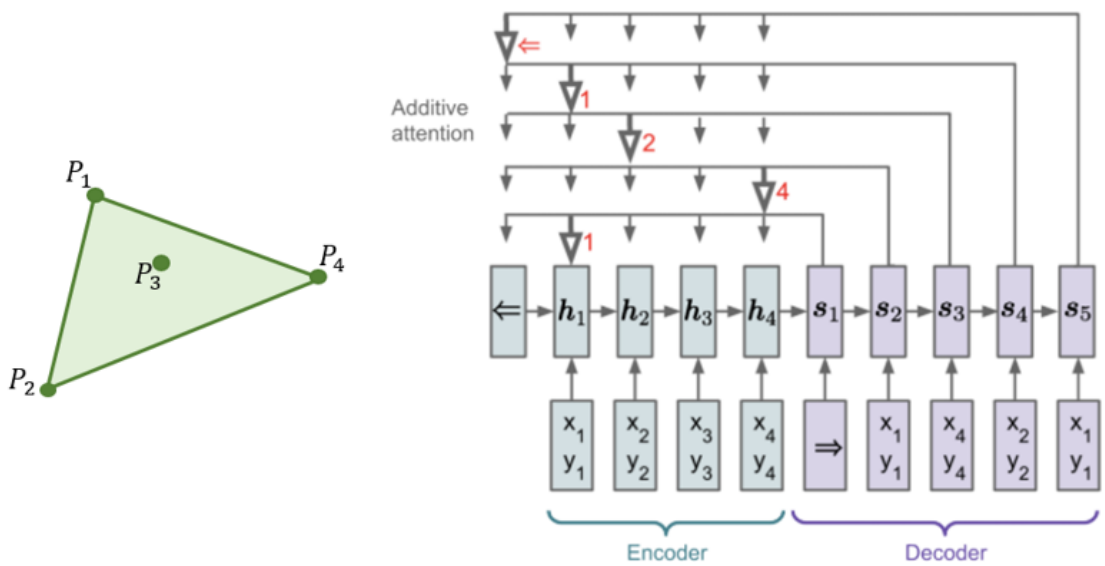

Figure 2.13: Architecture of Pointer Network model.

the model learns to map an input sequence $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ to an output sequence

$\mathbf{y} = \{y_1, y_2, ..., y_m\}$ which is a sequence of m indices, each between 1 to n. The encoder's hidden states are $\mathbf{h} = \{h_1, h_2, ..., h_n\}$ and the decoder's hidden sates are $\mathbf{s} = \{s_1, s_2, ..., s_m\}$. Ptr-Net applies attention between states using the *tanh* activation function and then normalizes it by using $softmax$ to generate the output conditional probability Equation 2.24 and Equation 2.25.

$$\mathbf{a}(s_{(t-1)}, h_j) = tanh(s_{(t-1)} + h_j) j = 1, .., n \tag{2.24}$$

$$P(y_i | y_1, y_2, ..., y_{i-1}, \mathbf{x}) = softmax(\mathbf{a}), j = 1, .., n \tag{2.25}$$

Here, $\mathbf{a}$ is a variable that represents the *tanh* operation and P in the probability. The probability is said to *point* to an element in the input sequence. The attention mechanism is simplified in Figure 2.14, as Ptr-Net does not blend the encoder states into the output with attention weights. In this way, the output only responds to the positions of the input elements.

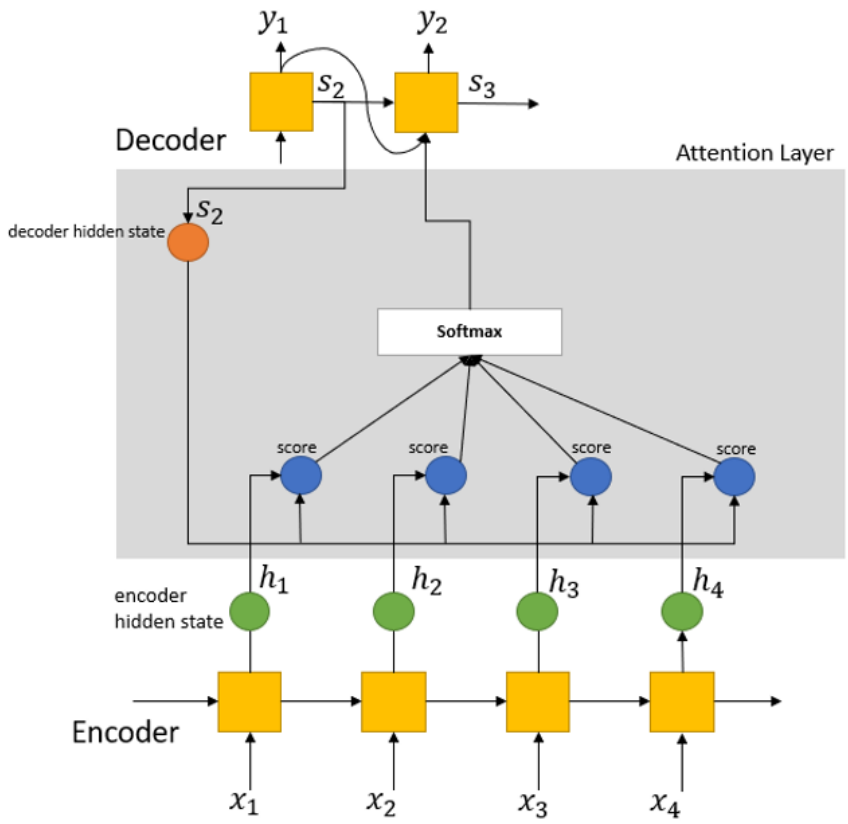

Figure 2.14: Attention in a pointer network model

Chapter 3

Problem Definition and Objective

Ridesharing refers to the method of transportation by which individual travelers share a vehicle for a trip. The problem of ridesharing tries to match customers and vehicles subject to constraints. These matchings are found by minimizing the total travel time of the fleet of vehicles available hence, it is also a formulation of the Vehicle Routing Problem (VRP). In this thesis we aim to solve a particular case of VRP called Capacitated Vehicle Routing Problem (CVRP) [2, 14, 31, 42, 46, 60], in which capacity restriction on vehicles are imposed. We solve two variants of CVRP where in the first case the vehicles in the fleet have same the capacity i.e. the vehicle fleet is homogeneous [2, 46, 60]. In the other case the vehicles in the fleet have varying capacity i.e. the vehicle fleet is heterogeneous. This problem is also known as Heterogeneous Fleet Vehicle Routing Problem (HFVRP) [14, 31, 42]. We solve a static and a deterministic case of the problem. Customer trip requests comprise of the customer's pick-up location, drop-off location and their demand which are known in advance. The objective is to minimize the total distance travelled by the entire fleet

while maintaining capacity constraints.

In this chapter we define CVRP and HFVRP. We also explain our approach towards solving the two problems. We describe the dataset used while solving the problems. We detail three methods of solving CVRP out of which one is based on greedy algorithm and the other two on linear programming (LP) algorithms. After that we describe two LP based algorithms to solve HFVRP. These solutions are used to generate training dataset for Pointer Network (Ptr-Net) models (described in Section 2.1.10.7). Ptr-Net is a supervised machine learning algorithm used for solving combinatorial optimization problems such as convex hull, delaunay triangulation etc. In this thesis we use a regular Ptr-Net model to solve CVRP and HFVRP. Later we describe an enhanced model of Ptr-Net that uses bidirectional long-short term memory (BLSTM) networks and train it to solve CVRP and HFVRP.

3.1 Problem Definition

3.1.1 Capacitated Vehicle Routing Problem

The classical formulation on CVRP defines it as a graph theoretical problem [56], where $G = (V, E)$ is a complete undirected graph, where $V = \{v_0, v_1, ..., v_n\}$ is the vertex set of $n+1$ modes where n is the number of customers. $E = \{e_{01}, e_{02}, ..., e_{n-1n}\}$ where $e_{ij} = \{v_i, v_j\}$ and $i \neq j$, for $0 \leq i, j \leq n$ is the edge set. Vertex v_0 denotes the depot where all vehicles start and end their rides. All the other vertices in $V_c = V \setminus \{v_0\}$ correspond to a single customer's drop-off location with each having a non-negative capacity demand $d_{v_i} \geq 0$. These customers are to be serviced by

unlimited number of k vehicles having identical capacity $C > 0$ such that $d_{v_i} \leq C$. The cost of traveling on edge e_{ij} is represented by $w_{e_{ij}}$ and it is assumed to be symmetric, i.e. $w_{e_{ij}} = w_{e_{ji}}$. Each customer location should be serviced by a single vehicle. The solution consists of a partition of V_c into k sets $\{r_1, r_2, ..., r_k\}$, each satisfying, $\sum_{j \in r_i} d_{v_j} \leq C$. It is a union of k cycles (Equation 3.1) whose intersection is the depot v_0 (Equation 3.2). Each cycle corresponds to a customer assignment serviced by one of the k vehicle.

$$r_i \bigcup_{i=1}^{k} r_j = V_c \quad , \forall 1 \leq i, j \leq n, i \neq j \tag{3.1}$$

$$r_i \bigcap_{i=1}^{k} r_j = v_0 \quad , \forall 1 \leq i, j \leq n, i \neq j \tag{3.2}$$

By associating a binary variable $x_{e_{ij}}$, we get a LP formulation as follows:

$$\min \sum_{e_{ij} \in E} w_{e_{ij}} x_{e_{ij}} \tag{3.3}$$

$$\text{s.t.} \sum_{e_{0j} \in E} x_{e_{0j}} = 2k, \tag{3.4}$$

$$\sum_{e_{ij} \in E} x_{e_{ij}} = 2 \quad \forall i = 1, .., n, \tag{3.5}$$

$$\sum_{e_{ij} \in E} x_{e_{ij}} \geq 2 \left\lceil \frac{\sum_{i \in V_c} d_{v_i}}{C} \right\rceil \tag{3.6}$$

$$x_{e_{ij}} \in \{0, 1\} \quad \forall e_{ij} \in E \tag{3.7}$$

Equation 3.3 is the objective function which tries to minimize the cost $w_{e_{ij}}$ of travelling an edge . When $x_{e_{ij}}$ is 1 for an edge e_{ij} then that edge is a part of the solution and when it 0 for an edge e_{ij} then that edge is not included in the solution. Constraint 3.4 states that a vehicle has to leave and enter a depot. Since each cycle r_m, $(m = 1, .., k)$ in the solution is serviced by a single vehicle, the vehicle must leave the depot by travelling an edge that connects the depot to the first customer location and the vehicle should come back to the depot after servicing the last customer by travelling an edge connected to the last customer location and the depot. Hence, in every cycle r_m the solution consists of two edges connected to the depot and in the overall solution there are $2k$ such edges. Constraint 3.5 is the degree constraint suggesting that each customer location should be visited once by the vehicle and the vehicle leaves that location after visiting it. Constraint 3.6 ensures that no assignment has total demand exceeding the capacity C. It is also known as *capacity constraint*. Constraints 3.7 enforce that each edge is traversed at-most once.

3.1.2 Heterogeneous Fleet Vehicle Routing Problem

Heterogeneous Fleet Vehicle Routing Problem (HFVRP) tries to find the minimal cost assignment for a fleet of vehicles with varying capacity. HFVRP is a generalization of CVRP, where the assumption that all vehicles have same capacity is relaxed. Similar to the definition of CVRP described in section 3.1.1, Let, $G = (V, E)$ be an undirected graph, with $V = \{v_0, v_1, ..., v_n\}$ is the vertex set one $n + 1$ nodes, where n is the number of customer locations. $E = \{e_{01}, e_{02}, ..., e_{n-1n}\}$ where $e_{ij} = \{v_i, v_j\}$ and $i \neq j$, for $0 \leq i, j \leq n$ is the edge set. Vertex v_0 denotes the depot where all vehicles start and end their rides. The vector set $V_c = V \setminus \{v_0\}$ corresponds to customer drop-off

locations with non-negative capacity demand $d_{v_i} \geq 0$. It is assumed that there are unlimited number of vehicles in the fleet each with capacity C_s where $s = \{1, 2, ..., T\}$ represents capacity of the vehicle such that $(C_1 < C_2 < ... < C_T)$. The cost of traveling an edge e_{ij} is represented by $w_{e_{ij}}$ and it is assumed to be symmetric, i.e. $w_{e_{ij}} = w_{e_{ji}}$. Each customer must be serviced by a single vehicle and no vehicle can service a set of customers when demand exceeds the capacity of the vehicle i.e. $d_{v_i} \leq C_s$. There are two decision variables $x^s_{e_{ij}}$ and $y^s_{e_{ij}}$. Variable $x^s_{e_{ij}}$ is equal to 1 when vehicle of type s travels the edge e_{ij} and 0 otherwise. The other variable is the flow variable $y_{e_{ij}}$ which is the vehicle load form v_i to v_j. The mathematical programming formulation of the problem is as follows [42]:

$$\min \sum_{s=1}^{T} \sum_{i=0}^{n} \sum_{j=0}^{n} w_{e_{ij}} x^s_{e_{ij}} \tag{3.8}$$

subject to,

$$\sum_{s=1}^{T} \sum_{i=0}^{n} x^s_{e_{ji}} = 2 \quad j = 1, 2, ..., n; \quad i \neq j \tag{3.9}$$

$$\sum_{s=1}^{T} \sum_{i=0}^{n} x^s_{e_{ij}} - \sum_{s=1}^{T} \sum_{l=0}^{n} x^s_{e_{jl}} = 0 \quad j = 0, 1, ..., n \tag{3.10}$$

$$\sum_{i=0}^{n} y_{e_{ij}} - \sum_{l=0}^{n} y_{e_{jl}} = d_{v_j} \quad j = 0, 1, ..., n, \tag{3.11}$$

$$\sum_{i=0}^{n} y_{e_{i0}} = 0 \tag{3.12}$$

$$y_{e_{ij}} \leq \sum_{s=1}^{T} x^s_{e_{ij}} C_s \quad i \neq j = 0, 1, ..., n, \tag{3.13}$$

$$y_{e_{ij}} \geq 0 \quad i \neq j = 0, 1, ..., n, \tag{3.14}$$

$$x^s_{e_{ij}} \in \{0,1\} \quad i \neq j = 0,1,...,n; \\ k = 1,...,T \tag{3.15}$$

Constraints 3.9 ensures that each customer present at a vertex location is serviced once by one vehicle. Constraint 3.10 guarantees that the vehicle leaves the location that it has arrived at. If vehicle of type k services a customer at vertex v_j by traveling the edge e_{ij} in which case $x^k_{e_{ij}}$ will be equal to 1 then, it must leave that customer location by travelling the edge e_{jl} in which case $x^k_{e_{jl}}$ will be equal to 1. Constraint 3.11 ensures that the customer demand for vertex v_j has been satisfied. When vehicle k with a certain load say, c_k travels to vertex v_j then after leaving the vertex location its load is reduced by d_{v_j}. Constraint 3.12 applies that after servicing the last customer the vehicle returns to the depot empty signifying that all customer demands have been satisfied. Constraint 3.13 ensures that total load from v_i to v_j does not exceed the capacity of the vehicle. Constraint 3.14 implies that the load of the vehicle while traveling from vertex v_i to v_j cannot be negative.

3.2 New York Yellow Taxi Dataset

We solve the above described problem using the New York Yellow Taxi dataset. The New York City Taxi and Limousine Commission (TLC) records the trips made by New York City's yellow taxis. Customers hail these taxis by signaling to a driver who is on duty and seeking a passenger. They do this by using e-hailing apps. The dataset contains the following attributes [16].

- **id** - a unique identifier for each trip

- **vendor_id** - a code indicating the provider associated with the trip record.
- **pickup_datetime** - date and time when the meter was engaged.
- **dropoff_datetime** - date and time when the meter was disengaged.
- **passenger_count** - the number of passengers in a vehicle.
- **pickup_longitude** - the longitude where the meter was engaged.
- **pickup_latitude** - the latitude where the meter was engaged.
- **dropoff_longitude** - the longitude where the meter was disengaged.
- **dropoff_latitude** - the latitude where the meter was disengaged.
- **store_and_fwd_flag** - indication of whether the trip record was held in the vehicle's memory before sending it to the vendor.
- **trip_duration** - duration of the trip in seconds.

There are other popular datasets like Solomon [61] which contain 56 instance of 100 customers, Augerat which contains 74 instance of 15 - 100 customers, Chrisrofides 15 instance of 13 - 101 customers, Fisher 3 instances of 45 - 135 customers, etc. [55]. Although these datasets have varied examples of closely clustered customer location and evenly distributed customer locations, they do not have enough instances to train a neural network model and to show the speed benefits of generating solutions for more than a 1000 requests at a time. Hence, we chose the New York Yellow Taxi dataset because it has real word representation of taxi rides and a large number of instance of VRP instances can be generated from it as it contains more than a million

ride request for just one year.

Our work is only concerned with the **pickup** and **drop-off** locations of taxi trips which are given in the form of longitude and latitude of each location . The visualization of the taxi pickup and drop-off locations is shown in Figure 3.1. Here, the red

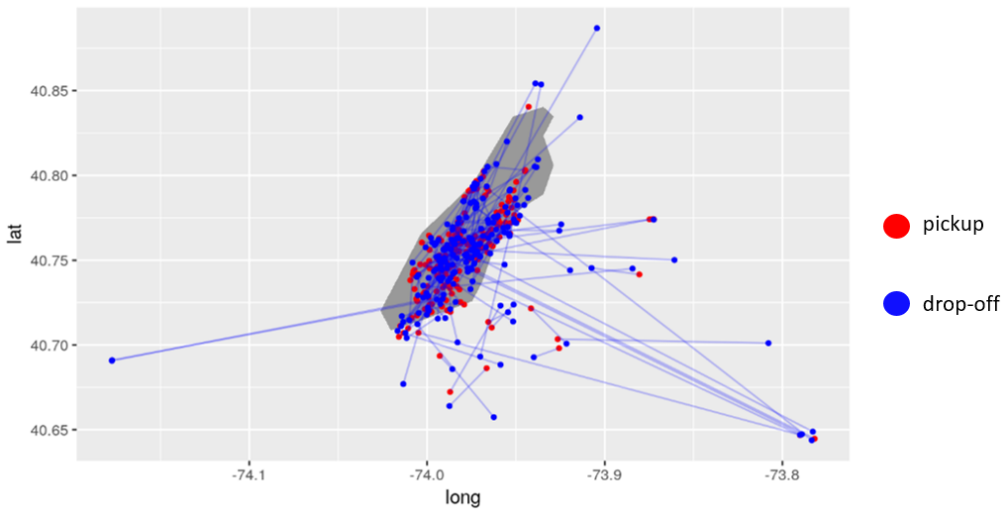

Figure 3.1: Pickup and dropoff in New York Yellow Taxi Dataset.

dots depict the pickup location of a trip and the blue dot depicts the drop-off location of a trip. A straight line connects the two locations, which is the Euclidean distance between them. To extract the data to generate ridesharing matches, we divide the Manhattan area into $200X200$ meters grid cells.

Let,

- n - total number of requests originating in a grid cell
- o_i - origin location of trip requests within a grid cell, $i = 1, 2, ..., n$
- v_i - destination location of trip requests, $i = 1, 2, ..., n$

- v_0 - the centroid of all starting location of trip requests.

We extract all the trips o_i originating in a grid cell and then calculate the centroid of those locations. The centroid acts as the depot v_0 for that particular grid cell and becomes the origin of all trips. The depot is set as the centroid because we assume that it reduces the amount of distance travelled by the fleet to pick up the customers before they begin the trip to their destination. Thus, we obtain trips originating in a grid cell denoted by a single location and ending in other parts of New York city. Figure 3.2 depicts such trips.

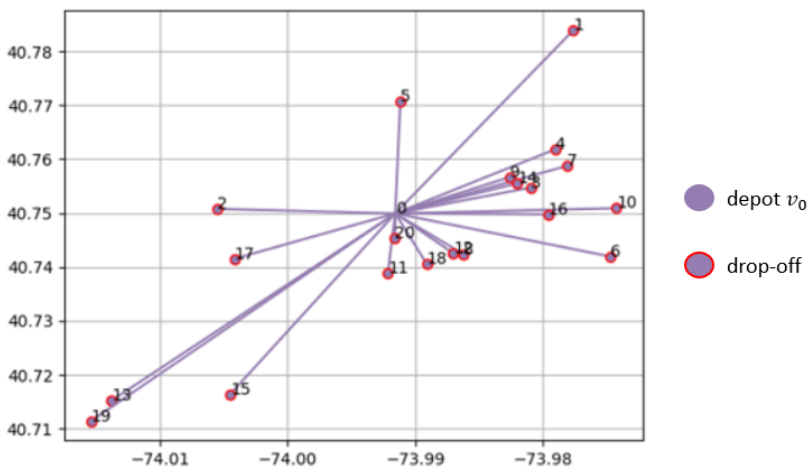

Figure 3.2: Example of trips originating in a cell.

3.3 Solving CVRP

In this section we introduce our approach for solving the CVRP. The first method uses a greedy algorithm based approach and there are two more methods based on

LP. The greedy algorithm approach is a quick heuristic for solving the CVRP. The LP based methods provide the optimal solution given the objective and the constraints. These methods are applied to the New York Yellow Taxi dataset to generate a training dataset for Ptr-Net models.

3.3.1 Greedy Solution

We designed a greedy algorithm based solution which finds ridesharing matches for customers and vehicles. We implemented a greedy heuristic because it generates approximate results in a fast manner. This helped comparing the trade-offs between the quality of the solution and the time it takes to generate the solution against other algorithms. We assume that the number of customers are n and the capacity of the vehicles in the fleet is uniform which is C. V is the set of vertices of customer drop-off locations and the depot. The matches produced are stored in a sequence M which is later processed to extract the grouping of customers with vehicles.

This greedy algorithm matches the first C closest customers together in a vehicle, then the next C and so on until all the n customers have been assigned to a vehicle. Since the fleet of the vehicles has homogeneous capacity the number of customers n is a multiple of vehicle capacity C.

This algorithm is applied to each grid cell extracted from the New York Yellow Taxi data set containing the depot location and customer drop-off locations. The ridesharing matches generated are used to create a dataset named Greedy Heuristic which will be used to train the Ptr-Net models.

Algorithm 3.1 Greedy Algorithm for Ridesharing Matches

1: $M \leftarrow \emptyset$
2: $TableT \leftarrow \emptyset$
3: $nodes \leftarrow V$
4: Create a set $S = \{S_1, ..., S_m\}$, where m = $\binom{n}{C}$, $S_i \in S$ is a set of customers and C is the capacity of the vehicle
5: **for all** $S_i \in S$ **do**
6: Generate a set $\{P_1, ..., P_l\}$ where $l = C!$ and each $P_j \in \{P_1, ..., P_l\}$ is a permutation of S_i
7: Calculate the distance travelled between nodes in each P_j
8: Find P_j with least travel distance $dist_j$
9: $TableT(i) \leftarrow insert(S_i, P_j, dist_j)$
10: **end for**
11: **while** $TableT \neq \emptyset$ **do**
12: Find P_{j*} with $min(\boldsymbol{dist})$ in $TableT$
13: $M \leftarrow P_{j*}$
14: Delete all records form $TableT$ that have in P_j any of the $nodes$ in P_{j*}
15: **end while**

3.3.2 Linear Programming Solutions

In this section we describe a Linear Programming (LP) based exact method for generating ridesharing solutions. Heuristic solutions like greedy may produce a viable solution but it may not always be optimized. Hence, we chose LP because it performs optimization on problems based on few simplifying constraints. It is very effective if the problem size is small. Since, in our experiments as discussed in Chapter 4 we solve small a instance of the problem therefore we chose LP as a technique to solve CVRP.

We provide two types of LP based solutions for solving CVRP. The difference between them is based on how the distance is calculated while minimizing the total distance travelled by the fleet. In these formulations we have assumed that the capacity of the vehicles C is 3. Hence, the formulation tries to share a vehicle between three

customers. Let,

- k - number of vehicles
- n - total number of customer requests
- x, y, z - customers related to their drop-off locations related to v_x, v_y and v_z, where $x, y, z \in \{1, ..., n\}$
- t_i^x - $\begin{cases} 1, & \text{if customer } x \text{ is seated in vehicle } i,\ i \in \{1, ..., k\} \\ 0, & \text{otherwise} \end{cases}$
- t_i^y - $\begin{cases} 1, & \text{if customer } y \text{ is seated in vehicle } i,\ i \in \{1, ..., k\} \\ 0, & \text{otherwise} \end{cases}$
- t_i^z - $\begin{cases} 1, & \text{if customer } z \text{ is seated in vehicle } i,\ i \in \{1, ..., k\} \\ 0, & \text{otherwise} \end{cases}$
- $p_i^{xyz} = t_i^x.t_i^y.t_i^z$ - $\begin{cases} 1, & \text{if customer } x, y \text{ and } z \text{ are seated in vehicle } i,\ i \in \{1, ..., k\} \\ 0, & \text{otherwise} \end{cases}$
- d^{xyz} = distance between customer x, y and z

The formulation then becomes:

$$\min \sum_{i=1}^{k} \sum_{x=1}^{n-2} \sum_{y=x+1}^{n-1} \sum_{z=y+1}^{n} d^{xyz} p_i^{xyz} \tag{3.16}$$

subject to,

$$\sum_{i=1}^{k} t_i^x = 1, \quad \forall x \tag{3.17}$$

$$\sum_{x=1}^{n} t_i^x = C, \quad \forall i \tag{3.18}$$

$$t_i^x, t_i^y, t_i^z, p_i^{xyz} \in \{0, 1\} \tag{3.19}$$

Constraint 3.17 specifies that each customer can only sit in one vehicle. Constraint 3.18 specifies that each vehicle can only accommodate a fixed number of customers.

3.3.2.1 Calculating Distance

The distance travelled variable d^{xyz} can be calculated in many ways. Depending on the objective of how we want to minimize the total travel distance, we propose two ways of calculating d^{xyz}.

- **Method 1**

In this method we try to minimize the travel distance between three passengers. Given passengers x, y and z, there are six ways to travel between the three. Out of which three will have different total distance travelled. For example, two of the orders are: $x \rightarrow y \rightarrow z$ and $z \rightarrow y \rightarrow x$ both will produce the same distance travelled. However, travel orders $y \rightarrow x \rightarrow z$ and $x \rightarrow z \rightarrow y$ will produce a different amount of distance travelled this is more clearly depicted in Figure 3.3.

For calculating distance d^{xyz}, we generate a 3-dimensional matrix Equation 3.20. The matrix stores the minimum travel distance between passengers x, y and z by the order

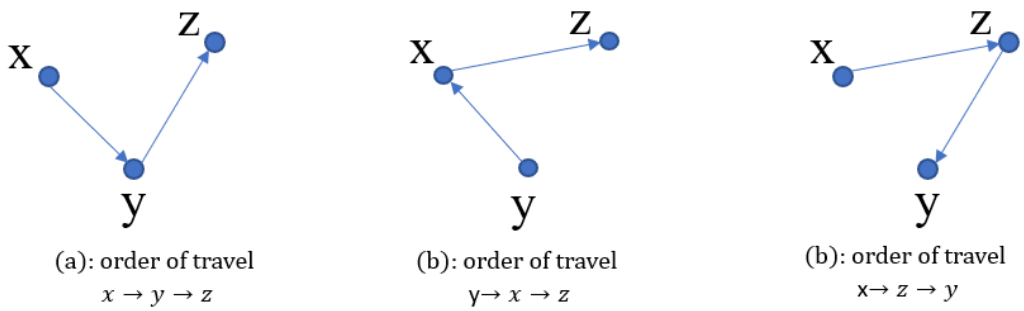

Figure 3.3: Possible order of travel for 3 passengers.

of travel shown in Figure 3.3.

$$distance\_matrix[x][y][z] = min \begin{cases} distance(x,y) + distance(y,z) \\ distance(y,x) + distance(x,z) \\ distance(x,z) + distance(z,y) \end{cases} \quad (3.20)$$

Here, $distance()$ is the Euclidean distance between two nodes. This algorithm is applied to each grid cell extracted from the New York Yellow Taxi dataset which contains the depot location and customer drop-off locations. The ridesharing matches generated are stored in a dataset named Linear Programming without Depot (LPWD) because the distance travelled to and from the depot is not taken into account when generating the distance matrix. This data will be used to train the Ptr-Net models.

- **Method 2**

In this method we try to minimize the total distance travelled between three passengers as well as the distance covered while travelling to and from the depot. The method is compared with **method 1** (described before) to see if the distance between

customer drop-off locations are significant as compared to the distance travelled between the depot and the customers. In this method there are six different orders of travel between customers and out of which only three will produce different values for distance covered. These orders are: $v_0 \rightarrow x \rightarrow y \rightarrow z \rightarrow v_0$, $v_0 \rightarrow y \rightarrow x \rightarrow z \rightarrow v_0$ and $v_0 \rightarrow x \rightarrow z \rightarrow y \rightarrow v_0$.

For calculating distance d^{xyz}, we generate a 3-dimensional matrix Equation 3.21. The

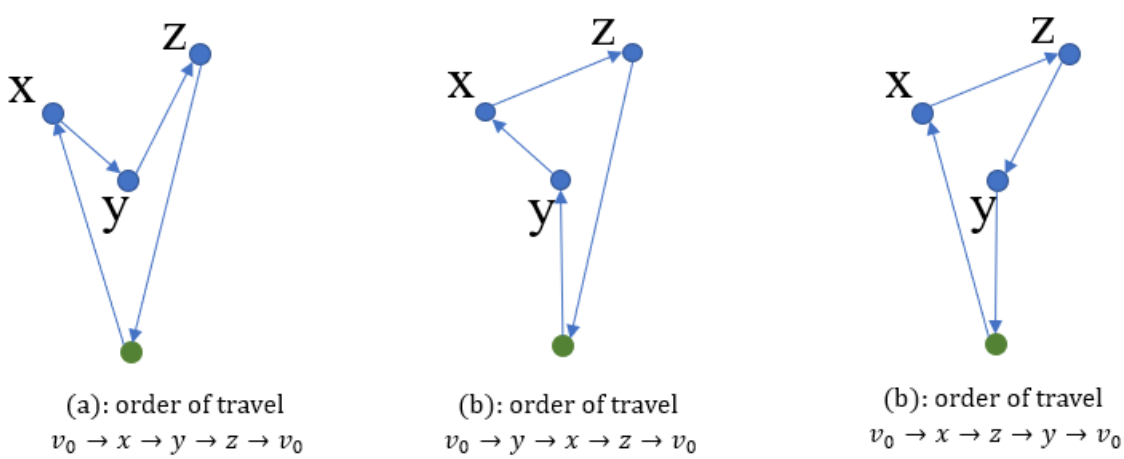

Figure 3.4: Possible order of travel for 3 passengers including the depot.

matrix stores the minimum travel distance between passengers x, y, z and v_0 based the order of travel as shown in Figure 3.4.

$$distance\_matrix[x][y][z] =$$
$$min \begin{cases} distance(v_0, x) + distance(x, y) + distance(y, z) + distance(z, x_0) \\ distance(v_0, y) + distance(y, x) + distance(x, z) + distance(z, x_0) \\ distance(v_0, x) + distance(x, z) + distance(z, y) + distance(y, x_0) \end{cases} \tag{3.21}$$

Here, $distance()$ is the Euclidean distance between two locations. This algorithm is

applied on each grid cell extracted from New York Yellow Taxi dataset. The ride sharing matches are stored in a dataset named Linear Programming with Depot (LPD). This data will be used to train the Ptr-Net models.

3.3.3 Solving HFVRP

3.3.4 Linear Programming Solutions

In this section we explain our approach to solve HFVRP. We have devised two LP based methods to generate ridesharing matches for customers who are being served by a heterogeneous fleet of vehicles. The formulations gives an optimal assignment while trying to minimize the total distance travelled by the entire fleet. In this formulation we have assumed that maximum capacity C_i for each vehicles in the fleet is 2 or 3. Hence, the vehicles are to be shared between 2 or 3 customers.
Let,

- k - number of vehicles
- n - total number of customer requests
- x, y, z - customers related to their drop-off locations related to v_x, v_y and v_z, where $x, y, z \in \{1, ..., n\}$
- t_i^x - $\begin{cases} 1, & \text{if customer } x \text{ is seated in vehicle } i,\ i \in \{1, ..., k\} \\ 0, & \text{otherwise} \end{cases}$

- t_i^y - $\begin{cases} 1, & \text{if customer } y \text{ is seated in vehicle } i,\ i \in \{1, ..., k\} \\ 0, & \text{otherwise} \end{cases}$
- t_i^z - $\begin{cases} 1, & \text{if customer } z \text{ is seated in vehicle } i,\ i \in \{1, ..., k\} \\ 0, & \text{otherwise} \end{cases}$
- $p_i^{xyz} = t_i^x.t_i^y.t_i^z + \bar{t}_i^x.t_i^y.t_i^z + t_i^x.\bar{t}_i^y.t_i^z + t_i^x.t_i^y.\bar{t}_i^z$.
- p_i^{xyz} - $\begin{cases} 1, & \text{if a combination of customers } x, y \text{ and } z \text{ are seated in vehicle } i,\ i \in \{1, ..., k\} \\ 0, & \text{otherwise} \end{cases}$
- $dist$ - $\begin{cases} d^{xyz}, & \text{distance between } x,\ y \text{ and } z \text{ if they are seated together} \\ d^{xy}, & \text{euclidean distance between } x \text{ and } y \text{ if they are seated together} \\ d^{xz}, & \text{euclidean distance between } x \text{ and } z \text{ if they are seated together} \\ d^{yz}, & \text{euclidean distance between } y \text{ and } z \text{ if they are seated together} \end{cases}$

$$\min \sum_{i=1}^{k} \sum_{x=1}^{n-2} \sum_{y=x+1}^{n-1} \sum_{z=y+1}^{n} dist.p_i^{xyz} \tag{3.22}$$

subject to,

$$\sum_{i=1}^{k} t_i^x = 1, \quad \forall x \tag{3.23}$$

$$\sum_{i=1}^{k} t_i^x \in \{2, 3\}, \quad \forall i \tag{3.24}$$

$$t_i^x, t_i^y, t_i^z, p_i^{xyz} \in \{0, 1\} \tag{3.25}$$

Constraint 3.23 specifies that each customer can only sit in one vehicle. Constraint 3.24 specifies that each vehicle should accommodate customers to its full capacity

which is either 2 or 3.

The formulation is similar to the formulation of CVRP in Section 3.3.2. The only difference here is that either two or three customers are grouped together or two. At-most any one of the variables t_i^x, t_i^y or t_i^z can be 0 and when this happens two customers share a vehicle.

When three customers are grouped together then the distance is calculated as d^{xyz}. We have applied two methods for calculating d^{xyz}:

- **Method 1: Absolute Distance**

To calculate d^{xyz} we generate a 3-dimensional matrix. Like discussed previously there are three possible orders of travel between the customers: $x \rightarrow y \rightarrow z$, $y \rightarrow x \rightarrow z$ and $x \rightarrow z \rightarrow y$. We select the travel order which generates the least travel distance. We assume that the first customer in the order is a, the second one is b and the third one is c. The matrix stores this distance if it is smaller than a constant variable $d_{absconst}$. Otherwise, it stores infinity (∞), suggesting that the three customers cannot be grouped together (Equations 3.26, 3.27 and 3.28). The variable $d_{absconst}$ is a measure of the closeness between three customers. The customers are matched with a vehicle only if the minimum travel distance between them is at-most equal to the value of $d_{absconst}$. By varying the value of $d_{absconst}$ we can define how close we want the customers to be in the ridesharing matches generated. A small value of $d_{absconst}$ will result in quick drop-offs for the customers who are seated in vehicles with maximum capacity 3. A large value of $d_{absconst}$ will group customers together who are far apart.

Let,

$$r = min \begin{cases} distance(x, y) + distance(y, z), & a = x, c = z \\ distance(y, x) + distance(x, z), & a = y, c = z \\ distance(x, z) + distance(z, y), & a = x, c = y \end{cases} \tag{3.26}$$

and,

$$depotDistance = distance(v_0, a) + distance(c, v_0) \tag{3.27}$$

then,

$$distance\_matrix[x][y][z] = min \begin{cases} r + depotDistance, & \text{if } r < d_{absconst} \\ \infty, & otherwise \end{cases} \tag{3.28}$$

For generating the solutions we set the variable $d_{absconst}$ to $0.25, 0.50$ and 0.75 and apply it on the grid cells extracted from the New York Yellow Taxi dataset. We create three datasets to train the Ptr-Net called LPAC25, LPAC50 and LPAC75.

- **Method 2: Relative Distance**

Similar to **Method 1: Absolute Constant** described above, there is another way of grouping three customers together. In this method the three-dimensional matrix stores the minimum travels distance between the first customer a, the second customer b and the third customer c only if the ratio $(r/depotDist)$ between the distance travelled between the three customers and distance travelled to and from the depot is less then a constant $d_{relconst}$. Otherwise, it stores infinity (∞), suggesting that the three customers should not be grouped together (Equations 3.29, 3.30 and 3.31). The variable $d_{relconst}$ is a measure of what part of the total distance a vehicle travels while travelling between customer drop-off location where it travels to and from the depot.

If $d_{relconst}$ is small the three customers who are far from the depot share a vehicle and if $d_{relconst}$ is large then three customers who are near the depot share a vehicle. The other case would be that if $d_{relconst}$ is small then vehicle will travel less between customers than to and from the depot and if $d_{relconst}$ is large then the vehicle will travel more between the customers than to and from the depot, hence reducing the total distance travelled. Let,

$$r = min \begin{cases} distance(x, y) + distance(y, z), & a = x, c = z \\ distance(y, x) + distance(x, z), & a = y, c = z \\ distance(x, z) + distance(z, y), & a = x, c = y \end{cases} \tag{3.29}$$

and,

$$depotDistance = distance(v_0, a) + distance(c, v_0) \tag{3.30}$$

then,

$$distance\_matrix[x][y][z] = min \begin{cases} r + depotDistance, & \text{if } r/depotDistance < d_{relconst} \\ \infty, & otherwise \end{cases} \tag{3.31}$$

For generating the solutions we set the variable $d_{relconst}$ to 0.25, 0.50 and 0.75 and apply it on the grid cells extracted from the New York Yellow Taxi dataset. We create three datasets to train the Ptr-Net called LPRC25, LPRC50 and LPRC75.

3.4 Training the Pointer Network

As discussed in Section 2.1.10.7, Ptr-Nets are used to solve combinatorial optimization problems as the output depends on the position of elements in the input. As, CVRP and HFVRP are examples of combinatorial optimization problems, we use Ptr-Net to solve them. We use the datasets generated by the different approaches discussed in Sections 3.3.1, 3.3.2 and 3.3.3 to train and test Ptr-Net. The advantage of Ptr-Net over the greedy algorithm and LP solutions (from now on we will call them **comparison algorithms**) is that Ptr-Net produces faster ridesharing solutions. The results are compared and this helps set-up expectations on what a data-driven model can achieve with respect to an exact and heuristic solution.

Using the comparison algorithms, we generate nine training datasets namely Greedy Heuristic, LPWD, LPD, LPAC25, LPAC50, LPAC75, LPRC25, LPRC50 and LPRC75. These datasets contain 20,000 records of labelled data. Each training dataset is divided into two attributes input and output. Each record in the input contains normalized coordinates of the depot and customer's drop-off locations and the output contains a *Tour* sequence as follows:

Tour: 0 - 9 - 12 - 2 - 0 - 0 - 7 - 8 - 3 - 0 - 0 - 13 - 6 - 5 - 0 - 0 - 11 - 14 - 4 - 0 - 0 - 15 - 10 - 1 - 0

Here, 0 maps to the coordinates of the depot and all the other integers maps to the vertices of customer drop-off locations. The vertices between 0s represents groups of customers assigned to a single vehicle. The Ptr-Net is trained on the input coordinates and learns the output *Tour* sequences. Ptr-Net is made up of multiple layers of unidirectional LSTM cells (Section 2.1.10.4) and is trained for 70-100 epochs.

The input data is processed before feeding into the Ptr-Net. For every record the

coordinates are normalized within a range of $[0, 1]$. Therefore, of this the location of the depot is not fixed at a certain location and varies depending on all customer drop-off locations. As suggested by Vinyals et al. [69] we sort the input data based on x-axis because through their experiments they found that models learned were not as good. This is done to reduce ambiguity during training.

As discussed before, the comparison algorithms produce ridesharing solutions along with the order of travel between customers i.e. the routing of the vehicles between customers and depot. This routing is not sorted because sorting might change the order of travel and may result in a different value of the distance travelled between the customers. The one drawback of sorting the input data before feeding into the Ptr-Net is that the routing of the vehicle within a cluster of assigned trips and the depot is lost. Thus, the solutions generated in the output sequence by the Ptr-Net only indicate which customers should be grouped together in the vehicles.

3.5 Bidirectional LSTM

The Ptr-Net used for training described in Section 3.4 is made up of layers of unidirectional LSTM cells. This architecture is similar to the one described in [69]. A unidirectional LSTM layer has LSTM cells passing hidden state information in a single direction. This layer is called Forward LSTM layer and it learns long term dependencies from the oldest (earlier in time) part of a sequence. A unidirectional LSTM's cells architecture fails to perform well on long sequences. We extend the Ptr-Net model described by adding layers of bidirectional LSTM cells. The bidirectional LSTM architecture in Figure 3.5 contains a Forward LSTM layer and a second

layer called the Backward LSTM layer. The Backward LSTM layer has LSTM cells that pass hidden state information in the opposite direction to the Forward LSTM layers cells. This is how the bidirectional layer learns long-term dependencies from the newest (most recent in time) part of the sequence. This adds additional context to the network and provides faster and better learning [59]. When the information

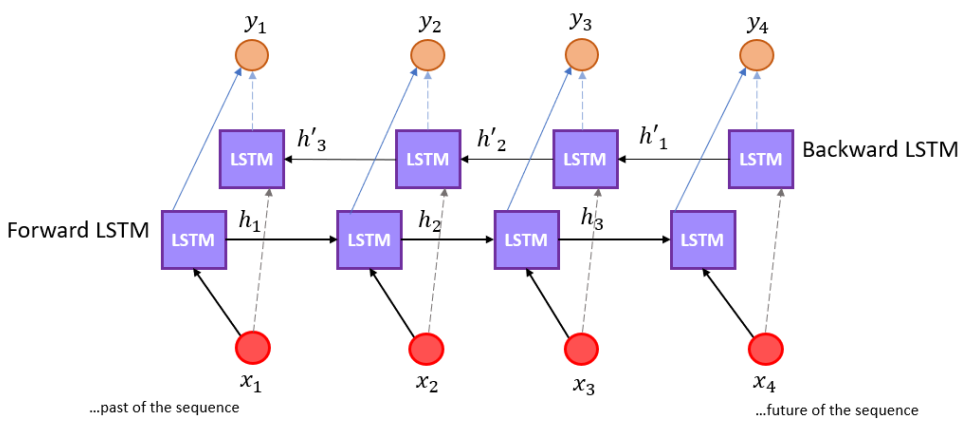

Figure 3.5: Structure of a Bidirectional LSTM network

from Forward and Backward LSTM cells are combined, the information from the beginning and the end of the sequence is preserved and the architecture performs better for longer sequences.

Chapter 4

Experimental Results and Analysis

In this chapter, we analyze the performance of the ridesharing solutions generated by the greedy algorithm and linear programming formulations. The results are compared with the predictions generated by the Pointer Network (Ptr-Net) models.

4.1 Testing Methodology

The aim of the experiment is to compare the ridesharing matches generated by the greedy, linear programming algorithms (comparison algorithms) and Ptr-Net models. We test the results on two different size of customers' requests that are provided as input. In the first case we test the results on groups of 15 customers and in the second case we test the results on groups of 30 customers'. The input to the comparison algorithms and the Ptr-Net contains a sequence of coordinates in the form of the depot location and customer drop-off locations. The output generated is

a sequence of the following form:

Tour: 0 - 9 - 12 - 2 - 0 - 0 - 7 - 8 - 3 - 0 - 0 - 13 - 6 - 5 - 0 - 0 - 11 - 14 - 4 - 0 - 0 - 15 - 10 - 1 - 0

Here, 0 maps to the coordinates of the depot and all the other integers map to the vertices of customer drop-off locations. The vertices between 0s are groups of customers matched to a single vehicle. A ridesharing solution for 15 customer requests is depicted in Figure 4.1. The solution sequence generated by comparison algorithms is called *target-matching* and the sequence generated by Ptr-Net is called *predicted-matching*.

We first generate datasets for training the Ptr-Net using the comparison algorithms.

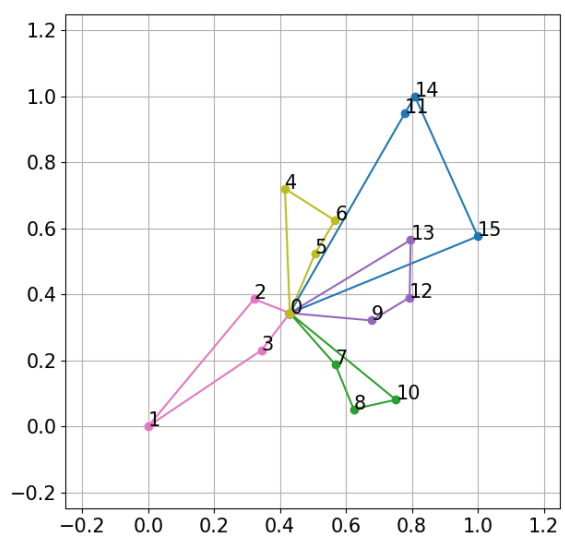

Figure 4.1: Ridesharing matches for 15 customers (LPRC75).

These datasets are named as Greedy, LPWD, LPD, LPAC25, LPAC50, LPAC75, LPRC25, LPRC50 and LPRC75. Two types of datasets are generated for each type of comparison algorithm. For the first dataset every record contains 15 customer

requests and for the second dataset every record contains 30 customers requests. A customer request is in the form of the customer drop-off location. Hence in totality there are 18 such datasets. Each dataset contains 20,000 records. We train 36 different models of Ptr-Net, 2 for each type of the 9 datasets which have different lengths of customers (15 and 30) and for each of those 2 different models of Ptr-Net one for LSTM and one for BLSTM. We compare the training quality of these models against the type of comparison system on which they are trained. For example, if the Ptr-Net-BLSTM has been trained on a LPAC75 solution for 30 customers then we compare the results of that model with LPAC75's solution for 30 customers. We compare the results generated by averaging them over 10 batches of 128 records each. Such an average gives us more accurate and reliable measurements of the results.

4.1.1 Architecture and Hyperparameters of Pointer Network

For this thesis we have not done extensive work on the hyperparameter search of Ptr-Net. We have used the same architecture throughout all the datasets. We have extended the unidirectional LSTM model of Ptr-Net with a bidirectional LSTM model (BLSTM) model and compared the results.

The unidirectional LSTM Ptr-Net (Ptr-Net-LSTM) model is made up of 5 layers each of 128 LSTM cells. It is trained on a batch size of 128 records for 60 epochs. The BLSTM Ptr-Net (Ptr-Net-BLSTM) model is made up of 10 layers each of 128 LSTM cells. It is also trained on a batch size of 128 units for 100 epochs.

4.1.2 Measurement of Results

We measure the results generated by comparison algorithms and Ptr-Net models on several metrics. All the results were averaged over 10 batches hence the metrics were averaged to get accurate results.

Since, the objective of the comparison algorithms is to minimize the total distance travelled by the entire fleet, we compare the algorithms based on the average total distance generated and their standard deviation and minimum and maximum distance. We also compare the average total distance generated by the predictions made by Ptr-Net-LSTM and Ptr-Net-BLSTM. We do this by measuring the *Optimality Gap* (OP). It helps to evaluate the performance of Ptr-Net against the comparison algorithms on minimizing the total distance travelled by the fleet. It is calculated as follows:

$$OP = \frac{|TotalDist_{comp\_alg} - TotalDist_{Ptr-Net}| * 100}{TotalDist_{comp\_alg}} \tag{4.1}$$

We also compare the average total time needed to generate one solution by each of the comparison algorithms. Since the Ptr-Net generates results in batches we compare the time taken by Ptr-Net to generate solutions for 128 records and the time taken by the comparison algorithms to generate 128 different sets of solutions. This helps measure the speed of Ptr-Net against the comparison algorithms.

4.2 Experiment Setup

We used a Dell XPS, 4.00 GHz processor, core i7-6700K CPU with 16GB RAM for the experiments. The development of comparison algorithms is done using python 3.7.4

	15 Customers		30 Customers	
Solution Type	Average Total Distance	Standard Deviation	Average Total Distance	Standard Deviation
Greedy	5.9005	0.2394	7.4077	0.3579
LPWD	**5.4834**	0.2406	6.8978	0.3360
LPD	5.4865	0.2415	**6.8951**	0.3324

Table 4.1: Average Total Distance for comparison algorithms (CVRP)

and PulP 1.6.8 as the linear programming framework. The development of Ptr-Net is done using tensorflow 1.13.0rc1 which includes both the training and inference code. Pulp is an open-source linear programming package which largely uses Python syntax and comes packaged with many industry-standard solvers. It is a high level modeling language that leverages the syntax and keywords of Python to describe mathematical programs. Tensorflow is an open source library created by Google. It is used for numerical computation and development of large-scale machine learning models. Tensorflow bundles together machine learning a neural network models. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++ environment.

4.3 Empirical Results

4.3.1 Average Total Distance

In this section we compare the average total distance covered by the entire fleet as generated by the comparison algorithms. This is presented in Table 4.1, Table 4.2 and Table 4.3. The results are presented for generating one solution for set of 15 customer requests and one solution for a set of 30 customer requests. From Table

	15 Customers		30 Customers	
Solution Type	Average Total Distance	Standard Deviation	Average Total Distance	Standard Deviation
LPAC25	6.01	0.3117	7.2670	0.3600
LPAC50	5.7053	0.2828	7.0669	0.3334
LPAC75	**5.5598**	0.2524	**6.9452**	0.3209

Table 4.2: Average Total Distance for comparison algorithms (LPAC, HFVRP)

	15 Customers		30 Customers	
Solution Type	Average Total Distance	Standard Deviation	Average Total Distance	Standard Deviation
LPRC25	6.1574	0.2963	7.7014	0.3362
LPRC50	5.7388	0.2288	7.0859	0.3202
LPRC75	**5.5244**	0.2416	**6.8840**	0.3239

Table 4.3: Average Total Distance for comparison algorithms (LPRC, HFVRP)

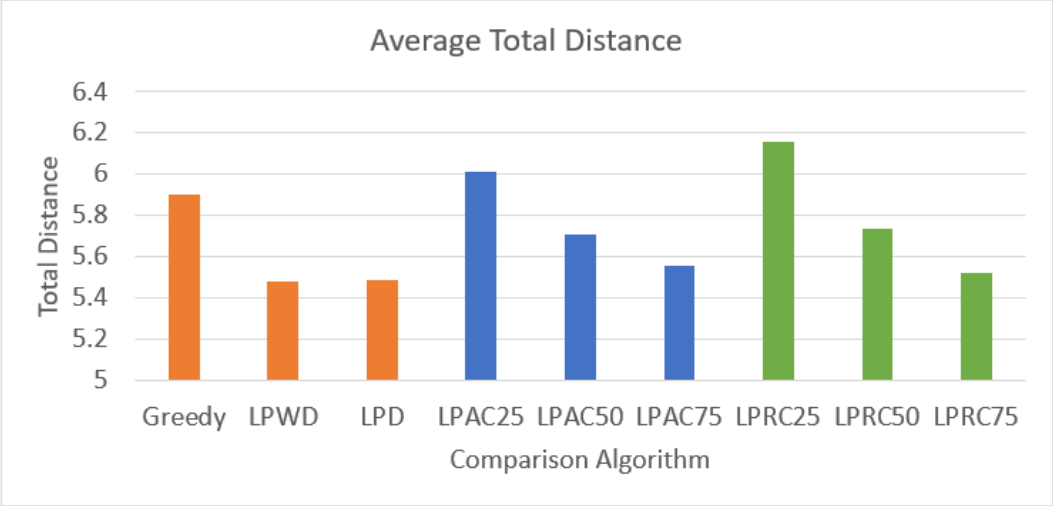

Figure 4.2: Average Total Distance for 15 customer requests. Orange indicates CVRP solution, blue indicates HFVRP solution using LPAC and green indicates HFVRP solution using LPRC

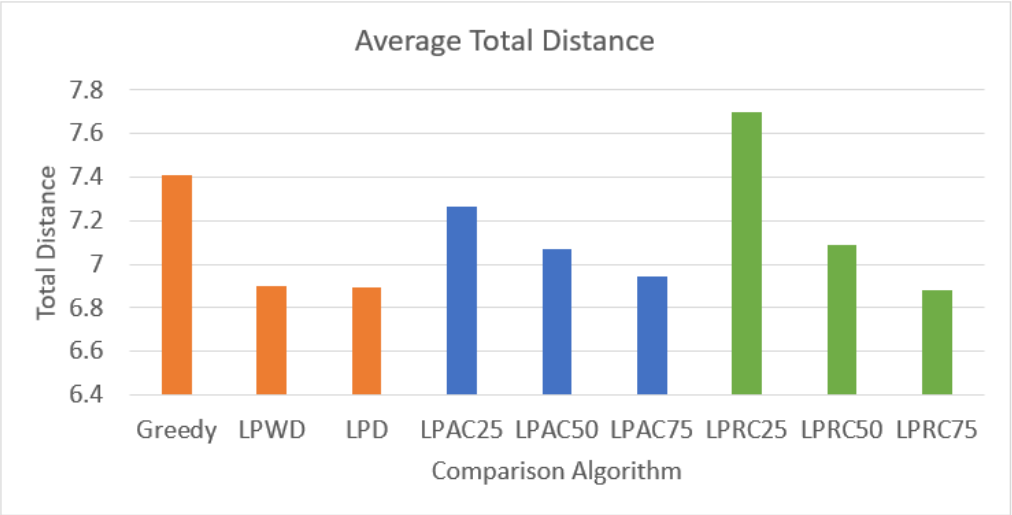

Figure 4.3: Average Total Distance for 30 customer requests. Orange indicates CVRP solution, blue indicates HFVRP solution using LPAC and green indicates HFVRP solution using LPRC

4.1, Figure 4.2 and Figure 4.3 it can be noted that LPWD and LPD generate the least average total distance for a set of 15 customers and for a set of 30 customers. From Table 4.2, Figure 4.2 and Figure 4.3 LPAC75 generates the least average total distance with a value of 5.5598 units in the case of 15 customers and 6.9452 in the case of 30 customers. Conversely, LPAC25 generates the worst average total distance with a value of 6.01 units in the case of 15 customers and 7.267 units in the case of 30 customers. From Table 4.3, Figure 4.2 and Figure 4.3, LPAC75 generates the least average total distance with a value of 5.5598 units in the case of 15 customers and 6.884 in the case of 30 customers. Conversely, LPAC25 generates the worst average total distance with a value of 6.1574 units in the case of 15 customers and 7.7014 units in the case of 30 customers. It can be seen from Figure 4.4 that LPRC25 generates a solution in which there is a group of three customers matched to a vehicle and

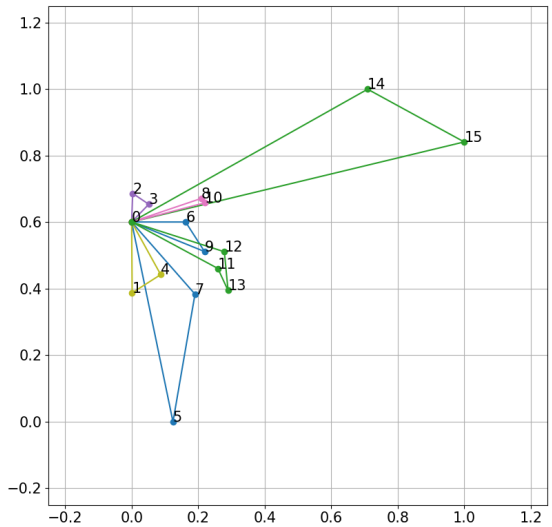

Figure 4.4: Ridesharing matches generated for 15 customers (LPRC25)

six groups of two customers matched to different vehicles. Hence, the total distance increases due to the added commutation to and from the depot for each of those groups. On the other hand, LPRC75 has just one group of two customers matched to a vehicle (Figure 4.5) hence, it saves on extra travel distance of to and fro from the depot.

On comparing Figure 4.1 to Figure 4.6 we can see that the greedy algorithm groups customers based on their proximity to the depot whereas LPRC75 groups customers based on their closeness to each other. Therefore, the average total distance by the greedy algorithm is worse than LPRC75 and other solutions.

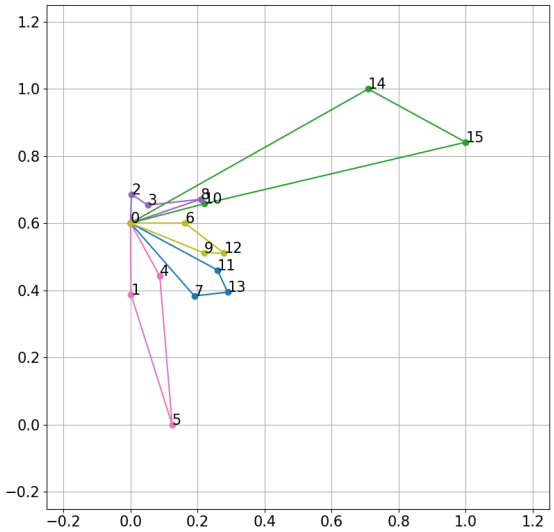

Figure 4.5: Ridesharing matches generated for 15 customers (LPRC75)

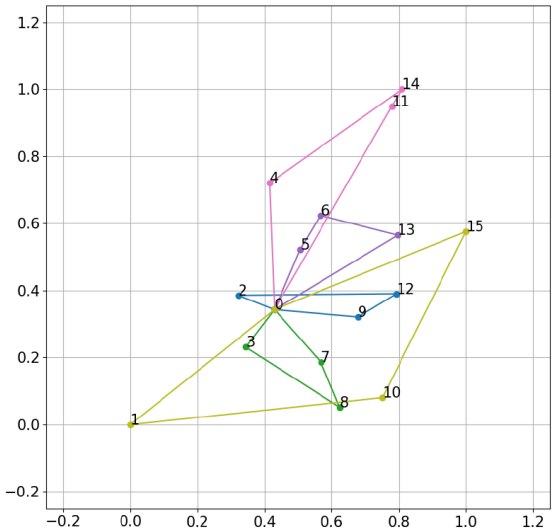

Figure 4.6: Ridesharing matches for 15 customers (Greedy)

Solution Type	Average Total Distance	Average Total Distance Ptr-Net-LSTM	Optimality Gap (%)	Average Total Distance Ptr-Net-BLSTM	Optimality Gap (%)
Greedy	5.9005	6.212	5.27	5.943	0.719
LPWD	5.4834	5.985	8.65	5.851	6.71
LPD	5.4865	6.111	11.38	5.951	8.47

Table 4.4: Optimality Gap for 15 customers (CVRP)

4.3.2 Optimality Gap

There are certain errors in the Ptr-Net models as discussed in Section 4.4. After correcting those errors we compare the average total distance between the comparison algorithms and the outputs of the Pointer Network. For the case of CVRP, it can be seen from Table 4.4 and Figure 4.7 that Ptr-Net-LSTM and Ptr-Net-BLSTM models have comparable performance in the case of 15 customers. For a sequence of 30 customers, which is a longer sequence Table 4.7 and Figure 4.8 indicate that Ptr-Net-BLSTM model performs better than the Ptr-Net-LSTM model. This is also true for HFVRP. From Table 4.5, Table 4.6 and Figure 4.7 it can be observed that Ptr-Net-LSTM and Ptr-Net-BLSTM have comparable performance in the case if 15 customers. Ptr-Net-BLSTM performs better in the case of 30 customers (Table 4.8, Table 4.9 and Figure 4.8).

Solution Type	Average Total Distance	Average Total Distance Ptr-Net-LSTM	Optimality Gap (%)	Average Total Distance Ptr-Net-BLSTM	Optimality Gap (%)
LPAC25	6.013	6.551	8.94	6.402	6.46
LPAC50	5.7053	6.301	11.47	6.145	7.71
LPAC75	5.5598	5.914	6.37	5.825	4.76

Table 4.5: Optimality Gap for 15 customers (LPAC, HFVRP)

Solution Type	Average Total Distance	Average Total Distance Ptr-Net-LSTM	Optimality Gap (%)	Average Total Distance Ptr-Net-BLSTM	Optimality Gap (%)
LPRC25	6.1574	6.644	7.9	6.487	5.35
LPRC50	5.7388	6.524	13.68	6.466	12.67
LPRC75	5.5244	6.04	9.33	5.891	6.63

Table 4.6: Optimality Gap on ridesharing for 15 customers (LPRC, HFVRP)

Solution Type	Average Total Distance	Average Total Distance PN-LSTM	Optimality Gap (%)	Average Total Distance PN-BLSTM	Optimality Gap (%)
Greedy	7.4077	10.428	40.77	8.938	20.66
LPWD	6.8978	8.512	23.4	8.1	17.42
LPD	6.8951	8.482	23.01	7.871	14.15

Table 4.7: Optimality Gap of ridesharing for 30 customers (CVRP)

Solution Type	Average Total Distance	Average Total Distance PN-LSTM	Optimality Gap (%)	Average Total Distance PN-BLSTM	Optimality Gap (%)
LPAC25	7.267	8.736	20.21	8.359	15.02
LPAC50	7.0669	9.678	36.94	8.045	13.84
LPAC75	6.9452	8.793	26.6	7.509	8.11

Table 4.8: Optimality Gap of ridesharing for 30 customers (LPAC, HFVRP)

Solution Type	Average Total Distance	Average Total Distance PN-LSTM	Optimality Gap (%)	Average Total Distance PN-BLSTM	Optimality Gap (%)
LPRC25	7.7014	10.644	38.2	8.968	16.44
LPRC50	7.0859	8.647	22.03	7.845	10.71
LPRC75	6.884	8.441	22.61	7.488	8.77

Table 4.9: Optimality Gap of ridesharing for 30 customers (LPRC, HFVRP)

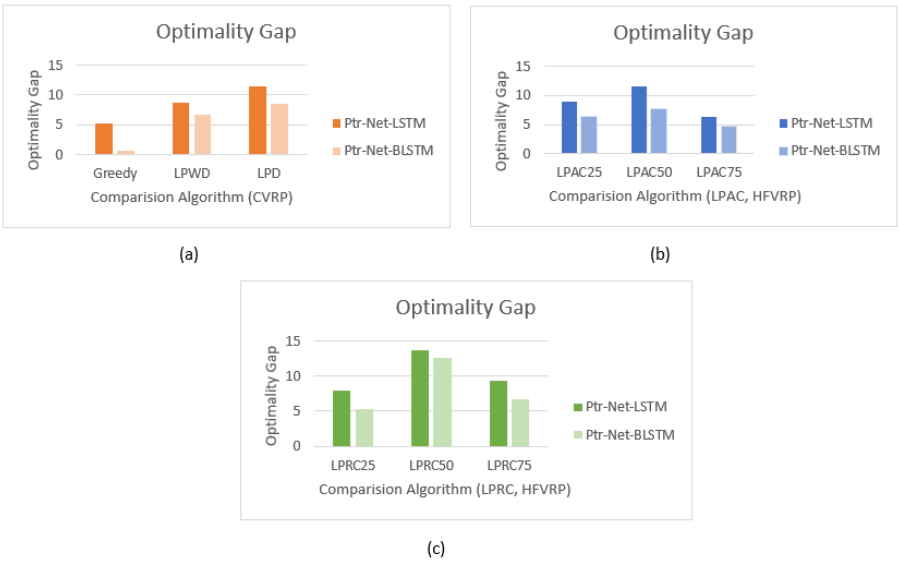

Figure 4.7: Optimality Gap of ridesharing matches for 15 customers.(a) for CVRP (b) For HFVRP using LPAC (c) for HFVRP using LPRC

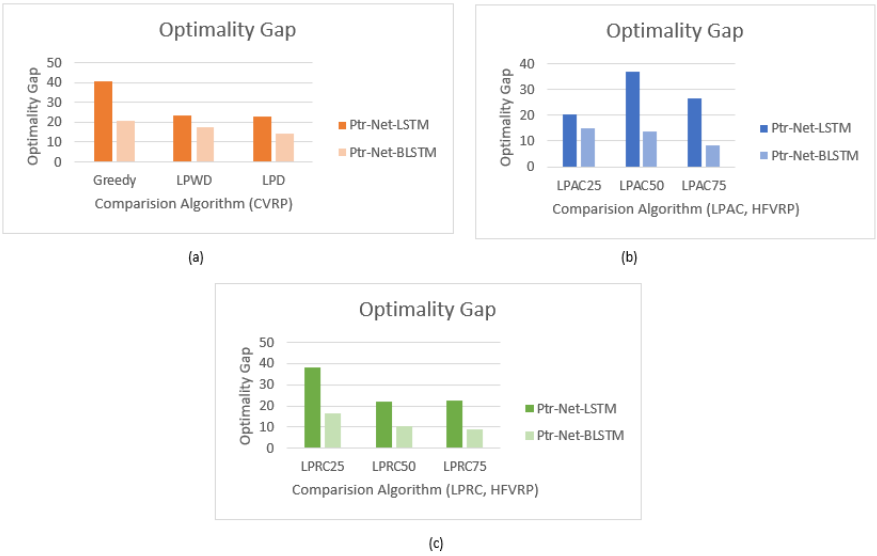

Figure 4.8: Optimality Gap of ridesharing matches for 30 customers.(a) for CVRP (b) For HFVRP using LPAC (c) for HFVRP using LPRC

4.3.3 Minimum and Maximum Distance

In this section we compare the minimum and maximum distances generated by comparison algorithms. For CVRP it can be observed from Table 4.10 that Greedy generates the most travel distance of 9.781 units for 15 customers and 12.809 units for 30 customers. LPWD generates the least travel distance of 3.493 units for 15 customers and 3.603 units for 30 customers. In the case of HFVRP out of the three LPAC solutions LPAC25 generates the maximum travel distance of 10.579 units for 15 customers and 12.604 units for 30 customers. On the other hand, LPAC75 and LPAC50 generate the least minimum travel distance of 3.741 units for 15 customers and 3.608 units for 30 customers. Out of the three LPRC solutions LPRC25 generates the most maximum travel distance of 10.173 units for 15 customers and 12.985 units

	15 Customers		30 Customers	
Solution Type	Minimum Distance	Maximum Distance	Minimum Distance	Maximum Distance
Greedy	3.606	**9.781**	3.628	**12.809**
LPWD	**3.493**	8.392	**3.603**	11.905
LPD	3.552	8.392	3.603	11.905

Table 4.10: Minimum and Maximum distance generated for CVRP

	15 Customers		30 Customers	
Solution Type	Minimum Distance	Maximum Distance	Minimum Distance	Maximum Distance
LPAC25	3.745	**10.579**	3.71	**12.604**
LPAC50	**3.741**	9.202	**3.608**	11.702
LPAC75	**3.741**	8.889	**3.608**	11.702

Table 4.11: Minimum and Maximum distance generated by LPAC for HFVRP

for 30 customers. LPRC75 on the other hand generates the least minimum travel distance of 3.601 units for 15 customers and 3.605 units for 30 customers.

4.3.4 Average Total Time

In this section, we compare the average total time taken by the comparison algorithms to generate ridesharing matches for a set of 15 and 30 customers at a time.

	15 Customers		30 Customers	
Solution Type	Minimum Distance	Maximum Distance	Minimum Distance	Maximum Distance
LPRC25	3.645	**10.173**	3.97	**12.985**
LPRC50	3.645	8.989	3.608	11.893
LPRC75	**3.601**	8.889	**3.605**	11.702

Table 4.12: Minimum and Maximum distance generated by LPRC for HFVRP

	15 Customers (one set)		30 Customers (one set)	
Solution Type	Average Total Time (secs)	Standard Deviation	Average Total Time (secs)	Standard Deviation
Greedy	**0.0298**	0.0018	**0.441**	0.0115
LPWD	0.5364	0.0537	12.7355	0.4028
LPD	0.5309	0.0598	12.8773	0.4455

Table 4.13: Average Total Time for solving CVRP

	15 Customers (one set)		30 Customers (one set)	
Solution Type	Average Total Time (secs)	Standard Deviation	Average Total Time (secs)	Standard Deviation
LPAC25	**0.4413**	0.1031	**14.164**	1.0236
LPAC50	0.4621	0.1241	17.1796	1.2535
LPAC75	**0.4693**	0.1188	**20.701**	2.5334

Table 4.14: Average Total Time for solving HFVRP using LPAC solutions

For CVRP, it can be observed from Table 4.13 that irrespective of the number of customers, greedy heuristic produces the fastest solution. It generates a solution for a set of 15 customers in 0.0298 secs on an average and a solution for a set of 30 customers in 0.441 seconds on an average. For HFVRP, it can be observed from Table 4.14 that although LPAC25 generates one of the worst solutions it consumes the least amount of time out of the three LPAC solutions. For 15 customers it takes 0.441 seconds and for 30 customers it takes 14.164 seconds. On the other hand, although LPAC75 generates one of the best solutions, it takes the most amount of time. It takes 0.4693 seconds for 15 customers and 20.701 for 30 customers. The same behavior can be observed in LPRC solutions. LPRC25 takes 0.3973 seconds generate a solution for 15 customers and 16.983 seconds for 30 customers. Whereas, LPRC75 takes 0.4799 seconds to generate a solution for 15 customers and 25.1808 seconds for 30 customers.

	15 Customers (one set)		30 Customers (one set)	
Solution Type	Average Total Time (secs)	Standard Deviation	Average Total Time (secs)	Standard Deviation
LPRC25	**0.3973**	0.0875	**16.9834**	2.6756
LPRC50	0.415	0.0912	21.3999	1.5675
LPRC75	**0.4799**	0.1044	**25.1808**	2.8041

Table 4.15: Average Total Time for HFVRP using LPRC solutions

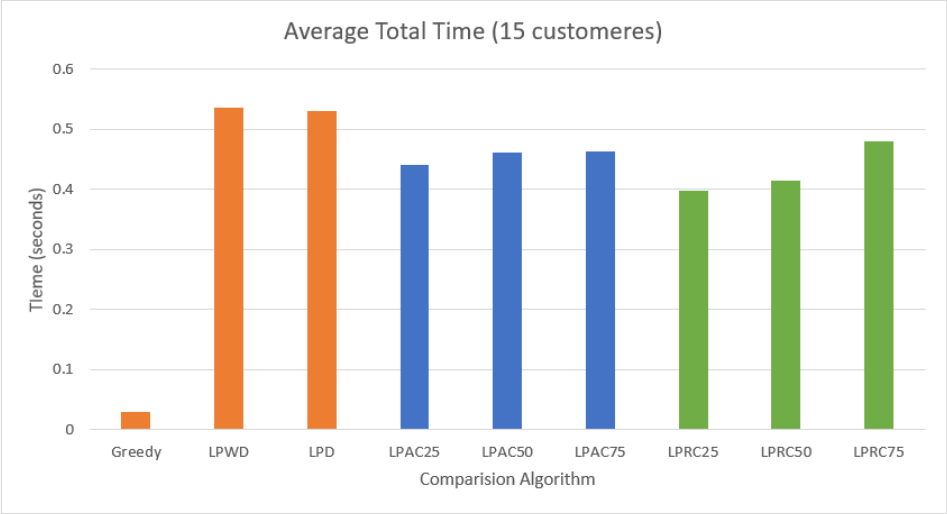

Figure 4.9: Average Total Time per 15 customers. Orange indicates CVRP solution, blue indicates HFVRP solution using LPAC and green indicates HFVRP solution using LPRC

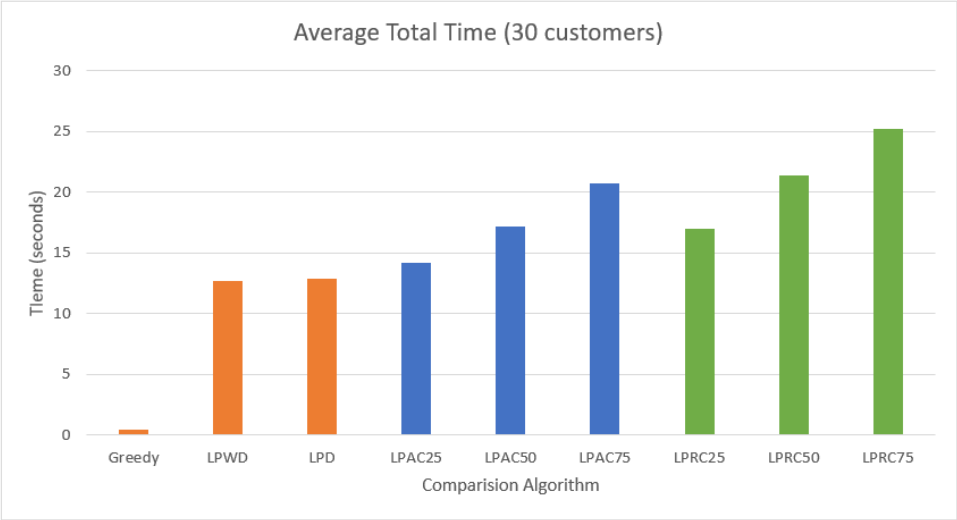

Figure 4.10: Average Total Time per 30 customers. Orange indicates CVRP solution, blue indicates HFVRP solution using LPAC and green indicates HFVRP solution using LPRC

After fixing the error as discussed in Section 4.4, we compare the average time needed to generate 128 sets of final solutions by the Ptr-Net models. Since, the Ptr-Net models are run two times, once to generate the initial solutions and then second time to generate matching for the excluded vertices. The time taken by the models will be doubled. Ptr-Net models have significant speed-ups as compared to the comparison algorithms. For CVRP, Ptr-Net LSTM is 4.61 times faster than the Greedy solution and around 78 times faster than LPWD and LPD for the case of 15 customers and Ptr-Net-BLST is 2 times faster than the Greedy solution and around 36 times faster than LPWD and LPD. For the case of 30 customers Ptr-Net-LSTM is 42 times faster and Pte-Net-BLSTM is 20.18 times faster than the greedy heuristic. For HFVRP, in

Solution Type	Algorithm (secs)	Ptr-Net-LSTM (secs)	Speed-up	Ptr-Net-BLSTM (secs)	Speed-up
Greedy	3.8156	0.827	**4.61**	1.9087	**1.99**
LPWD	68.659	0.8878	77.33	1.9006	36.12
LPD	67.9603	0.8637	78.73	1.9123	35.53

Table 4.16: Time Comparison for 128 sets of 15 customers (CVRP)

Solution Type	Algorithm (secs)	Ptr-Net-LSTM (secs)	Speed-up	Ptr-Net-BLSTM (secs)	Speed-up
LPAC25	56.4889	0.8637	65.39	1.7454	32.19
LPAC50	59.1564	0.851	69.51	1.6961	34.87
LPAC75	59.3024	0.851	69.68	1.6973	34.93

Table 4.17: Time Comparison for 128 sets of 15 customers (LPAC, HFVRP)

the case of 15 customers Ptr-Net-LSTM is more than 1000 times faster than the LP solutions and Ptr-Net-BLSTM in more than 500 times faster. In the case of 30 customers Ptr-Net-LSTM is more than 2000 times faster and Ptr-Net-BLSTM is more than 1000 times faster. It is interesting to note that the BLSTM model takes twice the amount of time as compared to the LSTM model because of the added backward LSTM layer.

Solution Type	Algorithm (secs)	Ptr-Net-LSTM (secs)	Speed-up	Ptr-Net-BLSTM (secs)	Speed-up
LPRC25	50.8633	0.907	56.1	1.7329	29.35
LPRC50	53.1212	0.9096	58.39	1.7206	30.87
LPRC75	61.4361	0.8926	68.82	1.6637	36.92

Table 4.18: Time Comparison for 128 sets of 15 customers (LPRC, HFVRP)

Solution Type	Algorithm (secs)	Ptr-Net-LSTM (secs)	Speed-up	Ptr-Net-BLSTM (secs)	Speed-up
Greedy	56.454	1.3316	**42.39**	2.7958	**20.18**
LPWD	1630.144	1.3512	1206.37	2.7504	592.67
LPD	1648.294	1.326	1243.02	2.7674	595.599

Table 4.19: Time Comparison for 128 sets of 30 customers (CVRP)

Solution Type	Algorithm (secs)	Ptr-Net-LSTM (secs)	Speed-up	Ptr-Net-BLSTM (secs)	Speed-up
LPAC25	1812.992	1.3426	1350.27	2.8856	628.27
LPAC50	2198.988	1.3135	1674.14	2.2473	978.47
LPAC75	2649.728	1.3014	2036.02	2.2129	1197.39

Table 4.20: Time Comparison for 128 sets of 30 customers (LPAC, HFVRP)

Solution Type	Algorithm (secs)	Ptr-Net-LSTM (secs)	Speed-up	Ptr-Net-BLSTM (secs)	Speed-up
LPAC75	2649.728	1.3014	2036.02	2.2129	1197.39
LPRC25	2173.875	1.4494	1499.76	2.2543	964.31
LPRC50	2739.187	1.4034	1951.71	2.4397	1122.75
LPRC75	3233.142	1.3522	2383.59	2.2808	1413.11

Table 4.21: Time Comparison for 128 sets of 30 customers (LPRC, HFVRP).

4.4 Issues with Pointer Network

In this section we discuss several issues in the ridesharing matches generated by Ptr-Net models. Such as:

- Mismatch of groups
- Groups with repeated vertices
- Repeated vertices in the entire sequence
- Excluded vertices

We will discuss these issues in detail in the following sections.

The following is an example of a ridesharing solution generated by LPAC25 algorithm (Figure 4.11) this is also known as *target-matching*.

Target-matching:

Tour: 0 - 1 - 5 - 0 - 0 - 2 - 4 - 0 - 0 - 3 - 6 - 7 - 0 - 0 - 8 - 9 - 0 - 0 - 10 - 13 - 11 - 12 - 0 - 0 - 14 - 15 - 0.

The Ptr-Net is fed with the same coordinates of depot and customer drop-off locations and generates the following ridesharing solution (Figure 4.12). This solution is known as *predicted-matching*.

Predicted-matching:

Tour: 0 - 1 - 5 - 7 - 0 - 0 - 2 - 3 - 4 - 0 - 0 - 5 - 7 - 0 - 0 - 9 - 9 - 10 - 0 - 0 - 11 - 13 - 13 - 0 - 0 - 12 - 14 - 15 - 0.

It is interesting to note that in the solution generated there are a few issues. Firstly, the *predicted-matching* is not similar to the *target-matching*. The groups of customers

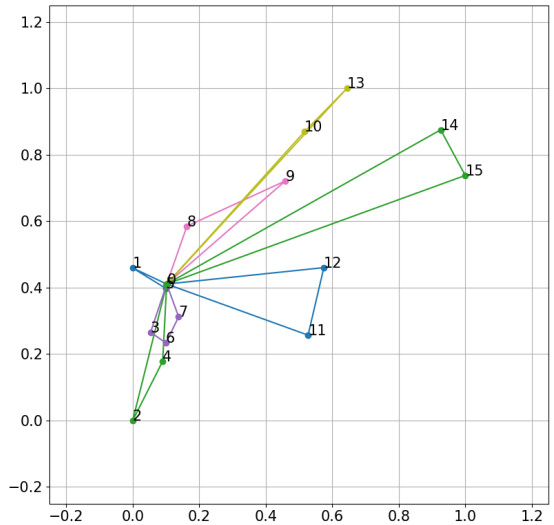

Figure 4.11: Target-matching for 15 customers (LPAC25).

matched to vehicles are not the same. Some of the vertices are repeated throughout the entire sequence. Vertices 5 and 7 are repeated in two groups. Some vertices are also repeated twice within the same group. For example, group (9 - 9 - 10) and group (11 - 13 - 13). Some of the vertices are also missing from the sequence. Here, vertices 6 and 8 are not included in the solution.

4.4.1 Mismatch of groups

Given the *target-matching* and the *predicted-matching*:

Target-matching:

Tour: 0 - 1 - 2 - 4 - 0 - 0 - 3 - 5 - 6 - 0 - 0 - 7 - 8 - 11 - 0 - 0 - 9 - 10 - 0 - 0 - 12 - 13 - 0 - 0 - 14 - 15 - 0.

Predicted-matching:

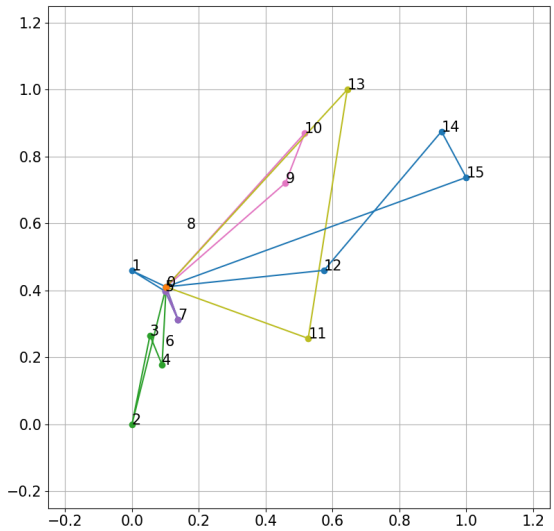

Figure 4.12: Predicted-matching for 15 customers (Ptr-Net-LSTM).

Tour: 0 - 1 - 2 - 3 - 0 - 0 - 2 - 5 - 6 - 0 - 0 - 7 - 12 - 11 - 0 - 0 - 9 - 11 - 0 - 0 - 8 - 13 - 0 - 0 - 14 - 15 - 0.

It is observed that the groups in the sequence do not match. Ptr-Net models inaccurately groups vertices which are very closely clustered. This behavior is recorded for both Ptr-Net-LSTM and Ptr-Net-BLSTM models. The mismatch results in the change of the total distance travelled by the fleet of vehicle while servicing the customers. Table 4.22 shows the average number of mismatch of groups per solution for a set of 15 customers. On an average, there are around 3.9 groups that are mismatched between the solutions generated by Ptr-Net-LSTM and the Greedy algorithm and 3.8 groups that are mismatched between the solutions generated by Ptr-Net-BLSTM and Greedy algorithm. Ptr-Net-LSTM performs the worst for LPRC50 where on an average it mismatches 4.4 groups per sequence. Ptr-Net-BLSTM performs slightly better than Ptr-Net-LSTM. Table 4.23 shows the average mismatch of groups per solution

Solution Type	Average mismatch (Ptr-Net-LSTM)	Standard Deviation	Average mismatch (Ptr-Net-BLSTM)	Standard Deviation
Greedy	3.9	1.0036	3.8	1.1184
LPWD	3.6	1.118	3.3	1.154
LPD	3.5	1.2421	3.3	1.2198
LPAC25	4.2	1.4387	4.0	1.3415
LPAC50	4.2	1.0394	3.9	1.3342
LPAC75	4.0	1.3627	3.7	1.2294
LPRC25	4.1	1.5563	4.1	1.7127
LPRC50	4.4	1.1284	4.0	1.3808
LPRC75	3.9	1.3375	3.8	1.3649

Table 4.22: Average mismatch and difference in distance for 15 customers.

Solution Type	Average mismatch (Ptr-Net-LSTM)	Standard Deviation	Average mismatch (Ptr-Net-BLSTM)	Standard Deviation
Greedy	9.4	0.7384	8.4	1.0988
LPWD	7.3	1.6061	6.9	1.6767
LPD	7.4	1.405	7.0	1.3935
LPAC25	8.2	1.7674	8.1	1.6287
LPAC50	9.9	1.0342	8.1	1.3866
LPAC75	8.1	1.4796	8.1	1.4834
LPRC25	11.5	1.322	8.8	1.8768
LPRC50	8.6	1.6183	8.4	1.7564
LPRC75	8.1	1.7113	8.1	1.5715

Table 4.23: Average mismatch and difference in distance for 30 customers.

for a set of 30 customers. Ptr-Net-LSTM performs the worst for LPRC25, where there is an average mismatch of 11.5 groups per solution generated. Ptr-Net-BLSTM performs better than Ptr-Net-LSTM as the average number of mismatch of groups is lower for Ptr-Net-BLSTM.

4.4.2 Groups with repeated vertices

In this section we will discuss one another error that is observed in the *predicted-matching* generated by Ptr-Net models. Given the following *predicted-matching*:
0 - 1 - 4 - 0 - 0 - 3 - 7 - 7 - 0 - 0 - 5 - 6 - 8 - 0 - 0 - 9 - 9 - 11 - 0 - 0 - 12 - 15 - 0 - 0 - 13 - 14 - 0
It can be observed that the groups (3,7,7) and (9,9,11) have vertices that are repeated within the same group. This repetition of vertices helps reduce the total distance travelled by the fleet. This happens when the vertices are closely clustered together and there are not enough vertices to form a group within the nearby region of non-repeating vertex. Although the vertices are repeated, the group still forms a valid matching. If we omit duplicate vertices then groups (3,7) and (9,11) can be matched to two separate vehicles. One of the consequences of repeated vertices within a group is that it results in exclusion of some vertices from the output. The resolution for this is discussed later. Table 4.24 shows the average number of repeated vertices within a group per sequence of output generated for 15 customers. From the results we can conclude that this occurs once in every 3 or 4 output sequences generated. Table 4.25 shows this for a group of 30 customers. There is at most one such group in output sequence that is generated. From the results we can conclude that Ptr-Net-BLSTM performs better than Ptr-Net-LSTM.

Solution Type	Ptr-Net-LSTM (15 customers)	Standard Deviation	Ptr-Net-BLSTM (15 customers)	Standard Deviation
Greedy	0.24	0.488	0.16	0.4226
LPWD	0.23	0.4634	0.23	0.4945
LPD	0.36	0.6341	0.23	0.3555
LPAC25	0.26	0.4918	0.15	0.3839
LPAC50	0.29	0.4865	0.27	0.4677
LPAC75	0.32	0.5293	0.31	0.5412
LPRC25	0.21	0.4446	0.17	0.4349
LPRC50	0.28	0.5086	0.26	0.4795
LPRC75	0.37	0.559	0.35	0.5242

Table 4.24: Average number of groups with repeated vertices (15 customers)

Solution Type	Ptr-Net-LSTM (30 customers)	Standard Deviation	Ptr-Net-BLSTM (30 customers)	Standard Deviation
Greedy	0.78	0.8421	0.77	0.8337
LPWD	0.95	0.9761	0.76	0.8217
LPD	0.72	0.7854	0.62	0.7917
LPAC25	0.87	0.9518	0.69	0.7083
LPAC50	1.04	1.1774	1.11	1.0363
LPAC75	1.43	1.0133	1.05	0.9581
LPRC25	0.50	0.7754	0.55	0.7284
LPRC50	1.01	0.9639	0.78	0.8094
LPRC75	0.83	0.8171	0.75	0.9141

Table 4.25: Average number of groups with repeated vertices (30 customers)

Solution Type	Ptr-Net-LSTM (15 customers)	Standard Deviation	Ptr-Net-BLSTM (15 customers)	Standard Deviation
Greedy	1.74	0.9711	1.55	0.8569
LPWD	1.51	0.9667	1.43	0.8231
LPD	1.49	0.8609	1.49	0.7911
LPAC25	1.31	0.9677	1.28	0.8311
LPAC50	1.93	1.0596	1.31	0.9046
LPAC75	1.46	0.9265	1.37	0.9109
LPRC25	1.11	0.8569	1.05	0.7551
LPRC50	1.41	1.0053	1.27	0.9075
LPRC75	1.43	1.0503	1.41	0.9348

Table 4.26: Average number of repeated vertices (15 customer)

4.4.3 Repeated vertices in an entire sequence

Given the following *predicted-matching*:

Predicted-matching: 0 - 1 - 6 - 0 - 0 - 2 - 3 - 0 - 0 - 4 - 5 - 0 - 0 - 6 - 8 - 0 - 0 - 9 - 10 - 0 - 0 - 10 - 15 - 0 - 0 - 12 - 13 - 14 - 0.

This happens because the system tries to group at-least two passengers per ride. This comes at the expense of duplicating passengers while excluding other passengers out of the output sequence. (Figure 4.13). The system forces to group two customers together even if they don't fit and results into excluding some customers out of the output generated. Table 4.26 and Table 4.29 summarize the results of the average number of repeated vertices that occur in every output sequence generated. For a group of 15 customers there are at least 3 repeated points in every two output sequences generated. For a group of 30 customers there are around 4 to 5 repeated points per output sequence generated. Ptr-Net-BLSTM performs better than Ptr-Net-LSTM in both the cases.

Solution Type	Ptr-Net-LSTM (15 customers)	Standard Deviation	Ptr-Net-BLSTM (15 customers)	Standard Deviation
Greedy	5.44	1.5952	4.83	1.541
LPWD	4.08	1.2554	3.48	1.5048
LPD	4.04	1.3603	4.03	1.5932
LPAC25	4.31	1.6439	3.62	1.5892
LPAC50	4.67	1.0306	3.79	1.5902
LPAC75	6.03	1.4947	5.32	1.2715
LPRC25	3.5	1.6082	2.77	1.5549
LPRC50	4.09	1.4522	3.46	1.5256
LPRC75	4.2	1.569	4.09	1.465

Table 4.27: Average number of repeated vertices (30 customer)

4.4.4 Excluded vertices

In this section we will discuss the vertices not included in the *predicted-matching* sequences. Given the following *predicted-matching*:

Predicted-matching: 0 - 1 - 6 - 0 - 0 - 2 - 3 - 0 - 0 - 4 - 5 - 0 - 0 - 6 - 8 - 0 - 0 - 9 - 10 - 0 - 0 - 10 - 15 - 0 - 0 - 12 - 13 - 14 - 0

It can be seen that vertices 7 and 11 are not included in the output sequence. They are excluded because of the vertices which are repeated in the sequence. We can see that vertices 6 and 10 are repeated in the sequence. Table 4.28 and Table 4.29 summarizes this for 15 and 30 customers. It is observed that for 15 customers 7% to 13% of vertices are excluded per sequence. For 30 customers there are 10% to 20% of vertices excluded per output sequence. This results in the decrease in the total distance travelled by the fleet. In this case again the Ptr-Net-BLSTM model performs better than the Ptr-Net-LSTM model. We handle the excluded vertices for our final results. We re-index these vertices and feed them into the Ptr-Net models to generate a new *predicted-matching*.

Solution Type	Ptr-Net-LSTM (15 customers)	Standard Deviation	Ptr-Net-BLSTM (15 customers)	Standard Deviation
Greedy	1.81	0.9226	1.61	0.8726
LPWD	1.52	0.9997	1.46	0.875
LPD	1.58	0.9677	1.52	0.9429
LPAC25	1.32	0.8361	1.28	0.9216
LPAC50	2.015	1.0234	1.31	0.9415
LPAC75	1.45	0.9265	1.45	0.8925
LPRC25	1.12	0.8636	1.08	0.8715
LPRC50	1.43	1.0383	1.39	0.89473
LPRC75	1.46	1.0503	1.46	0.9129

Table 4.28: Average number of excluded vertices (15 customer)

Solution Type	Ptr-Net-LSTM (30 customers)	Standard Deviation	Ptr-Net-BLSTM (30 customers)	Standard Deviation
Greedy	5.98	1.7114	5.28	1.6315
LPWD	4.32	1.5416	3.69	1.5609
LPD	4.28	1.7819	4.28	1.5986
LPAC25	4.59	1.6768	3.78	1.9424
LPAC50	4.07	1.9105	4.17	1.4452
LPAC75	6.07	1.8226	5.93	1.5438
LPRC25	3.73	1.7036	2.81	1.4793
LPRC50	4.31	1.5767	3.74	1.5334
LPRC75	4.44	1.734	4.33	1.7333

Table 4.29: Average number of excluded vertices (30 customer)

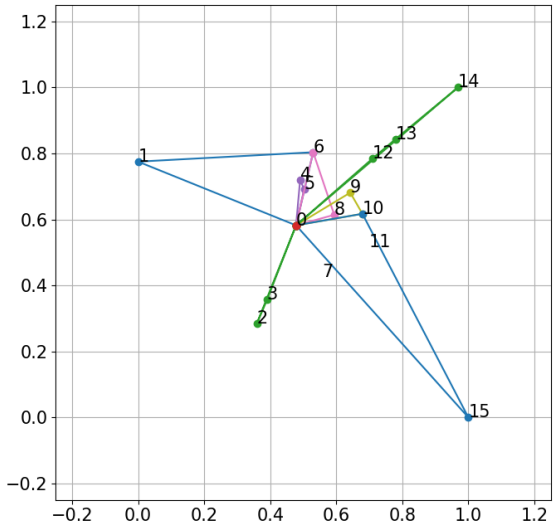

Figure 4.13: Predicted-matching for 15 customers with repeated vertices.

4.5 Serviceable Matchings

In this section we discuss the average number of serviceable matchings generated by the Ptr-Net models. Serviceable matchings are those groups in predicted matchings that do not contain repeated vertices. If vertices are duplicated then their other instances are removed and the customers are matched to vehicles based on the remaining instances in the group. If after removing repeated vertices a group is left with just a single vertex then that vertex is added to the list of excluded vertices. For example given the following *predicted-matching*:

Predicted-matching: 0 - 1 - 2 - 0 - 0 - 2 - 4 - 0 - 0 - 3 - 9 - 0 - 0 - 9 - 7 - 7 - 0 - 0 - 7 - 8 - 12 - 0 - 0 - 11 - 14 - 15 - 0

It is observed that vertices 2, 7 and 9 are repeated over the entire sequence. We remove the repeated instances of these vertices and any group that is left with a

Solution Type	Average Groups	Ptr-Net-LSTM (15 customers)	Ptr-Net-BLSTM (15 customers)	% Improvement
Greedy	5	3.88	3.94	1.5463
LPWD	5	4.03	4.07	0.9925
LPD	5	4.06	4.09	0.7389
LPAC25	6.4	5.38	5.42	0.7434
LPAC50	5.9	4.75	5.04	6.1052
LPAC75	5.5	4.53	4.59	1.3245
LPRC25	6.9	6.04	6.13	1.4901
LPRC50	6.1	5.42	5.60	3.321
LPRC75	5.5	4.85	4.95	2.0618

Table 4.30: Average number of serviceable groups (15 customers)

single vertex after removing the repeated vertices. The sequence that is left are the groups of vertices ready to be serviced by the vehicles. For the above example (1, 2), (3, 9), (7, 8, 12) and (11, 14, 15) are the groups that are ready to be immediately serviced. Tables 4.30 and Tables 4.31 compares the average number serviceable groups generated by Ptr-Net-LSTM and Ptr-Net-BLSTM models. For a set of 15 customers 77% to 87% groups are serviceable when compared to the target matching. Ptr-Net-BLSTM model performs better than the Ptr-Net-LSTM with an improvement ranging between 0.73% to 6.1% in terms of the number of serviceable groups generated. Similarly, for a set of 30 customers 73% to 93% of the groups are serviceable when compared to the target matching. Ptr-Net-BLSTM model performs better than the Ptr-Net-LSTM with an improvement ranging between 0.24% to 13.15% in terms of number of serviceable groups generated.

Solution Type	Average Groups	Ptr-Net-LSTM (15 customers)	Ptr-Net-BLSTM (15 customers)	% Improvement
Greedy	10	7.39	7.61	2.9769
LPWD	10	7.87	8.08	2.6683
LPD	10	7.78	7.81	0.3856
LPAC25	11.1	8.61	8.95	3.9488
LPAC50	10.9	8.32	8.81	5.8894
LPAC75	10.8	7.52	8.22	9.3085
LPRC25	13.5	11.02	12.47	13.1578
LPRC50	11.7	8.96	9.48	5.8035
LPRC75	9.6	8.23	8.25	0.243

Table 4.31: Average number of serviceable groups (30 customers)

4.6 Root Mean Square Percentage Error

In this section we calculate the Root Mean Square Percentage Error (RMSPE) to measure the average magnitude of error between each *target-matching* and *predicted-matching*. The RMSPE is calculated as:

$$RMSPE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\Big(\frac{total\_dist_{comp\_alg_i} - total\_dist_{Ptr-Net_i}}{total\_dist_{comp\_alg_i}}\Big)^2} * 100 \tag{4.2}$$

Here, *total_dist* is the total distance travelled for a set of 15 or 30 customers. We average the values over a batch of solution therefore for our calculations we have set n = 128. This helps us in measuring the accuracy of the two models against every *target-matching*.

In the case of CVRP, it is observed from Table 4.32, Figure 4.14 and Figure 4.15 that in the case of 15 customers Ptr-Net-LSTM and Ptr-Net-BLSTM have comparable performance. In the case of 30 customers Ptr-Net-LSTM perform worse than Ptr-Net-BLSTM model with an error on 41.58% for greedy heuristic and Ptr-Net-BLSTM

	15 Customers		30 Customers	
Solution Type	Ptr-Net-LSTM	Ptr-Net-BLSTM	Ptr-Net-LSTM	Ptr-Net-BLSTM
Greedy	17.97	17.13	**41.58**	**21.87**
LPWD	16.22	12.57	34.51	24.51
LPD	18.11	15.78	37.83	23.63

Table 4.32: Root Mean Square Percentage Error (CRP)

	15 Customers		30 Customers	
Solution Type	Ptr-Net-LSTM	Ptr-Net-BLSTM	Ptr-Net-LSTM	Ptr-Net-BLSTM
LPAC25	16.84	12.72	33.63	24.63
LPAC50	17.05	13.47	35.02	20.02
LPAC75	17.56	12.89	31.87	22.74

Table 4.33: Root Mean Square Percentage Error (HFVRP, LPAC)

generates solution that are 21.87% worse than the target solutions. For HFVRP, it can be observed from Table 4.33, Table 4.34, Figure 4.14 and Figure 4.15 it can be observed that in the case of 15 customers Ptr-Net-LSTM and Ptr-Net-BLSTM have comparable performance. In the case of 30 customers the Ptr-Net-LSTM model performs 25% to 35% worse than the target solution and Ptr-Net-BLSTM performs 17% to 25% worse than the target solution depending on the comparison system.

According to the results we can conclude that Ptr-Net models generate approximate ridesharing solutions with the huge advantage in taking computationally less time in generating the solutions for large number of ride requests.

	15 Customers		30 Customers	
Solution Type	Ptr-Net-LSTM	Ptr-Net-BLSTM	Ptr-Net-LSTM	Ptr-Net-BLSTM
LPRC25	13.59	13.74	35.46	21.8
LPRC50	18.18	14.8	30.15	20.66
LPRC75	18.31	15.33	25.69	17.69

Table 4.34: Root Mean Square Percentage Error (HFVRP, LPRC)

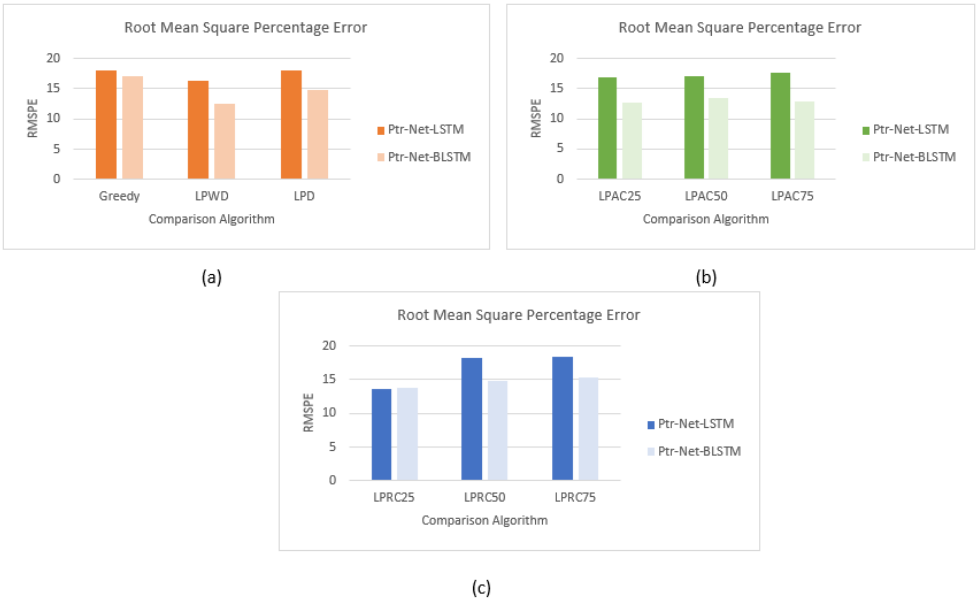

Figure 4.14: Comparison of RMSPE (15 customers). (a) for CVRP (b) For HFVRP using LPAC (c) for HFVRP using LPRC

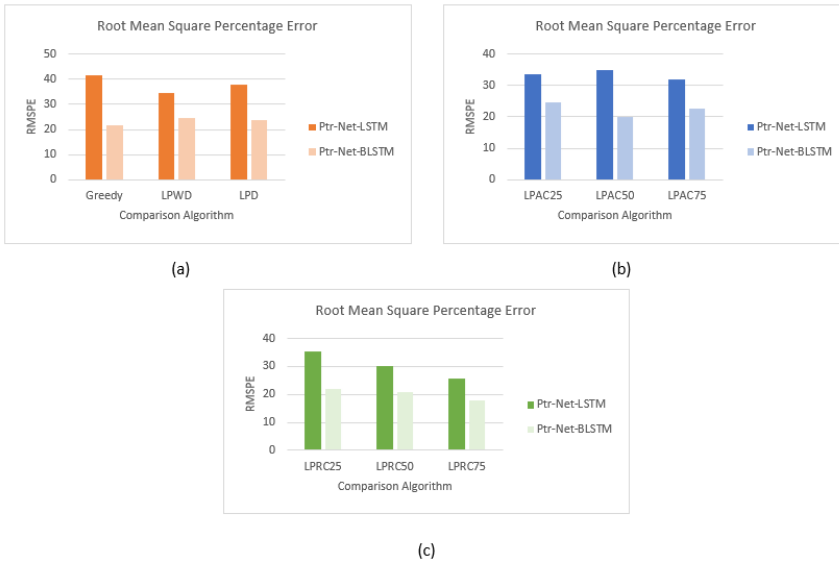

Figure 4.15: Comparison of RMSPE (30 customers). (a) for CVRP (b) For HFVRP using LPAC (c) for HFVRP using LPRC

Next, we compare the RMSPE of the comparison algorithms against each other. We use the following formula:

$$RMSPE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}\left(\frac{total\_dist_{target_i} - total\_dist_{comp\_alg_i}}{total\_dist_{target_i}}\right)^2} * 100 \tag{4.3}$$

Here n = 126 and $total\_dist_{comp\_alg}$ is the total distance of the solution whose accuracy we want to measure.

First, we compare the solutions generated by the greedy heuristic against the other LP solutions. From Table 4.35 it can be observed that greedy heuristic generates solutions which are 10% off against the LP solutions.

On comparing LPD to LPWD it is evident from Table 4.36 that LPD is off by 2%-3%

	15 Customers	30 Customers
Solution Type	Greedy	Greedy
LPWD	10.39	9.47
LPD	9.8	9.57

Table 4.35: Root Mean Square Percentage Error for Greedy solution

	15 Customers	30 Customers
Solution Type	LPD	LPD
LPWD	3.05	2.23

Table 4.36: Root Mean Square Percentage Error for LPD

against LPWD.

Next we compare the LPAC solutions against each other. From Table 4.37 it is evident that LPAC25 is perform better when compared against LPAC50 where its solutions are off by as 7.91% in the case of 15 customers and 5.37% in the case of 30 customers. When compared against LPAC75 the solutions are off by 10.38% in the case of 15 customers and 7.2% in the case of 30 customers.

On comparing LPAC50 to LPAC75 it is evident from Table 4.38 that LPAC50 solutions are off by 3%-5% against LPAC75.

	15 Customers	30 Customers
Solution Type	LPAC25	LPAC25
LPAC50	7.91	5.37
LPAC75	10.38	7.2

Table 4.37: Root Mean Square Percentage Error for LPAC25

Finally, we compare the LPRC solutions against each other. From Table 4.39 it is evident that LPRC25 performs better when compared against LPRC50 where its solutions are off by as 9.3% in the case of 15 customers and 9.19% in the case of 30 customers. When compared against LPRC75 the solutions are off by 13.84% in the case of 15 customers and 11.96% in the case of 30 customers.

	15 Customers	30 Customers
Solution Type	LPAC50	LPAC50
LPAC75	5.53	3.35

Table 4.38: Root Mean Square Percentage Error for LPAC50

	15 Customers	30 Customers
Solution Type	LPRC25	LPRC25
LPRC50	9.3	9.19
LPRC75	13.84	11.96

Table 4.39: Root Mean Square Percentage Error for LPRC25

On comparing LPRC50 to LPRC75 it is evident from Table 4.40 that LPRC50 solutions are off by 4%-5% against LPRC75.

	15 Customers	30 Customers
Solution Type	LPRC50	LPRC50
LPRC75	5.68	4.11

Table 4.40: Root Mean Square Percentage Error for LPRC50

Chapter 5

Conclusion

Ridesharing is a formulation of VRP which is an NP-Hard combinatorial optimization problem. A lot of research has been carried out on solving VRP using exact, heuristic and meta-heuristic algorithms. As the present technology is moving towards the application of data-driven and intelligent algorithms, in this thesis we have used a neural network architecture called Pointer Network (Ptr-Net) to solve the ridesharing problem.

We developed a greedy heuristic and linear programming solution (together called comparison algorithms) to solve the VRP. Using these solutions we generated datasets to train two models of Ptr-Net. The first Ptr-Net also called Ptr-Net LSTM is an adaptation of the model suggested by Vinyals et al. [69]. We improved this model by using Bidirectional Long Short Memory (BLSTM) cells and called the new model Ptr-Net-BLSTM.

We compared the average total distance generated by the comparison algorithms, where the results showed that LPD, LPWD and LPRC75 generated one of the best

solutions for the problems. The Ptr-Net-LSTM and Ptr-Net-BLTM models generated approximate solutions which initially had some errors. We discussed these errors, which were: the mismatch of groups, repeated vertices ad excluded vertices. After, dealing with errors, the final results showed that the Ptr-Net-LSTM model generated approximate solutions which were 5% - 40% worse than the target solutions. Ptr-Net-BLSTM model showed an improvement over the previous model and generated approximate solutions which were 0.72%-20% worse than the target solutions. The advantage of using Ptr-Net models can be seen in the time required to generate solutions. The results showed that for generating a batch of 128 solutions of 15 and 30 customers the Ptr-Net-LSTM model showed a speed-up by being 4 - 2000 times faster than the comparison algorithms. Similarly, the Ptr-Net-BLSTM model was 2 - 1000 times faster. Hence, Ptr-Net models can be effectively used for generating fast and approximate solutions for the ridesharing problem. Through our experiments we conclude that data-driven models can be used to learn solutions on combinatorial optimization problems.

One of the limitations of the Ptr-Net models is that they show poor results for long sequence of input. Therefore, we divided the problem into sets of 15 and 30 customers and generated results in a batch of 128 solutions at a time.

5.1 Future Work

Future work includes work on generating dynamic solutions where the ridesharing matches are generated as and when the customer requests comes in at real-time. More work can be done on exploring the effectiveness of Ptr-Net models on generating

solutions for multiple depots. More enhanced models can be created to handle time windows associated with each customer requests. More work can also be done in exploring and optimizing the hyper-parameters of the network.

Bibliography

[1] MAH Akhand, Shahina Akter, Mohd Abdur Rashid, and SB Yaakob. Velocity tentative pso: An optimal velocity implementation based particle swarm optimization to solve traveling salesman problem. *IAENG International Journal of Computer Science*, 42(3), 2015.

[2] MAH Akhand, Zahrul Jannat Peya, Tanzima Sultana, et al. Solving capacitated vehicle routing problem with route optimization using swarm intelligence. In *2015 2nd International Conference on Electrical Information and Communication Technologies (EICT)*, pages 112–117. IEEE, 2015.

[3] MAH Akhand, Pintu Chnadra Shill, Md Forhad Hossain, ABM Junaed, and K Murase. Producer-scrounger method to solve traveling salesman problem. *International Journal of Intelligent Systems and Applications*, 7(3):29, 2015.

[4] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, 114(3):462–467, 2017.

[5] Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig. Joint language and translation modeling with recurrent neural networks. 2013.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Barrie M Baker and MA Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[8] Michel L Balinski and Richard E Quandt. On an integer program for a delivery problem. *Operations research*, 12(2):300–304, 1964.

[9] John E Bell and Patrick R McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced engineering informatics*, 18(1):41–48, 2004.

[10] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

[11] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[12] Julien Bramel and David Simchi-Levi. A location based heuristic for general routing problems. *Operations research*, 43(4):649–660, 1995.

[13] Wen-Chyuan Chiang and Robert A Russell. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63(1):3–27, 1996.

[14] Eunjeong Choi and Dong-Wan Tcha. A column generation approach to the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, 34(7):2080–2095, 2007.

[15] Nicos Christofides, Aristide Mingozzi, and Paolo Toth. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, 20(1):255–282, 1981.

[16] NYC Open Data City of New York. Nyc open data. https://opendata.cityofnewyork.us/. Online; accessed 11 May 2020.

[17] Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[18] Jean-François Cordeau and Mirko Maischberger. A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033–2050, 2012.

[19] Zbigniew J Czech and Piotr Czarnas. Parallel simulated annealing for the vehicle routing problem with time windows. In *Proceedings 10th Euromicro workshop on parallel, distributed and network-based processing*, pages 376–383. IEEE, 2002.

[20] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[21] Alexandre De Brébisson, Étienne Simon, Alex Auvolat, Pascal Vincent, and Yoshua Bengio. Artificial neural networks applied to taxi destination prediction. *arXiv preprint arXiv:1508.00021*, 2015.

[22] Marco Dorigo and Thomas Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In *Handbook of metaheuristics*, pages 250–285. Springer, 2003.

[23] Matteo Fischetti, Paolo Toth, and Daniele Vigo. A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*, 42(5):846–859, 1994.

[24] Marshall L Fisher. Optimal solution of vehicle routing problems using minimum k-trees. *Operations research*, 42(4):626–642, 1994.

[25] Marshall L Fisher and Ramchandran Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

[26] Wisconsin Institute for Discovery. Integer linear programming, 1999.

[27] Yong Gao, Dan Jiang, and Yan Xu. Optimize taxi driving strategies based on reinforcement learning. *International Journal of Geographical Information Science*, 32(8):1677–1696, 2018.

[28] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management science*, 40(10):1276–1290, 1994.

[29] Billy E Gillett and Leland R Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations research*, 22(2):340–349, 1974.

[30] Fred Glover and Manuel Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

[31] Bruce Golden, Arjang Assad, Larry Levy, and Filip Gheysens. The fleet size and mix vehicle routing problem. *Computers & Operations Research*, 11(1):49–66, 1984.

[32] Bruce L Golden. Vehicle routing problems: formulations and heuristic solution techniques. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE OPERATIONS RESEARCH CENTER, 1975.

[33] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[34] Matej Grajciar. Space-time demand adaptive taxi routing. 2015.

[35] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[36] Miyoung Han, Pierre Senellart, Stéphane Bressan, and Huayu Wu. Routing an autonomous taxi with reinforcement learning. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 2421–2424, 2016.

[37] Franklin T Hanshar and Beatrice M Ombuki-Berman. Dynamic vehicle routing using genetic algorithms. *Applied Intelligence*, 27(1):89–99, 2007.

[38] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[39] Ishan Jindal, Zhiwei Tony Qin, Xuewen Chen, Matthew Nokleby, and Jieping Ye. Optimizing taxi carpool policies via reinforcement learning and spatio-temporal mining. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1417–1426. IEEE, 2018.

[40] Gilbert Laporte, Michel Gendreau, J-Y Potvin, and Frédéric Semet. Classical and modern heuristics for the vehicle routing problem. *International transactions in operational research*, 7(4-5):285–300, 2000.

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

[42] Young Hoon Lee, JI Kim, KH Kang, and KH Kim. A heuristic for vehicle fleet mix problem using tabu search and set partitioning. *Journal of the Operational Research Society*, 59(6):833–841, 2008.

[43] Claude Lemaréchal. Cauchy and the gradient method. *Doc Math Extra*, 251:254, 2012.

[44] Jan Karel Lenstra and AHG Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[45] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the*

24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1774–1783, 2018.

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

[47] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[48] Staff M. Macrumors. https://waymo.com. Online; accessed 11 May 2020.

[49] Shuo Ma, Yu Zheng, and Ouri Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 410–421. IEEE, 2013.

[50] Xiaolei Ma, Zhimin Tao, Yinhai Wang, Haiyang Yu, and Yunpeng Wang. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54:187–197, 2015.

[51] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[52] Roberto Montemanni, Luca Maria Gambardella, Andrea Emilio Rizzoli, and Alberto V Donati. Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10(4):327–343, 2005.

[53] Beatrice Ombuki, Brian J Ross, and Franklin Hanshar. Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24(1):17–30, 2006.

[54] Masayo Ota, Huy Vo, Claudio Silva, and Juliana Freire. Stars: Simulating taxi ride sharing at scale. *IEEE Transactions on Big Data*, 3(3):349–361, 2016.

[55] Ted Ralphs. Vrp datasets. https://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/index.htm.old. Online; accessed 11 May 2020.

[56] Ted K Ralphs, Leonid Kopman, William R Pulleyblank, and Leslie E Trotter. On the capacitated vehicle routing problem. *Mathematical programming*, 94(2-3):343–359, 2003.

[57] S Ropke, JF Cordeau, M Iori, and D Vigo. Branch-and-cut-and-price for the capacitated vehicle routing problem with two-dimensional loading constraints. *Proceedings of ROUTE, Jekyll Island*, 13, 2007.

[58] Paolo Santi, Giovanni Resta, Michael Szell, Stanislav Sobolevsky, Steven H Strogatz, and Carlo Ratti. Quantifying the benefits of vehicle pooling with shareability networks. *Proceedings of the National Academy of Sciences*, 111(37):13290–13294, 2014.

[59] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[60] Kwangcheol Shin and Sangyong Han. A centroid-based heuristic algorithm for the capacitated vehicle routing problem. *Computing and Informatics*, 30(4):721–732, 2012.

[61] Solomon. Solomon vrp-rep. http://www.vrp-rep.org/datasets/item/2014-0013.html. Online; accessed 11 May 2020.

[62] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

[63] I Sutskever, O Vinyals, and QV Le. Sequence to sequence learning with neural networks. *Advances in NIPS*, 2014.

[64] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[65] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432, 2015.

[66] Tesla. Tesla. https://www.tesla.com/autopilot?utm=. Online; accessed 11 May 2020.

[67] Paul M Thompson and Harilaos N Psaraftis. Cyclic transfer algorithm for multivehicle routing and scheduling problems. *Operations research*, 41(5):935–946, 1993.

[68] Subhashini Venugopalan, Marcus Rohrbach, Jeffrey Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko. Sequence to sequence-video to text. In *Proceedings of the IEEE international conference on computer vision*, pages 4534–4542, 2015.

[69] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[70] Kang-Ping Wang, Lan Huang, Chun-Guang Zhou, and Wei Pang. Particle swarm optimization for traveling salesman problem. In *Proceedings of the 2003 international conference on machine learning and cybernetics (IEEE cat. no. 03ex693)*, volume 3, pages 1583–1585. IEEE, 2003.

[71] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[72] Wayom. Wayom. https://waymo.com. Online; accessed 11 May 2020.

[73] Jun Xu, Rouhollah Rahmatizadeh, Ladislau Bölöni, and Damla Turgut. Real-time prediction of taxi demand using recurrent neural networks. *IEEE Transactions on Intelligent Transportation Systems*, 19(8):2572–2581, 2017.