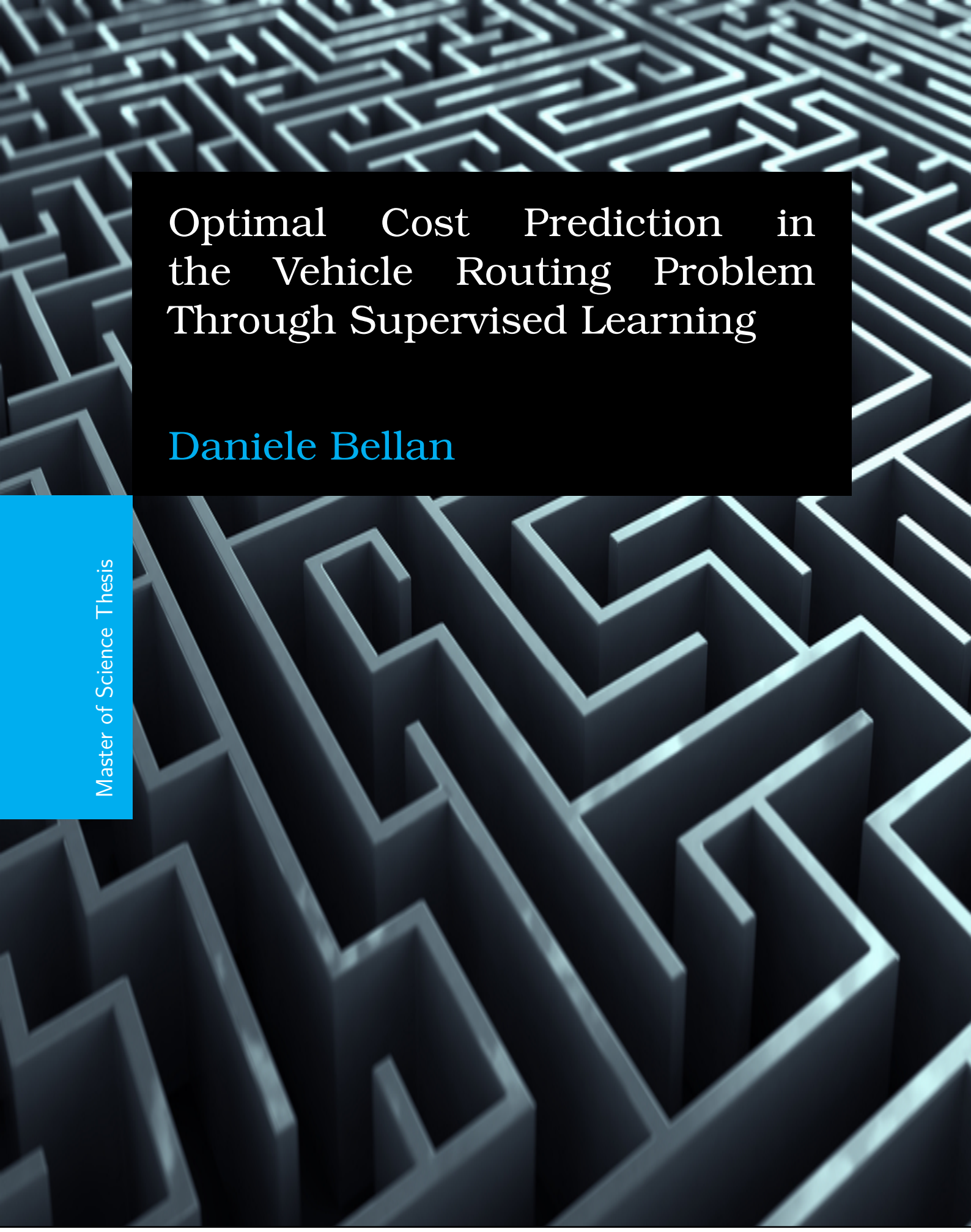


Optimal Cost Prediction in the Vehicle Routing Problem Through Supervised Learning

Daniele Bellan

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Optimal Cost Prediction in the Vehicle Routing Problem Through Supervised Learning

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Daniele Bellan

June 29, 2018

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Coverpage image taken from the book: Finch, J. (2017). *Supporting struggling students on placement: A practical guide.* Bristol, UK; Chicago, IL, USA: Policy Press at the University of Bristol. Retrieved from http://www.jstor.org/stable/j.ctt1t8960m

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Optimal Cost Prediction in the Vehicle Routing Problem Through Supervised Learning

by

Daniele Bellan

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: June 29, 2018

Supervisor(s):

prof.dr.ir. Javier Alonso-Mora

Dr.ir. Michal Čáp

Reader(s):

prof.dr.ir. M. Wisse

prof.dr.ir. Javier Alonso-Mora

Dr.ir. Michal Čáp

Abstract

Machine learning techniques aim to train a model in such a way that it can approximate complex dynamics like the vehicle routing problem. In the recent years, combinatorial neural networks and deep learning methods have been used to predict the solution of routing problems. However, the approaches investigated so far in literature could be cumbersome to apply and replicate. Although such methods obtained good results in predicting the solutions of simple routing problems, their structures are complex and they do not consider any sort of precedence constraints, an aspect that is crucial in passenger transportation. The goal of this thesis is to apply supervised learning to predict the optimal cost of single-vehicle pick-up and delivery problem, leading to a simpler implementation compared with combinatorial neural networks. First, the most suitable machine learning model able to approximate problem is chosen as a neural network with one hidden layer and a ReLu activation function. Then, the input features that improve the prediction accuracy are searched. In particular, very good results are observed by feeding the solutions of heuristic algorithm as input to the neural network. Compared to baseline prediction method which returns the length of the route computed by greedy heuristic, an improvement of 40% in prediction accuracy is obtained with the proposed approach. Finally, the model is improved to achieve better generalization properties with respect to a higher number of requests, by using the average optimal length as an additional input feature.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor prof.dr.ir. Javier Alonso-Mora for his assistance during the writing of this thesis and the chance he gave me to face such challenging and interesting problem, supporting me in exploring topics I appreciate. I would also thank my daily supervisor Dr.ir. Michal Čáp for his full support, his patience and the help given during these laborious months; he was fundamental in steering me at the right moments.

I want to thank my family, that I have always felt close during these two years. Their trust and support were fundamental during my growth.

I would like to thank my housemates, Marco and Ilario, which became my family in The Netherlands, always ready to share all the moments and ford all the rivers.

I would like to thank my friends spread around the world, which I feel we have never been apart.

Finally, I would like to thank Sara, for whom I feel the greatest gratitude. Without her support, her esteem, and her endless, unconditional faith in me, this challenging two-years experience from which I learned so much, wouldn't have been the same.

Delft, University of Technology — Daniele Bellan
June 29, 2018

"The maze wasn't meant for you"

— *Westworld - Dolores Abernathy*

"Considerate la vostra semenza:
fatti non foste a viver come bruti,
ma per seguir virtute e canoscenza"

"Consider well the seed that gave you birth:
your were not born to live as brutes,
but to follow virtue and knowledge."
— *La Divina Commedia - Dante Alighieri*

Chapter 1

Introduction

Personal transportation in the urban environment changed noticeably in the 19th century. Private automobiles allowed point-to-point travels, ensuring autonomy and privacy. Although the total percentage of households without a vehicle is increasing in the large metropolitan areas [1], the world's population is growing by one percent per year [2]. The projection of the trend implies a continuous growing demand for personal transportation. A greater number of vehicles would affect congestion, travel-time uncertainty, the likelihood of accidents, fuel consumption, and pollution. Furthermore, the more the vehicles, the more infrastructures are required, such as roads and parking spaces. The personal transportation based on private vehicles is thus not sustainable in an urban dense environment. The major challenge in urban mobility is then satisfying the demand for urban transportation and preserving the benefits of private vehicles, providing at the same time quality service for the users and the environment. A large number of vehicles on the road network has often low occupancy rate, that is an inefficient use of resources. Therefore, in the recent years, several studies in the field of the Mobility on Demand (MOD) systems investigated ride-sharing, where more travellers share the trip in a single vehicle. The key idea is to reduce the number of vehicles by increasing their occupancy. The results are promising: high capacity ride-sharing allows point-to-point transportation reducing the total travelled distance without increasing dramatically the single trip duration [3].

Customers requests have to be assigned to vehicles with the aim to guarantee a reliable and convenient ride-sharing service. To perform the assignment, the problem to solve consists in finding which trips can be shared and then matching those trips with the vehicles, such that a quality service in terms of waiting time and delay for the customers is ensured, and the travelling costs are reduced. Once the requests are combined, the goal is to find the best route which services the requests.

In literature, the MOD and Autonomous Mobility On Demand (AMOD) with ride-sharing [4, 5, 3, 6], are known as variants of the Vehicle Routing Problem (VRP) [7], of the Dynamic Pick-up and Delivery Problem (DPDP) [8] and of the Dial-A-Ride Problem (DARP) [9]. There are several challenges arising from AMOD ride-sharing systems: how to select which

travels can be combined? And how to assign the shared trips to the vehicles? One method is based on the shareability graph proposed in [10] and extended in a large-scale scenario [3]. The problem of finding the best assignment between possible shared trips and vehicles is often formulated as an optimization problem [3, 6, 11]. The cost function to minimize could be the sum of delays and waiting times [3], travelled distance [12], or operational costs and emissions [6]. However, the optimum is hard to compute in a given limited time budget. Such problem is in fact known to be computationally hard [13], being a complicated version of the Travelling Salesman Problem (TSP) [14], that is a well-known combinatorial optimization problem. In fact, the number of possible combinations between trips and vehicles grows exponentially when passengers and vehicles increase. Since a reliable service must ensure real-time booking experience, heuristic methods can help to estimate a solution [15] and reduce the size of the problem. Some well-known approaches used to bound the exploration of all possible combinations are heuristic methods [5], feasibility check [6], limits on the number of vehicles, limits on the number of possible combinations or timeout for the computation [3]. Authors in [3] were driven by the following question: is it possible to solve a large-scale ride-sharing scenario using a smart combinatorial optimization formulation? They match vehicles to the requests using a shareability graph and build a tree where the branches to explore are the possible trips; a timeout is set to limit the trips generation. Then, they formulated an Integer Linear Programming (ILP) to assign trips to the vehicles. Authors proved that their algorithm is anytime optimal: the method returns the best solution found up to the provided time budget, while an infinite time budget would lead to return the optimal solution. This practically means that more computational time leads to a better solution. We want to contribute to the work following the same idea. However, what if instead of increasing the time budget we reduce the number of combinations to explore by discarding some possible solutions violating a condition?

In the recent years, many authors tried to design algorithm based on combinatorial neural networks [16] and deep learning [17, 18] which solve the TSP. Although the results are promising and the algorithm mathematically elegant, their structure could be cumbersome to apply and replicate. In addition, to the best of our knowledge, learning methods to solve the ride-sharing problem have not been investigated deeply yet. Furthermore, we expect that also these algorithms would present almost the same degree of complexity shown in previous works [17, 18]. Our goal is to design a model using supervised learning able to predict the cost of a combinatorial optimization problem, in particular of the single-vehicle Pick-up and Delivery Problem (PDP). In this way, we can use the estimated value to steer the optimization algorithm to the optimal value, narrowing the number of possible combinations to explore, i.e. discarding the trips that have a cost much higher than the estimated. We apply supervised learning since we strongly believe that there exists some (highly non-linear) function capable to model the relation between the customers' requests, and the optimal cost of the routing problem. It is indeed experimentally found that a Neural Network (NN) trained with the results of heuristic algorithms applied to the problems is able to predict the optimal cost with a reasonable degree of accuracy.

The thesis is organized as follows. Chapter 2 introduces basic concepts and notations that will be useful during the text, such as graph theory, combinatorial optimization problems, and supervised learning. Chapter 3 gives an overview of the motivation leading this work. Chapter 4 provides the reader with the problem statement, the notations and the nomenclature adopted in the thesis. Chapter 5 focuses on the literature concerning machine learning

methods applied to routing problems. Chapter 6 shows the optimization formulations we designed to solve the TSP and the PDP. Chapter 7 presents the systematic approach used to design the supervised models and in Chapter 8 the knowledge of Chapter 7 is applied and extended. Finally, conclusions and further development proposals are discussed in the last Chapter.

Chapter 2

Preliminaries

In this chapter, we will provide the major concepts needed to understand the core of the thesis. In particular, we first will focus on the mathematical tools we use to model and solve combinatorial optimization problems; then, we will define the Vehicle Routing Problem (VRP) with its variants and then we will present the basics of supervised learning.

2-1 Graph Theory

In this section, we will describe and define a graph, its properties and the nomenclature we will adopt.

2-1-1 Basics concepts and definitions

Graphs are discrete mathematical structures which can conveniently describe and model pairwise relations between objects. Many real-world situations are indeed described by means of a diagram consisting of a set of points together with lines (edges) joining certain pairs of these points (nodes) [19]; such connection can describe any kind of relations between nodes. Due to their mathematical abstraction, graphs are very flexible and used to model different systems.

A formal description is the following [20]:

Definition 2-1.1. *A graph is a pair* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *with* $\mathcal{V} = \{v_1, v_2, \cdots, v_n\}$ *being a set of nodes or vertices and* $\mathcal{E}$ *a set of edges or arcs.*

A visual representation of graph is in Figure 2-1. In this Figure there are five nodes and six edges. Elements of edges set $\mathcal{E}$ are generally indicated with (v_i, v_j) or e_{ij}, which is an arc from v_i to v_j. Edge (v_i, v_j) or e_{ij} are said to be *outgoing* with respect to the node v_i and *ingoing* with respect to the node v_j. It is possible to associate to each edge a weight a_{ij}. A graph is said *undirected* if $a_{ij} = a_{ji}\ \forall (v_i, v_j) \in \mathcal{E}$. Intuitively, this means that the direction

between two vertices is irrelevant. If all the nodes are connected with all the other nodes, the graph is said to be fully connected.

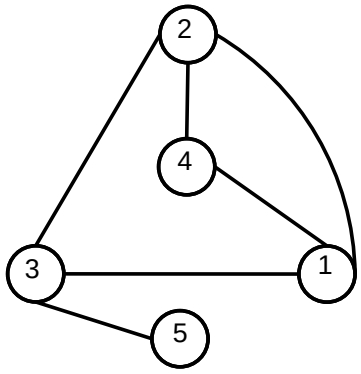


Figure 2-1: An example of an undirected not fully connected graph topology.

Graphs' properties can be investigated using associated matrices which describe their topology and structure. One of these matrix is the adjacency matrix A, whose generic element is a_{ij}. The reader might have noticed that the entry has the same nomenclature of the weights associated to the edges: both model the same quantity indeed. In this thesis, we suppose the weights non-negative. Therefore:

$$a_{ij} > 0 \text{ if } e_{ij} \in \mathcal{E}, \;\; a_{ij} = 0 \text{ otherwise.} \tag{2-1}$$

It is worth mentioning that

$$a_{ii} = 0 \;\; \forall i \in \mathcal{E},$$

that means that all the weights on the diagonal are equal to zero. The adjacency matrix can then be written as follows:

$$A = \begin{bmatrix} 0 & a_{12} & \dots & a_{1,n} \\ a_{21} & 0 & \dots & a_{2,n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{bmatrix}. \tag{2-2}$$

When the graph is undirected, the adjacency matrix is symmetric, i.e. $A = A^T$. Finally, we define a *spanning tree*. A spanning tree of a graph is a tree containing each node of the graph using a single edge just once. When the edges are weighted, the Minimum Spanning Tree (MST) is the spanning tree whose the sum of the edges of such tree is the minimum between all the possible spanning trees.

2-2 Combinatorial Optimization Problems

In this section, we will give an overview of optimization problems, with particular attention to combinatorial optimization problems. The section is not aimed to be exhaustive, but only to provide the reader with the basic tools to understand the concepts we will focus further on.

Referring to [21] and [22], we define an *optimization problem*:

Definition 2-2.1. *A mathematical optimization problem, or just optimization problem, has the form:*

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & g_i(x) \leq b_i, \ \ \forall i = 1, \ldots, m \\ & h_i(x) = 0, \ \ \forall j = 1, \ldots, \bar{h} \end{array} \tag{2-3}$$

The vector $x = (x_1, x_2, \ldots, x_n)$ is the *optimization variable* of the problem, the function $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*, the functions $g_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m$ are the *inequality constraint functions*, the constraints $b_1, \ldots, b_m$ are the limits (or bounds for the constraints), and the functions $h_j : \mathbb{R}^n \to \mathbb{R}, j = 1, \ldots, \bar{h}$ are the *equality constraints*. A vector x^* is called *optimal* or a *solution* of the problem if it has the smallest objective value among all vectors that satisfy the constraints: for any z with $g_1(z) \leq b_1, \ldots g_m(z) \leq b_m$ and $h_1(z) = \ldots = h_{\bar{h}}(z) = 0$, we have $f(z) > f(x^*)$.

We generally consider families or classes of optimization problems. For instance, the optimization problem of Eq. (2-3) is called a *linear program* if the objective and constraints functions are linear. If the optimization problem is not linear, it is called a *non-linear program*. A convex optimization problem is one in which the objective and constraint functions are convex [21].

Among the differences stated before, optimization problems can be divided into two major categories: those with continuous variables and those with discrete variables [22]. An optimization problem, like the one shown in Eq. (2-3), is nothing but an abstraction of the problem of choosing the best possible vector in $\mathbb{R}^n$ from a set of candidate choices. The variable x represents the choice made [21]. If the solution must be searched in a *finite* (or possibly countably infinite) set, the problem is what we called *combinatorial optimization problem*, otherwise it is a *continuous optimization problem*.

Example 2-2.1

Imagine you would enjoy a drink in a bar with your friends. However, since you are the driver, you should avoid drinking anything with an alcohol content greater than 10%. Therefore, you ask the bartender to prepare you a cocktail with such maximum gradation and with three different drinks up to him. To satisfy his customer, the bartender has some choices to do. First of all, since the bar has only one type of glass, he knows that the maximum quantity of liquid the glass can contain. Hence, he must choose wisely the amount of liquor and alcohol-free drinks to maintain the 10% alcoholic content. He has seven different kinds of liquors and twenty kinds of alcohol-free juices. Therefore, he must select three drinks and from that compute how much liquor put in the cocktail.
The choice of the kind of liquor is a combinatorial optimization problem. The amount, given the liquor, is a continuous optimization problems.

Looking at Figure 2-2, we present another example.

Example 2-2.2

Imagine you have an undirected connected graph with three nodes. Finding a spanning tree of minimum length is a combinatorial optimization problem: the solution is the permutation of the edges in such a way that the sum of their weights is minimized.

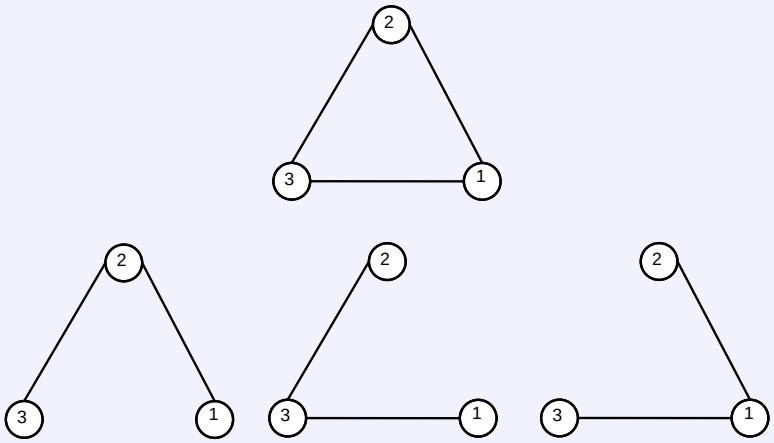


Figure 2-2: Example of a combinatorial optimization problem: find the Minimum Spanning Tree.

There are three possible combinations:

- $e_{32} + e_{21}$,
- $e_{23} + e_{31}$,
- $e_{31} + e_{12}$.

In the following subsections, we will describe some popular combinatorial optimization problems related to the transportation.

2-2-1 The Travelling Salesman Problem

One of the most common combinatorial optimization problems is the Travelling Salesman Problem (TSP) [14]. A salesman, starting from an initial depot, must visit n cities in such a way that the total travelled distance is minimized. A more formal definition of the TSP is [23]:

Definition 2-2.2. *Given a finite set of locations $\mathcal{X}$ of $|\mathcal{X}|$ points from a metric space, the Traveling Salesman Problem is to find the minimum-length tour (usually, cyclic) which visits each point $x \in \mathcal{X}$ (at least) once.*

The TSP is one of the best known combinatorial optimization problems and it can be modelled using a connected and undirected graph: each city is a node and the links between nodes are edges. Thus, the TSP can be interpreted as a sort of MST searching. The problem is combinatorial since the solution is in the set of cardinality $\bar{p}$:

$$\bar{p} = n\,!$$

where n is the number of cities to visit and $n!$ is the factorial of n, $n! = n \cdot (n-1) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$. Further in the text, we will refer without distinction to the minimum-length tour with the equivalent terms *shortest path*, and *optimal length*. An example of TSP is in Figure 2-3. An interesting property is proved in [24], where authors aim to develop an upper bound for heuristic algorithms that solves the TSP. The TSP belongs to the class of NP-complete problems [13] and it has been proved to be Non-deterministic Polynomial-time (NP)-hard. Hence, TSP cannot be solved in polynomial time, unless P=NP [25].

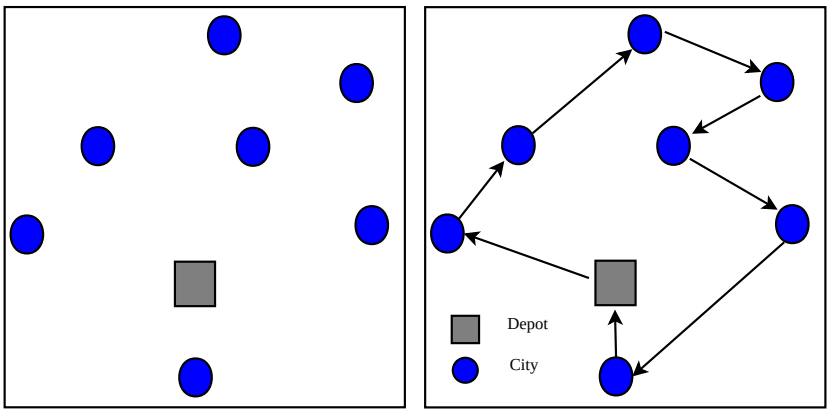


Figure 2-3: Schematic representation of the TSP. In the left image, there are the depot and the cities to visit. In the right image, the route is designed. The salesman starts and ends at the depot after visiting each city once.

A survey on the state of the algorithms used to solve the TSP is [26], where exact and approximation algorithms are described. In Chapter 6 we will show an Integer Linear Programming (ILP) formulation approach to solve the TSP. It is clear that the curse of dimensionality, where with *dimension* we here intend for the number of cities, affects the TSP, which becomes intractable for *large* instances. But what we mean by *large*? The answer is related to the running time and the application purpose. Since we want to eventually provide a real-time transportation service, the time is crucial. Therefore, it is possible that the optimal solution cannot be computed for problems size that cannot be solved in seconds.

A simple heuristic to implement to solve a TSP instance is the greedy algorithm. Greedy heuristic iteratively constructs the path by adding to the sequence the closest node to the current node (i.e. the shortest edge). The algorithm is listed in Algorithm 1. It is also possible to indicate the greedy function of a TSP with n points as

$$G(\mathbf{P}) = G(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n) \tag{2-4}$$

where $\mathbf{p}_i = [x_i, y_i]$ is the generic point describing the position of $i-$th city in the Euclidean plane, and (x_i, y_i) are the coordinates. The matrix

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_n \end{bmatrix}$$

collects all the coordinates of the n points. The function $G(\mathbf{P})$ or simply G returns the length of the route computed with Algorithm 1.

Algorithm 1: Compute greedy heuristic solution of TSP.

```
// Input:  Symmetric Adjacency matrix A
// Output:  The greedy heuristic cost shortest and the path path
// Initialization:
path=[ ], Q = {0, 1, ··· , n}, i=0, j=0, shortest=100n, c=0
for i < n do
    p=[ ];
    p[i]← i;
    Q_temp ←remove(Q,i);
    j←i;
    while ( j<n AND Q_temp NOT empty) do
        min_edge← min(A[j, l], l ∈ Q_temp, l ≠ j);
        c←c+A[j,min_edge];
        j←min_edge;
        append(p, j);
        Q_temp = remove(Q_temp, j);
        if c>shortest then
            break;
        end
    end
    if c<shortest then
        shortest← c;
        path←p;
    end
end
return shortest, path;
```

Although greedy heuristic is generally faster than optimization formulation, especially in large size instances, the results can be sub-optimal and lead to an over-approximation of the cost. Other heuristic are the Clark and Wright algorithm [27], and the Local Neighborhood Search (LNS), the Lin-Kernighan algorithm [28]. Another popular method uses a lower bound of the TSP to estimate the deviation between such bound and the optimal cost and thus heuristically determine the optimal cost. The lower bound is known as Held-Karp lower bound (HK), which provides a solution to the Linear Programming (LP) relaxation of the ILP formulation of the TSP [29, 30]. However, finding the HK is itself a combinatorial optimization problem, since it requires the computation of a sequence of MST. An iterative estimation approach of the HK is proposed in [31].

2-2-2 The Vehicle Routing Problem

VRP is another very popular combinatorial optimization problem, introduced as a generalization of the TSP. The classical VRP aims to find the optimal routes to serve customers distributed along the city with a fleet of vehicles, where each vehicle starts and ends its route

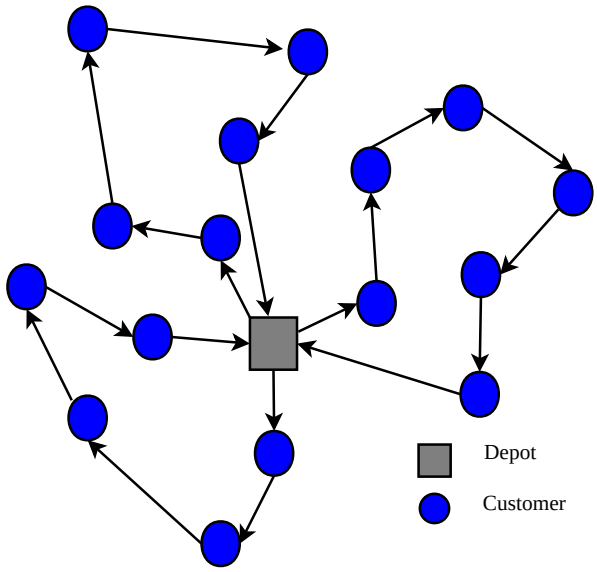


Figure 2-4: Schematic representation of the VRP with three vehicles, starting and ending at the depot after visiting once each customer.

at the (unique) depot and each customer is visited only once (i.e. the load cannot be distributed between more vehicles or more trips) by a single vehicle. A schematic representation of the problem is in Figure 2-4. In this framework, *optimal* stands for the minimum of a certain cost function. Customers are supposed to wait for a delivery or to dispatch a certain load. A mathematical formulation is generally the following [32]:

Definition 2-2.3. *The VRP is defined on a directed graph* $\mathcal{G}(\mathcal{V}, \mathcal{E})$ *where* $\mathcal{V} = \{v_0, v_1, \dots, v_n\}$ *represents the set of vertices and* $\mathcal{E} = \{(v_i, v_j) | (v_i, v_j) \in \mathcal{V}^2,\ i \neq j\}$ *is the arc set, with a cost* a_{ij} *defined over each edge* $e_{ij} \in \mathcal{E}$*. The VRP consists in finding a set of routes for the* N *vehicles based at the depot, such that each of the vertices is visited exactly once, minimizing the overall routing cost.*

The cost could represent the total travelled distance, emissions, or travel time. The vertex v_0 is traditionally the depot. The other vertices could be both the physical intersection between two roads within the network or more likely the position of customers' requests to be served. Vehicles can be homogeneous or heterogeneous, with a certain capacity that must not be exceeded (often referred to as Capacitated Vehicle Routing Problem (CVRP) [33]). The complexity is increased by including real-world aspects. There exist in fact several variants, such as the Vehicle Routing Problem with Time Windows (VRPTW) [34, 35], where each customer must be visited within a time interval, which can be soft or hard; the Multi-Depot Vehicle Routing Problem (MDVRP) [36], where more than one depot is considered: vehicles can start from and end to different depots; the Stochastic Vehicle Routing Problem (SVRP) [37, 38] where there are stochastic demands, load or travel times; the Dynamic Vehicle Routing Problem (DVRP) [32], where the demands are dynamically generated and hence the routing must be performed online; when the demands are dynamic and stochastic, that is the Dynamic Stochastic Vehicle Routing Problem (DSVRP) [39]; routing problems with varying travel times due to congestion is investigated in [40]; there exists then the Pick-up and Delivery

Problem (PDP) [41], where vehicles pick-up load at one location and delivery it at another location; finally, the Dynamic Pick-up and Delivery Problem (DPDP) [8] can models both goods and passenger transportation. For the passenger transportation, the VRP is also known in literature as the Dial-A-Ride Problem (DARP) [9].

The VRP has been proved to be NP-Hard [13] and hence exact algorithm are only efficient for small-size problems [42]. For this reason, heuristic [43, 42] and metaheuristic [35] methods are often more suitable for practical application. Local search algorithm [34], Stochastic Programming [37] and the Ant Colony Optimization [44] are also popular methods.

Variants and the existing trends in the literature of the VRP are well-documented in the extensive surveys [7, 45], where a taxonomic review of the VRP literature is provided. Another very detailed bibliography is [46], published in 1995.

2-2-3 The Pickup-and-Delivery Problem

A variant of the VRP is the PDP. In such scenario, goods (or people, see also next subsection) are transported between an origin and a destination [41, 8]. Each problem instance has n requests corresponding to the origins and n associated destinations. Therefore, a graph with $2n$ nodes can model PDP with n requests. Since destinations' nodes must be reached after the origin, the graph is not undirected for all the edges and some permutations are discarded. A more formal definition is:

Definition 2-2.4. *Consider n requests, where to each requests is associated a pick-up (origin) location and a drop-off (destination) location. The PDP is defined on a graph* $\mathcal{G}(\mathcal{V}, \mathcal{E})$ *where* $\mathcal{V} = \{v_0, v_1, \dots, v_{2n+1}\}$ *represents the set of vertices and* $\mathcal{E} = \{(v_i, v_j) | (v_i, v_j) \in \mathcal{V}^2,\ i \neq j\}$ *is the arc set, with a cost* a_{ij} *defined over each edge* $e_{ij} \in \mathcal{E}$*. Node* v_0 *represents the depot and node* v_{2n+1} *represents the final position to reach for all the vehicles. The subset* $\mathcal{V}_O \subset \mathcal{V}$*, with* $\mathcal{V}_O = v_1, \dots, v_n$ *represents the origin and the subset* $\mathcal{V}_D \subset \mathcal{V}$*, with* $\mathcal{V}_D = v_{n+1}, \dots, v_{2n}$ *the destinations. The origin* v_i *has its corresponding destination* v_{n+i}*. The PDP consists in finding a set of routes for the N vehicles based at the depot, such that each of the vertices is visited exactly once, minimizing the overall routing cost and ensuring that the nodes of subset* $\mathcal{V}_O$ *are visited before the corresponding nodes of subset* $\mathcal{V}_D$*.*

As for the TSP, it is possible to define the Adjacency matrix.

2-2-4 Dial-a-Ride Problem

The DARP it can be interpreted as a PDP with time windows [41]. The key idea is to design vehicles routes and scheduling to satisfy a set of requests, where each request includes an origin and a destination. Both origins and destinations must be reached within the given time windows interval. The main difference between DARP and PDP is the relevance given to the customer inconvenience; in general, the problems to solve are minimizing the cost subject to full demand satisfaction given a set of constraints or maximizing the satisfied demand given the vehicle availability and a set of constraints. Quality criteria are the length of the route, the waiting time, the travel time, the latency between the expected arrival time and the actual travel time. In the DARP, minimizing the latency is the often considered as the

strictest requirement, since it is addressed to reduce the passengers' discomfort. Although there exists some promising results [9], they are still limited to a relatively small scale scenario. Nevertheless, the DARP modelling is suitable to model the high-capacity Mobility on Demand (MOD).

2-3 Supervised Machine Learning methods

A supervised learning method aims to derive a black-box model of an unknown, often non-linear and non-convex system, given training input data and target output data. Referring to the routing problems, the dynamics between the features of a specific instance of the problem and the optimal value is unknown and hard to define. Therefore, usual regressor methods are not suitable to estimate the relationship between input and output. In addition, supervised learning methods can train model able to generalize to unseen data.

2-3-1 Neural Networks

Artificial Neural Networks are a functional imitation of biological neural networks [47]. The main feature of Neural Network (NN) is its ability to learn complex relations by generalizing from a limited amount of training data. Therefore, such networks are suitable to be black-box models of non-linear non-convex multivariable systems. To be more specific, NN or Multi-layer Perceptron (MLP) is a supervised learning model that learns a function $f : \mathbb{R}^p \rightarrow \mathbb{R}^o$.

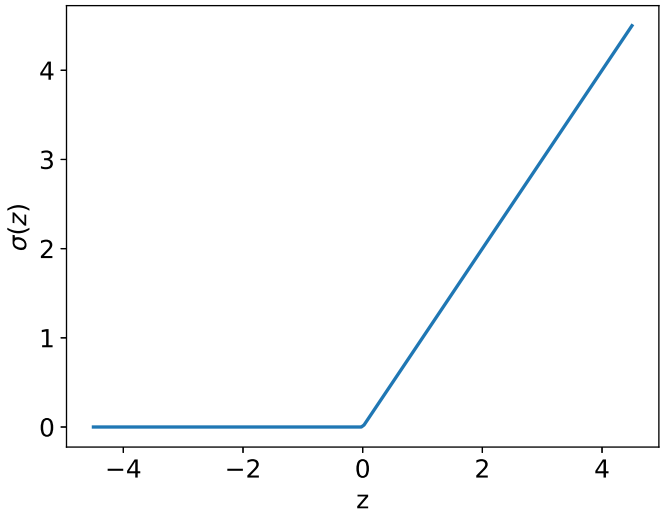


Figure 2-5: ReLu activation function.

Given a set of input features $X = [x_1, x_2, \ldots, x_p]$ and a target y, the algorithm can thus learn complex non-linear functions between input and output. The input features are weighted,

summed and then passed by a non-linear activation function $\sigma(z_j)$, with:

$$z_j = \sum_{i=1}^{p} w_{ij}^h x_i + b_j^h, \quad j = 1, \dots, h, \tag{2-5}$$

where h is the number of neurons, w_{ij}^h and b_j^h are the weight and the bias respectively. The output of the neural network will be:

$$y = \sum_{j=1}^{h} w_j^o \sigma(z_j) + b^o, \tag{2-6}$$

where we consider a single-output layer. The aim is to tune the parameters w^o and w^h in such a way that the difference between the output of the network and the target is minimized. This problem can be formulated as a non-linear optimization problem with respect to the weights. One of the most famous algorithms is a first order gradient method, the error backpropagation [48].

The key insight of NN is to "induce" non-linearity in the mapping between input and output by the means of the activation function. Two very common activations function are the ReLu function:

$$\sigma(x) = \max(0, z), \tag{2-7}$$

which can be observed in Figure 2-5, and the sigmoid function which can be observed in Figure 2-6:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2-8}$$

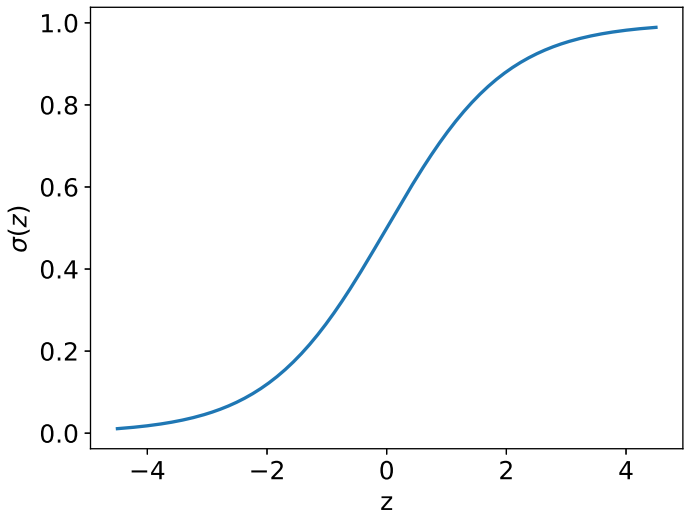


Figure 2-6: Sigmoid activation function.

From such Figures, we can see how the ReLu function is unbounded, while the sigmoid function is limited in the range $[0, 1]$.

2-3-2 Support Vector Regression

Support Vector Machine (SVM) was introduced the first time in 1995 as new machine learning method for classification problems. The machine maps non-linearly input vectors to a very high-dimension feature space. In this feature space, a linear decision surface is constructed. This non-linear mapping is called *kernel*. Special properties of the decision surface ensure high generalization ability of the learning machine [49]. Furthermore, SVM has the flexibility to represent complex functions being resistant to overfitting. This happens because SVM minimize expected generalization loss instead of minimizing expected empirical loss on training data [50]. SVM have been extended to solve regressions problems, in the so-called Support Vector Regression (SVR). The SVR uses the same principle of the correspondent classification method. In the ε-SVR the goal is to find a function that has at most ε deviation from the target for all the training data, being at the same time as flat as possible [51]. More information can be founded in the very precise essay by Smola [51].

Chapter 3

Motivation

In this chapter, we will provide an overview of the literature related to ride-sharing and how this application domain can benefit from having value predictors.

3-1 Mobility on demand

The Vehicle Routing Problem (VRP) investigated in Chapter 2 can be applied to passengers transportation, modelling Mobility on Demand (MOD) systems. In this framework, the goal is to find the best assignment between customers requests for transportation and the vehicles. The promising results of robotics raised to consider autonomous fleet of vehicles, the so-called Autonomous Mobility On Demand (AMOD) system. Allowing only one passenger on board is a strong limitation on the potentiality of MOD systems. Relaxing such constraint and providing customers with a ride-sharing service, the number of vehicles for personal transportation can be reduced, and thus reducing also congestion and emissions. The goal is then to find the optimal combinations of shared trips and assign them to the vehicles. The assignment must be performed being aware of the state of the fleet, and of customers demands distribution.

In [10], authors consider a small scale scenario. They study the benefits of ride-sharing as a function of customers inconvenience, that is the maximum tolerate delay. In this work, a graph-based model computes the optimal strategies for combining the shared trips using the shareability graph, adopted also in [3] with some modifications. Although the method is limited for a large number of passengers, the paper gives meaningful insight for further developments.
A real-time data-driven simulation framework is proposed within a large-scale scenario [4]. Such framework is able to analyze several ride-sharing realistic possible situations, taking into account the needs and the constraints of the different entities of the problem. The work highlights the challenge given by the conflicting interests of multiple stakeholders, such as Government, taxi companies, and passengers: the first aims to reduce traffic and pollution, the second to maximize profits and minimize consumption and the third to minimize additional

stops induced by ride-sharing and waiting/travel time. The authors realized that a highly-scalable and flexible method is required. The algorithm proposed in [4] is linear in the number of trips and if combined with parallelization, the approach results eventually scalable. The effective flexibility of the work is illustrated by the results obtained with different cost functions. Despite the performance, the method does not rely on autonomous vehicles and the trip-vehicle assignment is performed not optimally, but with a greedy approach. Another work which proposes a greedy real-time strategy in a non-autonomous vehicles ride-sharing scenario is [5]; given a new request and an already existing scheduling, the authors aim to find the taxi status which satisfies the request with the minimum increase in travel distance. The greedy approach does not guarantee that the total travelled distance is minimized, but such method ensures both a real-time experience and copes well with the NP-hardness [15]. Authors of [4] also claim that the greedy approach matches a more realistic scenario in which is not always possible to foresee the future trips or make changes to the past trips.

In [6], authors account the problem of managing a fleet of autonomous vehicles to accommodate transportation requests, offering point-to-point services with ride sharing. The major focus is on scheduling and admission control. The scheduling problem aims to configure the most economical schedules and routes for the AVs to satisfy the admissible requests and it is formulated as a Mixed-Integer Linear Programming (MILP). Admission control generally refers to a validation process in communication systems; in this framework, it aims to determine the set of admissible requests among all requests to produce maximum profit and it is cast as a bi-level optimization (see the reference therein [6] for more information), which embeds the scheduling problem as the major constraint. A relevant contribution of [6], is the capability to modify the previously assigned but not yet serviced requests. A different approach can be found in [52, 53], where the objective function is designed in order to minimize the number of vehicles needed to be rebalanced along the network. Rebalancing is required due to the inevitable uneven distribution of the fleet within the network since some regions tend to be more popular than others. The problem of rebalancing is investigated also in [3], where the idle vehicles aim to pick-up the unserviced requests. A further extension is in [54], where the prediction of future demands helps to spread the vehicles along the road network trying to anticipate the requests.
Being aware of the traffic condition is crucial to accurately develop the routing of the vehicles: since the objective is generally to minimize the delays, knowing the travel times on different road segments allow to compute the shortest path is crucial. In [3] the travel times are deterministic and stored in a precomputed lookup table, while in [15] heuristic methods are used to avoid the online travel times computation till it is necessary. Similarly to [3], the authors in [4] aim to reduce the computational time dedicated to the shortest path algorithm using a cache-indexing method. Although there exist different methods to perform the shortest path algorithm [55], one should bear in mind that in large-scale scenario there could be billions of shortest path queries [4]. Hence, offline precomputed shortest paths allow maintaining the scalability [3]. It is important to mention that the interval dedicated to solving the problem should be long enough to provide a good solution, but also short enough to provide users with real-time service.
This trade-off is well-achieved in a recent and promising work [3]. This paper considers a dynamic on-demand ride-sharing service in a large scale scenario, i.e. with thousands of requests [3]. The paper considers a fleet of autonomous vehicles, whose passenger capacity can vary. The goal is to find feasible trips which minimize waiting time, travel time, and delays in order

to satisfy a batch of collected requests. The trips are thus assigned to the vehicles in such a way that a certain cost function is minimized, satisfying a set of constraints. During the trip, additional riders can be picked-up. Such framework is also flexible in case of time-windows, i.e. maximum user waiting time and maximum additional delay, which can be included in the set of constraints.
The work in [3] aims to provide a general framework for the multi-vehicle multi-request routing, in order to address the problems of ride-sharing. The overall objective is computing incrementally an optimal assignment of a set of request to a set of vehicles with given capacity. The trip-vehicle assignment is performed continuously, taking into account the incoming request. The method also considers the rebalancing of the fleet.

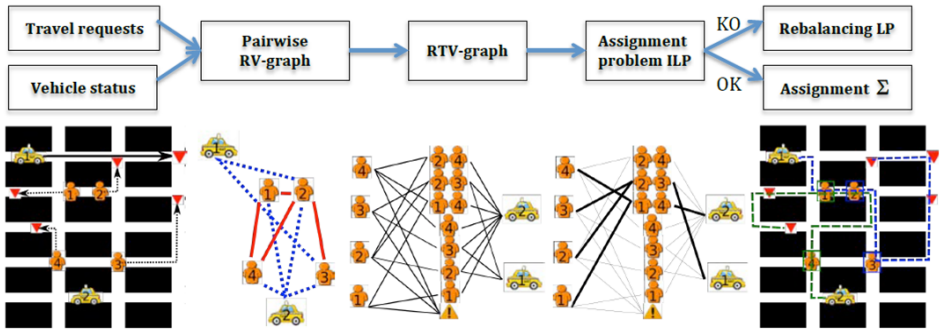


Figure 3-1: Schematic overview of the method for batch assignment of multiple requests to multiple vehicles proposed in [3]. The method consists of several steps leading to an integer linear optimization that provides an anytime optimal assignment. In the first picture from the left, we see an example of a street network with four requests (orange human, origin; red triangle, destination) and two vehicles (yellow car, origin; red triangle, destination of passenger). Vehicle 1 has one passenger, and vehicle 2 is empty. Then, go to the right, we see a pairwise shareability RV-graph of requests and vehicles. Cliques in this graph are potential trips. The next image presents RTV-graph of candidate trips and vehicles which can execute them. A node (yellow triangle) is added for requests that cannot be satisfied. After that, there is the optimal assignment given by the solution of the ILP, where vehicle 1 services requests 2 and 3 and vehicle 2 services requests 1 and 4. In the last image, the reader can observe the planned route for the two vehicles and their assigned requests. In this case, no rebalancing step is required because all requests and vehicles are assigned.

The problem shown is solved via four steps:

1. Compute a pairwise request-vehicle shareability graph, called RV-graph.
2. Compute a graph of candidate feasible trips, called RTV-graph, which merges one or more requests and vehicles able to service them.
3. Solve an Integer Linear Programming (ILP) to compute the best assignment of a set of vehicles to a set of trips.
4. Rebalance the remaining idle vehicles.

The shareability graph of the first step aims to translate spatio-temporal sharing problems into a graph-theoretic framework, capable to provide efficient solution [10]. The requests are

supposed to be dynamic. A pool of requests is maintained where new requests are added or removed. The removal can be caused when a request becomes passenger or when it cannot be matched with any vehicle due to the constraints. With a certain frequency, a batch assignment of the request to the autonomous vehicles is computed. The designed time interval in the experiments in [3] is 30s. Hence, since the requests are collected during such time interval, the overall dynamic optimization problem is split into static optimization problems, where each horizon is 30s. The batch assignment can be performed by solving the optimization problem considering the predicted state of the fleet at the assignment time.
As a variant of the VRP, the problem is NP-hard [13]. Hence, obtaining an optimal assignment can be computationally expensive, given that real-time results should be provided. A sub-optimal solution is thus returned within a runtime budget, which can be improved incrementally up to optimality. The method in [3] is referred to as anytime optimal algorithm; then, with a non-limited computational time, it would be capable to return the optimal solution.

The potential of the algorithm is based on the goodness of the RV-graph and of the RTV-graph. Saving time in generating such graphs and having a smaller size of possible trips lead the optimization algorithm to have more computational time to explore a narrowed set of solutions. Authors define trade-off (i.e. maximum number of edges per nodes, maximum number of vehicle per nodes) for the construction of the shareability graph. Speed-up algorithms can be used to prune the branches to explore [15]. The following step of the method is to explore the regions of the RV-graph for which its induced subgraph is complete, or cliques, to find feasible trips to build the RTV-graph. A trip is defined by a set of requests and it is feasible if the requests can be combined, picked-up and dropped-off by some vehicle while satisfying the constraints. A request may be part of several feasible trips of varying size, and a trip might be assigned to different vehicles. For each vehicle, a timeout is set, and after that, no more trips are explored. This choice leads to faster computation, yet to the price of sub-optimality of the solution. A trip is thus added to the RTV-graph if it results feasible. However, imagine that given a subset of requests we are able to predict which is the optimal cost (within a confidence bound) of servicing such requests. We are then predicting the cost of the optimal route. This means that if some trips have a cost outside the confidence bound, we can discard them from the RTV-graph. The goal of the thesis is, therefore, design a machine learning method able to predict the optimal cost of a routing problem. In particular, we aimed to predict the cost of single-vehicle Pick-up and Delivery Problem (PDP).

Chapter 4

Definitions and Problem Statement

In this chapter, we will provide a formal definition of the nomenclature we will adopt and explain the problem we aim to solve.

4-1 Definitions and notations

Consider the Pick-up and Delivery Problem (PDP) defined in subsection 2-2-3. Consider now a static, single-vehicle PDP without depot; the solution of the problem consists in finding the route in such a way all the requests are serviced ensuring certain constraints and minimizing the overall routing cost. Generally speaking, a request is a demand for a transportation service, given an origin where to be picked-up and destination where to be dropped off. We formally define a set of requests R:

Definition 4-1.1. *A set of n requests R is a set of tuples, where each tuple $r = (O_r, D_r)$ is defined by an origin $O_r \in \mathbb{R}^2$ and a destination $D_r \in \mathbb{R}^2$. Both origins and destination are spatially identified with their coordinates (x_r, y_r).*

Further in the text, when we handle different sets of requests with also different dimensions, we will refer to the set of requests as R_i^j, where the subscript i indicates the instance i, while the superscript j indicates the cardinality of the set.
When we refer to origin and destination as geometrical entities in the plane, we will call them indistinctly *points* or *nodes*. The former definition is related to the fact that origins and destination are geometric points in the two-dimensional plane; the latter is used since we model the problem using graph theory, where nodes are the origins/destinations and the edges are the Euclidean distances between nodes. Recalling Definition 2-2.4, we can now define the set of points (or nodes) $\mathcal{V}$:

Definition 4-1.2. *We define a set of points (nodes) $\mathcal{V} = \mathcal{V}_O \cup \mathcal{V}_D$, where*

$$\mathcal{V}_O = \{v_1, v_2, \dots, v_n\}$$

and

$$\mathcal{V}_D = \{v_{n+1}, \dots, v_{2n}\}$$

such that, being $r_1, \dots, r_n$ *some ordering over requests from set* R*, for all* $r_i \in R$ *with* $R \in \mathbb{R}^n$*, and* $i = 1, \dots, n$*, we have:*

$$v_i = O_i, \;\; v_{n+i} = D_i, \tag{4-1}$$

We thus define:

Definition 4-1.3. *The solution of PDP is the minimum-length tour which services each request, ensuring that the destination node is always reached after the corresponding origin node.*

Since we solve the PDP using an optimization formulation, we will call the length of the minimum-length tour as *optimal length.*

We said that the single vehicle has a set of requests to satisfy, say n requests. We define a route ρ as a sequence of all nodes, with dimension $2n$. The sequence of points can be re-ordered in $2n!$ permutations. Consequently, there exists $2n!$ possible routes:

$$\begin{aligned}
\rho_1 &= \{v_1, v_2, \dots, v_n, v_{n+1}, \dots, v_{2n}\} \\
\rho_2 &= \{v_2, v_1, \dots, v_n, v_{n+1}, \dots, v_{2n}\} \\
\vdots &\qquad \vdots \quad \vdots \quad \vdots \qquad \vdots \\
\rho_{2n!} &= \{v_{2n}, v_{n+1}, \dots, v_n, \dots, v_2, v_1, \}
\end{aligned}$$

The subscripts $\{1, 2, \dots, n\}$ indicates the origins, while $\{n+1, n+2, \dots, 2n\}$ indicates destination. Further in the text, we will also refer to the origin subset starting from 0 and ending with $n-1$ (and thus the destination $\{n, n+1, \dots, 2n-1\}$) to be consistent with the proposed algorithm and the code we wrote. .
We define a route as *feasible* or as *feasible* solution of the PDP if such route does not violate any precedence constraints. It means that the node v_i must precede the node v_{n+i}. Since there are some infeasible routes, the number of allowable permutations is reduced. The number $\bar{q}$ of feasible routes of a PDP with n requests is [56]:

$$\bar{q} = \frac{(2n)!}{2^n} \tag{4-2}$$

We can now define the set of feasible routes:

Definition 4-1.4. *Consider a single-vehicle PDP instance* I *defined by a set* R *of requests to be serviced. The set of requests* R *consists of* n *requests* $\{r_1, \dots, r_n\}$*, where the* i*-th request is* $r_i = (O_i, D_i)$*, with* $O_i \in \mathbb{R}^2$ *being the origin, and* $D_i \in \mathbb{R}^2$ *being the destination. There exist* $2n$ *nodes* $v \in \mathcal{V}$ *in the Euclidean plane: nodes* $v_0, v_1, \dots, v_{n-1}$ *are origins belonging to* $\mathcal{V}_O$*, and nodes* $v_n, v_{n+1}, \dots, v_{2n-1}$ *are destinations belonging to* $\mathcal{V}_D$*. Destination* v_{n+i} *corresponds to the origin* v_i*. The set of feasible routes* $\mathcal{R}$ *with cardinality* $\bar{q} = \frac{(2n)!}{2^n}$ *is the set of discrete sequences* $\rho \in \mathbb{N}^{2n}$*. The discrete sequences are permutations of all nodes* $v \in \mathcal{V}$ *such that node* v_i *precedes in the sequence the node* v_{n+i}*, with* $i = 0, \dots, n-1$*.*

The solution of a PDP instance $I = (R)$ *is denoted* $S(I)$ *is defined as the optimal route, and the value of instance* I*, denoted* $v(I)$*, is defined as the optimal length.*

Therefore, solving a PDP instance means finding an optimal route, in such a way that a cost is minimized. We aim to minimize the length of the route. It is then helpful to define a function which computes such length:

Definition 4-1.5. *Given a route $\rho \in \mathbb{N}^{2n}$, the function λ:*

$$\lambda : \underbrace{\mathbb{R}^2 \times \mathbb{R}^2 \times \cdots \times \mathbb{R}^2}_{2n \text{ times}} \to \mathbb{R} \tag{4-3}$$

compute the length of the route ρ as follows:

$$\lambda(\rho) = \sum_{i=0}^{2n-2} \sqrt{(x_j - x_l)^2 + (y_j - y_l)^2}, \;\; \rho[i] = v_j, \rho[i{+}1] = v_l, \;\; v_j \equiv (x_j, y_j), v_l \equiv (x_l, y_l) \tag{4-4}$$

.

Example 4-1.1

Consider $n = 2$ requests and the route $\rho = \{v_1, v_0, v_3, v_2\}$. Therefore, the vehicle travels along the path:

$$v_1 \to v_0 \to v_3 \to v_2 \tag{4-5}$$

It starts in v_1, ends in v_2 and reaches in between first v_0 and then v_3. The length of ρ is computed with the function $\lambda(\rho)$) of Eq. (4-4):

$$\lambda(\rho) = \lambda(v_1, v_0, v_3, v_2) = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} + \sqrt{(x_0 - x_3)^2 + (y_0 - y_3)^2} + \\ + \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2} \tag{4-6}$$

Once that we have a function computing the length of a route, we can define the function which describes the PDP:

Definition 4-1.6. *Given a PDP instance with n requests, and the set of feasible routes $\mathcal{R}$ of size $\bar{q}$, the function*

$$f(\lambda(\rho_1), \lambda(\rho_2), \cdots, \lambda(\rho_{\bar{q}})) = \min\{\lambda(\rho_1), \lambda(\rho_2), \cdots, \lambda(\rho_{\bar{q}})\}, \tag{4-7}$$

defined over the domain

$$f : \underbrace{\mathbb{R} \times \mathbb{R} \times \cdots \times \mathbb{R}}_{\bar{q} \text{ times}} \to \mathbb{R} \tag{4-8}$$

returns the length of the optimal route, that is the shortest length.

We will refer to this function with $f(\lambda)$. It is worth mentioning that $f(\lambda)$ is a composite function: f depends on λ which depends on ρ which depends on the set of request R. We have seen so far that given a set of requests $R \in \mathbb{R}^n$ how the optimal length is found; firstly, we generate all the feasible routes $\rho_i \in \mathcal{R}, i = 1, \ldots, \bar{q}$. Then, we compute their lengths through the function λ shown in Definition 4-1.5, and then we take the minimum thanks to $f(\lambda)$. The mapping between the set of requests and the optimal length is thus rather complex, requiring three steps. This process is shown in Eq. (4-9).

$$R \xrightarrow{\rho} \left\{ \begin{array}{c} \underbrace{\mathbb{R}^2 \times \mathbb{R}^2 \times \cdots \times \mathbb{R}^2}_{2n \text{ times}} \xrightarrow{\lambda(\rho_1)} \mathbb{R} \\ \underbrace{\mathbb{R}^2 \times \mathbb{R}^2 \times \cdots \times \mathbb{R}^2}_{2n \text{ times}} \xrightarrow{\lambda(\rho_2)} \mathbb{R} \\ \vdots \qquad \vdots \qquad\qquad \vdots \\ \underbrace{\mathbb{R}^2 \times \mathbb{R}^2 \times \cdots \times \mathbb{R}^2}_{2n \text{ times}} \xrightarrow{\lambda(\rho_{\bar{q}})} \mathbb{R} \end{array} \right\} \xrightarrow{f(\lambda)} \mathbb{R}. \tag{4-9}$$

Therefore, given a set R of n requests, the minimum is found once that all the lengths of feasible routes have been computed. This process is referred as *brute-force algorithm* and could be cumbersome.

Example 4-1.2

As an example, consider a PDP with five requests: it has 113400 feasible routes; thus, the function of Eq. (4-4) must be called 113400 times and the function of Eq. (4-8) must found the minimum between 113400 values.

Although optimization formulation can handle such complications, we want to emphasize that the PDP is affected by the so-called curse of dimensionality and compute the optimal length is hard in real-time.
We now define a more general function $\mathcal{F}(R)$ describing the single-vehicle PDP:

Definition 4-1.7. *Given a set R of n requests of PDP instance, the function $\mathcal{F}(R)$ defined over the domain*

$$\mathcal{F} : \underbrace{\mathbb{R}^2 \times \mathbb{R}^2 \times \cdots \times \mathbb{R}^2}_{n \text{ times}} \to \mathbb{R} \tag{4-10}$$

returns the length of the optimal route, that is the shortest length.

We aim to find a easy way to evaluate an approximation function $\bar{f}$ able to predict the value of the function shown in Eq. (4-10) given the input set of requests R.

4-2 Problem statement

The goal of this thesis is to apply machine learning methods to easily estimate a function $\bar{f}$ that given a set of requests of a PDP returns the length of the shortest route.

Definition 4-2.1. *Consider a single-vehicle PDP instance I defined by a set R of requests to be serviced. The set R consists of n requests $\{r_1, \ldots, r_n\}$, where the i-th request is $r_i = (O_i, D_i)$, with $O_i \in \mathbb{R}^2$ being the origin, and $D_i \in \mathbb{R}^2$ being the destination. We aim to find a function $\bar{f}(\varphi(R), \theta)$ in such a way that the difference $|\bar{f}(\varphi(R), \theta) - \mathcal{F}(R)|$ is minimized, where $\mathcal{F}(R)$ is the solution of the PDP instance I. We thus define a predictor $\bar{v}(I)$ as an approximation of the optimal length $v(I)$.*

The input argument $\varphi(R)$ is an unknown feature with still unknown dimensions depending on the set of requests R and will be determined in such a way that the function $\bar{f}(\varphi(R), \theta)$ is easy to evaluate. We will apply supervised learning methods to derive the model $\bar{f}(\varphi(R), \theta)$, where θ is the vector parameter of the machine learning model. The input vector to train the model will be indicated with $X \in R^{m \times p}$, and the output vector with $Y \in R^m$.

Chapter 5

State of the art

In this chapter we will provide an overview of the literature related to learning method applied to combinatorial optimization problems, emphasizing to open problems this thesis aims to solve.

5-1 Learning methods applied to combinatorial optimization problems

In this section, we will summarise the learning methods applied to solve combinatorial optimization problem and in particular the Travelling Salesman Problem (TSP).

5-1-1 Hopfield Network

One of the first work which arose the potential of Neural Network to solve combinatorial optimization problem was the so-called Hopfield-Tank network [57]. Such network computes the TSP tour of n cities using a network of n^2 neurons. The authors proposed an energy-like approach, using the circuit modelling to represent the network. They modified the energy function of the network in order to create an equivalence with the objective function of the TSP. Although the computational power was limited at that time, the results were good in small-scale ($\approx$ 10 cities). However, the network scales dreadfully; this behaviour and its sensitivity to hyper-parameters design were the greater limits. Further researches did not make evident improvements. An exhaustive review of Neural networks for combinatorial optimization, even though not very recent, is [58].

5-1-2 Neural combinatorial optimization networks with reinforcement learning

A substantial step further was possible due to deep learning. Specifically, after the introduction of attention mechanism [59] and Pointer Networks [16] the TSP is newly revisited.

Attention mechanism approach was proposed to address the neural machine translation [59]. It is based on and encode-decoder framework, where the encoder reads the input sentences (a sequence of vectors) and maps it (non-linearly) into another vector c, and the decoder aims to predict a probability over the translation given the context vector c. The most common approach is to use a Recurrent Neural Network (RNN) [60] since they are widely used in the field of learning functions over sequences from examples. However, these methods rely on a fixed a priori size of output dictionary; thus they cannot be directly applied combinatorial optimization problems since the output size depends on the length of the input sequence. Authors of [16] developed a Pointer Network architecture based on the attention mechanism to learn approximate solutions of TSP. Their model is addressed to learn the conditional probability of the output sequence of visited cities. Pointer networks use a softmax probability distribution as a pointer, indeed, to a specific position in the input sequence, rather than predicting an index value from a vocabulary of fixed size [17]. With such approach, they overcome the limitation imposed by attention mechanism. This method, being a supervised learning approach, is thus limited since the production of exact solutions is costly for a large number of cities. From such results, authors of [17] designed a neural combinatorial optimization with Reinforcement Learning (RL). The parameters of the Pointer Networks are optimized using model-free policy-based RL. Their algorithm not only overcomes the limitation of obtaining optimal values as training examples but they also empirically demonstrate that their method over-perform the Pointer Network also in terms of generalization. Although such framework works well on TSP, it performs poorly in systems varying over time such as dynamic Vehicle Routing Problem (VRP) [61]. This happens because if an element changes, the whole Pointer Network must be updated for computing the probabilities in the next decision points; in addition, the back-propagation process needs to store all the gradients since the algorithm needs to know when the dynamic elements changed. The model should, therefore, be invariant to the input sequence in order to avoid that inputs' changes affect the network. Authors of [61] argued that the RNN is a useful tool, but is likely to add unnecessary complications; thus, it should be avoided unless the input conveys sequential information (as in the language translation). The approach proposed in [61] removes the RNN encoder and leaves the RNN decoder. The mapping from the inputs to the higher dimensional space is made using embedding convolution layers. From the results shown in their paper, it seems that their approach applied to TSP has a faster training time than [17] with almost same performance, while the running time comparison with [17] is omitted. The accuracy of VRP estimation is competitive compared with some heuristic methods, even if slower. However, the paper is less informative than [17] and thus less reproducible.
A work similar to [17] which aim to solve the TSP is [18]. Authors noticed that the previous neural combinatorial networks did not include as a baseline the farthest insertion heuristic, a simple algorithm that greedily inserts the most distant node into the best possible location. Including such heuristic baseline during the learning updates improves the speed of convergence and the accuracy of the final results. The aim of that work was emphasizing that there is a structure in combinatorial optimization problem that can be exploited to train model able to predict unseen problem instances.

Following this concept, the contribution of this thesis is twofold: first, we realized that all the approaches listed in this subsection are promising, but their implementation could be cumbersome. We instead want to design a simple way to estimate the cost of a combinatorial optimization problem. More specifically, we are interested in the prediction of the single-

vehicle Pick-up and Delivery Problem (PDP) optimal cost, a problem that has not been investigated yet. Furthermore, as second goal, we will show if a heuristic feature can positively affect the prediction accuracy.

5-2 Features of interest

As stated in the previous section, we aim to train a model able to predict the cost of a PDP in a supervised way. We thus wonder what are the features that are significant during the training process. We were led to the possible features to analyze by a paper which investigated how to automatically extract features of interest from VRP instances to adapt routing algorithm to a specific VRP instance [62]. The purpose of the authors was to propose features that can be used to automatically configure vehicle routing algorithm. The features are clustered by type: nodes distribution, geometric features, graph properties, heuristic features. In our work, we applied some of these features as input to the machine learning models and inferred that the most promising direction is the one which considers the heuristic algorithm results.

Chapter 6

Optimization formulations

This thesis aims to show how supervised learning can help to estimate the cost of a Pick-up and Delivery Problem (PDP). Therefore, we need training data and validation data. We used the optimization formulation to generate the data we need to apply the supervised learning. We proceeded incrementally, starting from the Travelling Salesman Problem (TSP) formulation to the PDP formulation.

6-1 Travelling Salesman Problem

In Chapter 2 we summarised the basic of graph theory and TSP. As reported, the solution of classical TSP is a cyclic tour, starting and ending at a depot. It is important to mention that in this section we provide a Integer Linear Programming (ILP) formulation of the TSP without a depot. Hence, the tour is not cyclic and both initial and final point of the tour are unknown. Our goal is to provide an optimization formulation independent from any initial point: we only want to find the shortest path between points to maintain the formulation as general as possible.

Consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes (points) belonging to $\mathbb{R}^2$ of the 2D-plane and $\mathcal{E}$ is the set of weighted edges connecting two points. In particular, the arc (edge) e_{ij} connects the point v_i to the point v_j, where $v_i \in \mathcal{V}$ and $v_j \in \mathcal{V}$. Each arc is weighted, and the weight a is the Euclidean distance between the two connected points, $a_{ij} = |v_i - v_j|_2^2$. For the sake of simplicity, from now on the text will refer indiscriminately to the weights and to the edges themselves as their weighted version: $e_{ij} = |v_i - v_j|_2^2$. Given n points (cities), the adjacency matrix is $A \in \mathbb{R}^{n \times n}$, where each element is the edge e_{ij}. Since the graph is undirected, the matrix is symmetric; plus, the diagonal is filled with zero. These are two properties that can be exploited to speed up the optimization algorithm. The solution to TSP consists in finding the minimum-length tour which visits each point $v \in \mathcal{V}$ (at least and at maximum) once. The problem is cast using an ILP formulation:

$$\min_{x_{ij}} \quad \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} x_{ij}e_{ij} \tag{6-1}$$

$$\text{s.t.} \quad \sum_{j=0}^{n-1} x_{ij} \leq 1, \ \ \forall i \tag{6-2}$$

$$\sum_{i=0}^{n-1} x_{ij} \leq 1, \ \ \forall j \tag{6-3}$$

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} x_{ij} = n-1, \tag{6-4}$$

$$\sum_{j=0}^{n-1}\sum_{i=0}^{n-1} x_{ij} = n-1, \tag{6-5}$$

$$u_i - u_j - nx_{ij} \leq n-1 \ \ \forall (i,j) \tag{6-6}$$

$$x_{ij} \in \{0,1\}, \ \ \forall (i,j) \tag{6-7}$$

$$e_{ij} \in \mathcal{E}, \ \ \forall (i,j) \tag{6-8}$$

$$(i,j) \in \mathcal{V} \times \mathcal{V} \tag{6-9}$$

$$u_i \in \mathbb{Z}, \ \ \forall i \in \mathcal{V} \tag{6-10}$$

where Eq. (6-1) is the objective function; x_{ij} is the binary variable to optimize: if $x_{ij} = 1$, the solution contains the edge $e_{i,j}$; otherwise, it does not. We have $n \times n$ binary variables and n integer variable to optimize. For the sake of simplicity, we consider a matrix $\mathcal{X}$, where $x_{ij}, (i,j) \in \mathcal{V} \times \mathcal{V}$ is a generic entry.
We here list and explain the constraints:

- Eq. (6-2): each column of the matrix $\mathcal{X}$ must have at maximum one element equal to one. It means that each node is reached as destination maximum once.
- Eq. (6-3): each row of the matrix $\mathcal{X}$ must have at maximum one element equal to one. It means that from each node only one branch departs.
- Eq. (6-4) ensures that the sum of the elements column by column must be $n-1$ and Eq. (6-5) that the sum of the elements row by row must be $n-1$.
- Eq. (6-6): this constraint, using a dummy variable u, avoid the formation of sub-tours: there is a single tour which connects all the nodes.
- Eq. (6-7) states that the optimization variables x_{ij} must be binary, Eq. (6-8) that the arcs e_{ij} belong to the edge set, Eq. (6-9) that the nodes i and the nodes$j$ belong to the node set, and Eq. (6-10): the dummy optimization variables u_i must be integer.

6-2 Pick-up and Delivery Problem

We are now formulation the Euclidean PDP without depot following the same approach of the TSP formulation of the previous section.

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes (points) belonging to $\mathbb{R}^2$ of the 2D-plane and $\mathcal{E}$ is the set of weighted edges connecting two points. The arc (edge) e_{ij} connects the node $v_i \in \mathcal{V}$ to the node $v_j \in \mathcal{V}$. Each arc is weighted, and the weight a_{ij} is the Euclidean distance $a_{ij} = |v_i - v_j|_2^2$. As in the previous section, we will refer to the edges as their weighted version: $e_{ij} = |v_i - v_j|_2^2$. Given n requests, to each request is assigned an origin and a destination. There are then two types of nodes: origin $v_i \in \mathcal{V}_O$ and destination $v_{n+i} \in \mathcal{V}_D$. In this case, we have a total of $2n$ nodes. Therefore, nodes $\{v_0, \dots, v_{n-1}\}$ are the origins and $\{v_n, \dots, v_{2n-1}\}$ are the destinations. The adjacency matrix is $A \in \mathbb{R}^{2n \times 2n}$, where each element is the edge e_{ij}. The solution of PDP consist in finding the minimum-length tour which visits each nodes $v \in \mathcal{V}$ (at least and at maximum) once, ensuring that the destination node v_{n+i} is always reached after the corresponding origin node v_i.

The problem is cast using an ILP formulation.

$$\min_{x_{ij}} \quad \sum_{i=0}^{2n-1} \sum_{j=0}^{2n-1} x_{ij} e_{ij} \tag{6-11}$$

s.t.

$$\sum_{j=0}^{2n-1} x_{ij} \leq 1, \ \ \forall i \tag{6-12}$$

$$\sum_{i=0}^{2n-1} x_{ij} \leq 1, \ \ \forall j \tag{6-13}$$

$$b_{ki} \leq b_{kj} + (1 - x_{ij}), \ \ \forall (i,j) \ \ \forall k \neq i \tag{6-14}$$

$$b_{kj} \leq b_{ki} + (1 - x_{ij}), \ \ \forall (i,j) \ \ \forall k \neq i \tag{6-15}$$

$$x_{ij} \leq b_{ij}, \ \ \forall (i,j) \tag{6-16}$$

$$b_{ii} = 0, \ \ \forall i \tag{6-17}$$

$$b_{i,i+n} = 0, \ \ \forall i \in \{0, \dots, n-1\} \tag{6-18}$$

$$a_{i+n,i} = 0, \ \ \forall i \in \{0, \dots, n-1\} \tag{6-19}$$

$$\sum_{i=0}^{n-1} \sum_{j=0}^{2n-1} x_{ij} = n - 1, \tag{6-20}$$

$$\sum_{i=0}^{2n-1} \sum_{j=n}^{2n-1} x_{ij} = n - 1, \tag{6-21}$$

$$\sum_{j=0}^{2n-1} x_{ij} = 1, \ \ \forall i \in \{0, \dots, n-1\} \tag{6-22}$$

$$\sum_{i=0}^{2n-1} x_{ij} = 1, \ \ \forall j \in \{n, \dots, 2n\} \tag{6-23}$$

$$x_{ij} \in \{0, 1\}, \ \ \forall (i,j) \tag{6-24}$$

$$e_{ij} \in \mathcal{E}, \ \ \forall (i,j) \tag{6-25}$$

$$(i,j) \in \mathcal{V} \times \mathcal{V} \tag{6-26}$$

$$b_{ij} \in \{0, 1\}, \ \ \forall (i,j) \tag{6-27}$$

Eq. (6-11) is the objective function: x_{ij} is the binary variable to optimize: if $x_{ij} = 1$, the

solution contains the edge $e_{i,j}$; otherwise, it does not. We have $2n \times 2n + 2n \times 2n$ binary variables to optimize. For the sake of simplicity, we consider a matrix $\mathcal{X}$, where $x_{ij}, (i,j) \in \mathcal{V} \times \mathcal{V}$ is a generic entry.
We here list and explain the constraints:

- Eq. (6-12): each column of the matrix $\mathcal{X}$ must have at maximum one element equal to one. It means that each node is reached as destination maximum once.
- Eq. (6-13): each row of the matrix $\mathcal{X}$ must have at maximum one element equal to one. It means that from each node only one branch departs.
- Constraints Eq. (6-14),Eq. (6-15), Eq. (6-16), Eq. (6-17) and Eq. (6-18) are intended to impose precedence ordering [63]. An auxiliary variable b_{ij} allows to avoid that destination precedes the corresponding origin in the route. In particular, constraints Eq. (6-14), Eq. (6-15) are called *copy constraints*, since their function is to copy all the value of b_{ki} to b_{kj} . If $x_{ij} = 1$, means that node i is immediately before node j and thus constraints Eq. (6-14) and Eq. (6-15) force $b_{ki} = b_{kj}$. Then, constraint Eq. (6-18) and Eq. (6-19), thanks to the relation Eq. (6-16), are prior constraints, forcing the pickup node to be visited before the corresponding delivery node.
- Eq. (6-20) ensures that all the origins but one (the first in the route), must have both incoming and outcoming flow, while Eq. (6-21) ensures that all the destinations but one (the last in the route), must have both incoming and outcoming flow.
- Eq. (6-22) ensures that all the origins have outcoming flow and Eq. (6-23) ensures that all the destination have incoming flow.
- Eq. (6-24) states that the optimization variables x_{ij} must be binary, Eq. (6-25) that the arcs e_{ij} belong to the edge set, Eq. (6-26) that the nodes i and the nodes$j$ belong to the node set, and Eq. (6-27) that the auxiliary optimization variables b_{iJ} must be binary.

Chapter 7

Supervised learning applied to combinatorial optimization problems

In this chapter, we will show the process that led us to a supervised learning model which predicts the length of a single-vehicle Pick-up and Delivery Problem (PDP) tour with two requests. Chapter 5 outlined existing learning methods in literature applied to combinatorial optimization problems. Such models have complex structures and are applied to predict the route, that is the order of the cities to visit (or the passengers' origin/destination). The goal of this chapter is twofold: first, we investigate how the input features affect the scalability of the supervised learning models when the desired output to predict is the length of the optimal route solving the single-vehicle PDP; then, we infer the most suitable learning method for this class of problems.

7-1 The optimal length function

A model able to predict the output of a PDP instance predicts the length of the optimal route (see Chapter 4). We saw that we can define a function $f(\lambda)$ which returns the shortest length among all possible routes λ. Therefore, the minimum feasible route is the minimum of the function shown in Eq. (4-7) and here re-proposed:

$$f(\lambda(\rho_1), \lambda(\rho_2), \cdots, \lambda(\rho_{\bar{q}})) = \min\{\lambda(\rho_1), \lambda(\rho_2), \cdots, \lambda(\rho_{\bar{q}})\}, \tag{7-1}$$

where $\bar{q} = \dfrac{(2n)!}{2^n}$ is the number of allowable feasible routes. It represent the cardinality of the set of feasible routes $\mathcal{R}$ and it is the number of permutations that satisfy the precedence constraints. We also define a more general function mapping from the set of requests R of a PDP to the optimal length (see Eq. (4-10)):

$$\mathcal{F} : \underbrace{\mathbb{R}^2 \times \mathbb{R}^2 \times \cdots \times \mathbb{R}^2}_{n \text{ times}} \to \mathbb{R} \tag{7-2}$$

The function in Eq. (7-2) is expected to be a highly non-linear function with multiple local minima. But how complex is this function?
As an example, consider two requests r_1, r_2. Each request has an origin $O_i = (x_i^o, y_i^o)$ and a destination $D_i = (x_i^d, y_i^d)$, with $i \in \{1, 2\}$. We will refer to both origins and destinations as *nodes* or *points*. Using the nomenclature of Chapter 4 to make the equation more readable getting rid of an index, we can refer to origin and destination as points $v_i \equiv (x_i, y_i)$ in the Euclidean plane. Therefore, v_1, v_2 are the origins and v_3, v_4 are the corresponding destinations. Imagine now that the shortest route ρ^* which services these requests is

$$\rho^* = \{v_1, v_2, v_4, v_3\},$$

and therefore the path to follow is:

$$O_1 \to O_2 \to D_2 \to D_1.$$

The length L of this path is computed using Eq. (4-4):

$$L = \lambda(\rho^*) = \lambda(v_1, v_2, v_4, v_3),$$

where ρ^* indicates the optimal route.

Now, the length of the route ρ^* computed with the function $\lambda(v_1, v_2, v_4, v_3)$ is:

$$L = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + \sqrt{(x_2 - x_4)^2 + (y_2 - y_4)^2} + \sqrt{(x_4 - x_3)^2 + (y_4 - y_3)^2},$$

where each term is a Euclidean distance. The same procedure can be applied to different routes and the function Eq. (7-1) returns the minimum over such lengths.

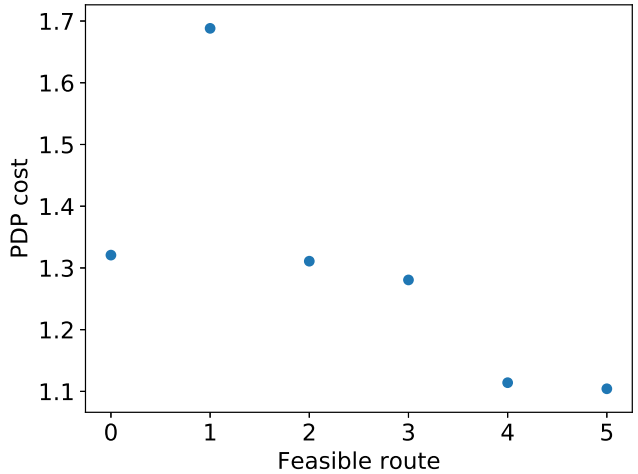


Figure 7-1: Graphical representation of function in Eq. (7-1) applied to a 2-requests Pick-up and Delivery Problem instance.

Table 7-1: Nodes coordinates of a PDP instance (see also Figure 7-1).

	x	y
O_0: node 0	0.570	0.514
O_1: node 1	0.937	0.0207
D_0: node 2	0.754	0.293
D_1: node 3	0.449	0.515

Example 7-1.1

To simplify the concept. consider a 2−requests PDP. There are four nodes to connect: two origins v_0, v_1 and two associates destinations v_2, v_3. There exist $4! = 24$ permutation of the four nodes v_0, v_1, v_2, v_3. However, due to precedence constraints, some permutations are infeasible and thus discarded. The feasible permutations (routes) are:

$$\begin{aligned}\rho_0 &= \{v_0, v_1, v_2, v_3\}\\ \rho_1 &= \{v_0, v_1, v_3, v_2\}\\ \rho_2 &= \{v_0, v_2, v_1, v_3\}\\ \rho_3 &= \{v_1, v_0, v_2, v_3\}\\ \rho_4 &= \{v_1, v_0, v_3, v_2\}\\ \rho_5 &= \{v_1, v_3, v_0, v_2\}.\end{aligned}$$

We assign to each permutation a cost, that is the length of the route, computed as in Eq. (4-4). Consider now four nodes whose coordinates are in Table 7-1. The plot in Figure 7-1 is indeed the function Eq. (4-7) applied to this example. From such Figure, we infer that the route for which the length is minimized is permutation 5:

$$\rho_5 = \{v_1, v_3, v_0, v_2\},$$

and thus the optimal route is $O_1 \to D_1 \to O_0 \to D_0$.

Although the function of Eq. (7-1) represented in Figure 7-1 has a unique minimum, each point in such Figure is computed using Eq. (4-4) and it is the sum of three Euclidean distances. The minimum must be found between the outcomes of six non-linear functions with probably multiple local minima. We realize that even the simplest PDP case presents several challenges. A graphical intuition may be helpful. Consider again a 2-requests PDP, where three nodes $\{v_0, v_1, v_2\}$ are fixed and the fourth P is varying. Node v_2 is the associated destination to origin v_0 and P is the associated destination to origin v_1. In this example, we want to solve a PDP where the variable point is the last nodes in the route. Therefore, we have only three feasible permutations:

$$\begin{aligned}\rho_0 &= \{v_0, v_1, v_2\}\\ \rho_1 &= \{v_0, v_2, v_1\}\\ \rho_2 &= \{v_1, v_0, v_2\}\end{aligned}$$

Table 7-2: Nodes coordinates. The set $\mathcal{X}$ and $\mathcal{Y}$ contain the coordinates ranging from 0 to 1 with a step of $1/l$.

	x	y
O_0: node 0	0.622	0.951
O_1: node 1	0.855	0.264
D_0: node 2	0.340	0.604
D_1: node 3	$x \in \mathcal{X}$	$y \in \mathcal{Y}$

Imagine now that instead of finding the optimal route we iteratively compute the length for each route, varying P. Figure 7-2 shows the experiments performed with the three-fixed points listed in Table 7-2. The coordinates of P vary in the interval $[0, 1]$ with step $1/l$ and $l = 20$. We thus obtain 400 possible combinations. As can be observed in Figure 7-2 the length depends on the route and it is not guaranteed that the for each P the optimal length is ensured by the same route.

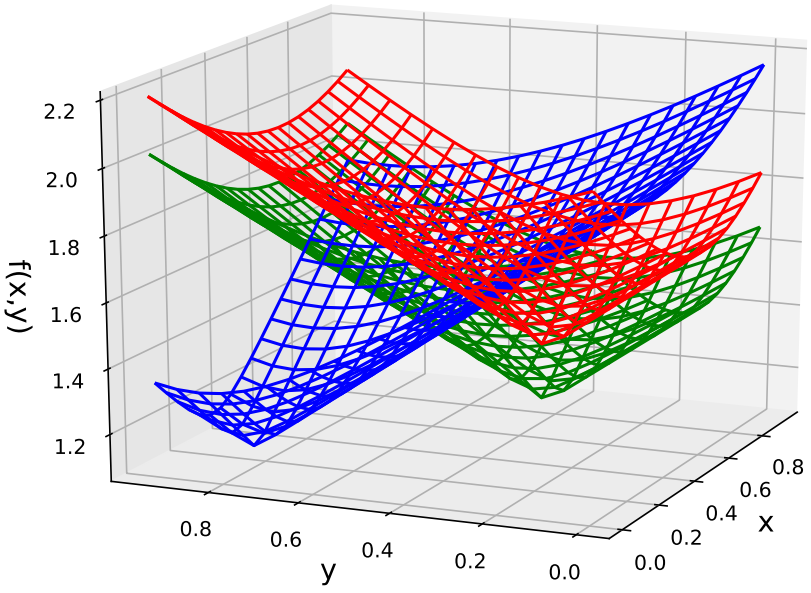


Figure 7-2: Three-dimensional plot of the length of the routes as function of a variable point.

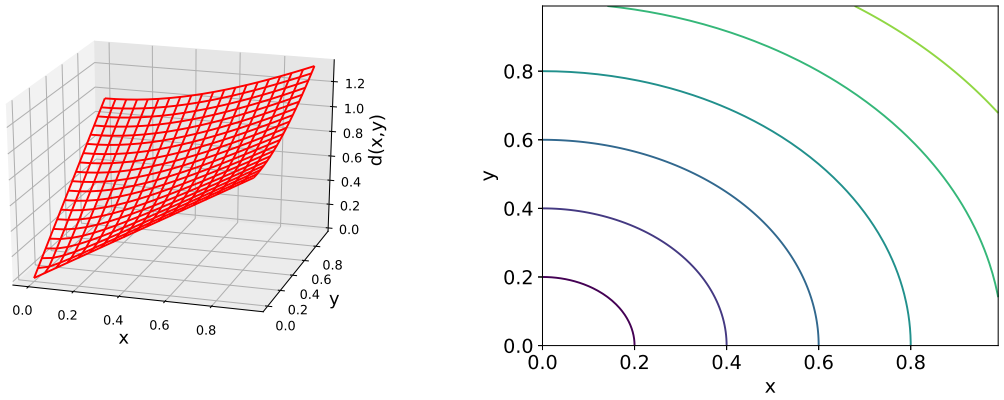
Predicting the length of a route could be manageable, but predicting the optimal length is more challenging indeed. In this chapter, we aim to provide a systematic investigation about the complexity of machine learning method to generate model able to predict the PDP optimal cost.

7-2 The Euclidean norm

As stated in the previous section, the solution of the PDP is nothing but the feasible route of minimum length. Such length is a sum of Euclidean distances. But how complex would be predicting only the Euclidean distance function? The simplest case is when one of the two points is the origin $O(0,0)$ of the Euclidean plane, and the other is $P(x,y)$. Given these two points, the Euclidean distance is:

$$d(x,y) = \sqrt{x^2+y^2}, \tag{7-3}$$

which is also called Euclidean norm (of P). The shape of this function is in Figure 7-3.



(a) 3D representation. **(b)** Contour plot.

Figure 7-3: 3D visualization of the function in Eq. (7-3) and its contour plot.

In the following paragraphs, two supervised learning methods will be discussed: Multi-layer Perceptron (MLP) (or Neural Network (NN)) and Support Vector Regression (SVR). These two models are trained in such a way that they are able to approximate the non-linear function of Eq. (7-3) shown in Figure 7-3.
The training input features for these models are the coordinates (x_i, y_i) of the generic points P_i, $i = 1, \dots, m$ where m is the dimension of the training set. The entries of the output vector are the values $\sqrt{x_i^2+y_i^2}$. Input and output vector are thus respectively:

$$X = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_m & y_m \end{bmatrix} \in \mathbb{R}^{m\times 2}, \quad Y = \begin{bmatrix} \sqrt{x_1^2+y_1^2} \\ \sqrt{x_2^2+y_2^2} \\ \vdots \\ \sqrt{x_m^2+y_m^2} \end{bmatrix} \in \mathbb{R}^m, \tag{7-4}$$

Training dataset and validation dataset shown further have dimensions respectively $m = 3000$ and $m_{val} = 300$.

Neural Networks The goal of this chapter is to eventually train a model which predicts the optimal length of the tour solving the 2-requests PDP. Therefore, the performance of the

model approximating the Euclidean norm will be evaluated in preparation for such purpose. As a general rule, the model should be as scalable as possible. Thus, we will focus on:

- the validation error, that is a Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{\sum_i^{m_{val}} (y_{val}^i - y_{pred}^i)^2}{m_{val}}} \tag{7-5}$$

  where y_{val}^i and y_{pred}^i are respectively the target value of the $i-$th validation example and the value estimated using the trained model applied the $i-$th validation example.

- the number of hidden layers and the number of hidden neurons h
- the training time.

A crucial aspect of the MLP design is the choice of the activation function. The concept behind this NN is to estimate a model using regression technique; we selected as activation function the ReLu function (see Eq. (2-7)) over the sigmoid activation function (see Eq. (2-8)) for two reasons: first of all, the ReLu function allows a network to easily obtain sparse representations [64] and it reduces the problem of vanishing gradients, that is one of the major drawbacks of sigmoid activation function.

Although such activation function has a small training time and allows the sparsity of the network activation, there is a relevant aspect that can affect the performance; imagine that the weight update is based on a gradient descent method. Consider now the case when the input of a neuron is smaller than zero, and therefore the output will be zero. But this implies that the gradient will be zero, and hence the related weights will not be updated. Such phenomenon means that when a neuron assumes state zero, it will never leave that state and will become blind to the variation of the error or of the input. This problem is known as *dying ReLu problem* [65].

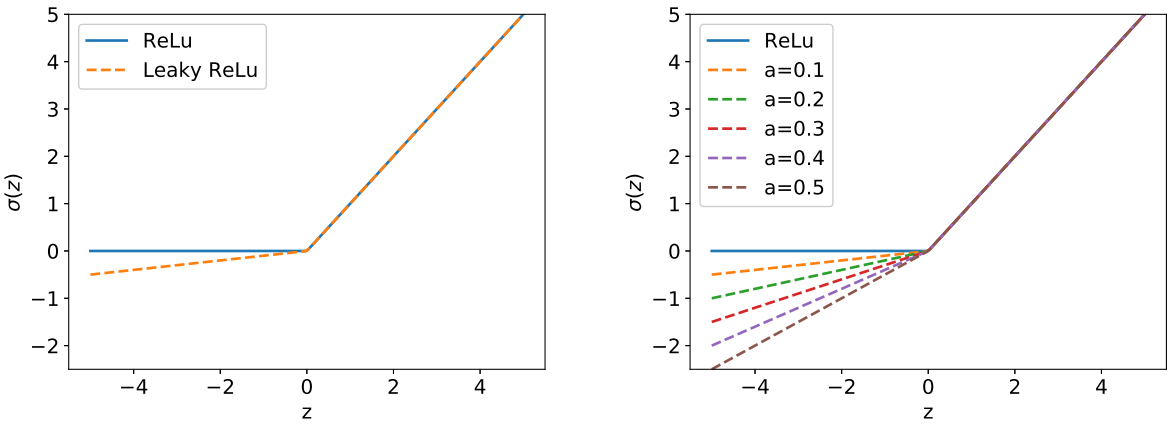


Figure 7-4: The leaky ReLu function compared with the classic ReLu function.

To overcome such problem, a common solution is to slightly modify the zero slope part of the function, as follows:

$$\sigma(z) = \begin{cases} z & z > 0 \\ az & z \leq 0 \end{cases} \tag{7-6}$$

Table 7-3: Performance of the Neural Networks predicting the Euclidean norm shown in Eq. (7-3).

Activation Function	Error	Neurons	Training time (s)
ReLu	3×10^{-3}	74	0.13
Sigmoid	5.57×10^{-2}	67	0.51
ReLu	1.7×10^{-3}	534	4.38

where a is a parameter which determines the slope. The activation function in Equation Eq. (7-6) is known as *leaky ReLu* [65]. Some examples can be seen in Figure 7-4.

The ReLu function is a suitable choice to obtain a fast learning time. In addition, since the function to approximate is relatively smooth, one hidden layer should be sufficient to achieve a good validation error. Finally, the number of neurons have been determined in such a way that the minimum error is obtained. The output of the ReLu function lies in the range $[0, \infty]$; therefore, it is not bounded, differently from the sigmoid activation function which is instead bounded in the range $[0, 1]$. The function is shown in Eq. (2-8) and Figure 2-6.

Such function has a smoother gradient and small change in the input, when $z \in [-2, 2]$, causes a significant change in the output, as can be seen in Figure 2-6. The sigmoid function has furthermore an advantage in terms of number of neurons; in fact, it has been proved that under certain assumptions, feedforward networks with one layer of sigmoidal nonlinearities achieve integrated squared error of order $\mathcal{O}(\frac{1}{h})$, where h is the number of hidden neurons [66]. Table 7-3 shows the performance of the networks: a model with ReLu activation function is able to provide a validation error of the order $\approx 10^{-3}$ with only 74 neurons. Moreover, as it could be expected, the training time is smaller for the ReLu function indeed. There were, in fact, fewer neurons and smaller computational expense, due to the fact that ReLu relation is mathematically simpler. The last row of Table 7-3 shows the price of having an even smaller error: the number of neurons is an order of magnitude greater, leading to a much longer training time.

Support Vector Regression Support Vector Machine (SVM) is a very popular approach for supervised learning, especially when any prior knowledge of the system is available [50]. The Euclidean norm is not actually the case. However, it is relevant to inspect the performance of such approach in terms of training time, validation error and complexity, as for the NNs, in prevision of further supervised learning problems.

As explained in Chapter 2, SVM can be extended to solve regression problem [67], leading to the so-called SVR. Due to the non-linearity of the function to predict (see Eq. (7-3)), a kernel is required to map into the high-dimensional feature space. The most suitable kernel is the Gaussian Radial Basis Function (RBF):

$$K(x, x') = e^{-\frac{\|x-x'\|^2}{2\sigma^2}} \tag{7-7}$$

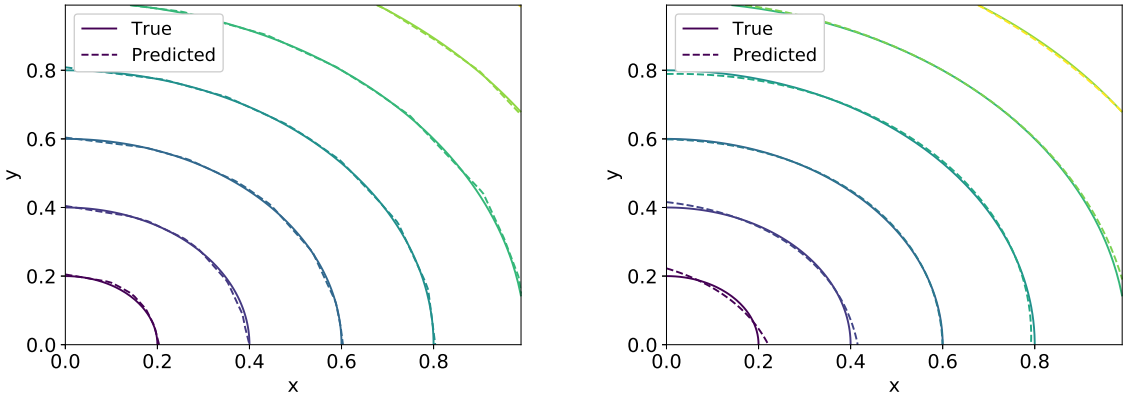
Once that the kernel has been chosen, one should select the penalty parameter associated with the error, that is C. Such parameter determines the trade-off between the flatness of the function to approximate and the amount up to which deviation from ε are tolerated [51]. A relatively small value of C, that is $C = 2$, has been chosen in order to obtain a greater

Table 7-4: Performance of the Support Vector Regression model predicting the Euclidean norm shown in Eq. (7-3)

C	Error	Training time (s)
2	4.01×10^{-3}	0.55

generalization. Other kernels and value of the penalty terms led to a worse performance in terms of training time and validation error. Table 7-4 shows that SVR error is greater compared to the NN and such error is also achieved with a longer training time.

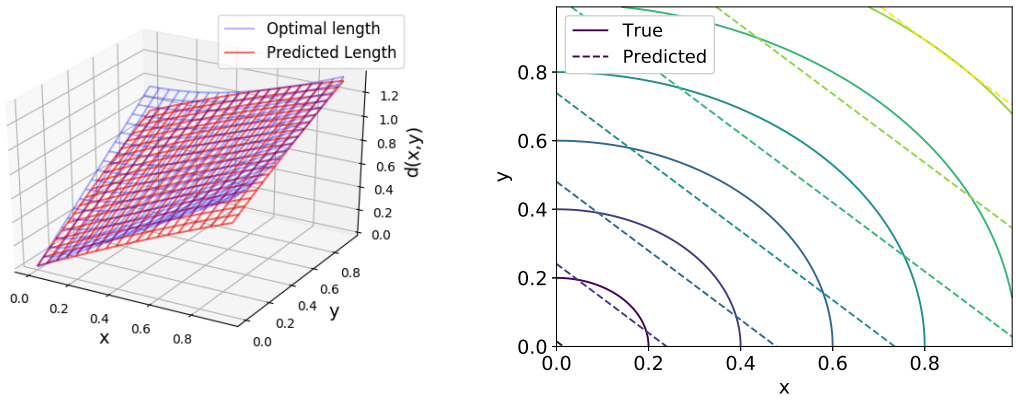
Testing of the models It is now useful to visualize the learnt models performance, that is their ability to replicate the shape shown in Figure 7-3. Looking at Table 7-3, the NN with the ReLu activation function had the smallest error. Therefore, as expected, the shape of the predicted function and the Euclidean norm from 7-3 overlap, as can be observed in Figure 7-5a. On the other hand, even if the sigmoid activation function has still an average low error, its ability to reproduce the shape of Figure 7-3 is poor compared to the ReLu activation function. This can be observed both in Figure 7-6a and in the corresponding contour plot in Figure 7-6b.



(a) ReLu activation function. **(b)** SVR

Figure 7-5: Contour plots of two different models. The solid line is the Euclidean function of Eq. (7-3), the dashed line is the predicted using NN or SVR.

In the previous paragraph, it was shown that the error of models trained with SVR had the same order of magnitude as the NN with ReLu activation function. Such performance can be appreciated in Figure 7-5b. It can be noticed how the accuracy worsens when the point is closer to the origin (see the first purple line for $0 \leq x \leq 0.2$ and $0 \leq y \leq 0.2$ in Figure 7-5b) and when one of the two coordinates is close to the zero. The performance of models investigated so far are summarized in Table 7-5, ordered in ascending order with respect to the training time.



(a) 3d representation. **(b)** Contour plot.

Figure 7-6: 3D visualization of the Neural Networks with sigmoid activation function performance and its contour plot.

Table 7-5: Summary of models performance to approximate the Euclidean norm shown in Eq. (7-3).

Model	Error	Training time (s)	Figure
NN with ReLu	3×10^{-3}	0.13	Figure 7-5a
SVR	4.01×10^{-3}	0.55	Figure 7-5b
NN with sigmoid	5.57×10^{-2}	0.51	Figure 7-6b

7-3 Euclidean distance between two points

In subsection 7-2, the function to approximate was the Euclidean norm of a point within the unit square. It depended on two parameters, x-coordinate and y-coordinate. Imagine now that the Euclidean distance is computed between two random points within the unit square. The formula of Equation Eq. (7-3) extends to:

$$d(x_1, x_2, y_1, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{7-8}$$

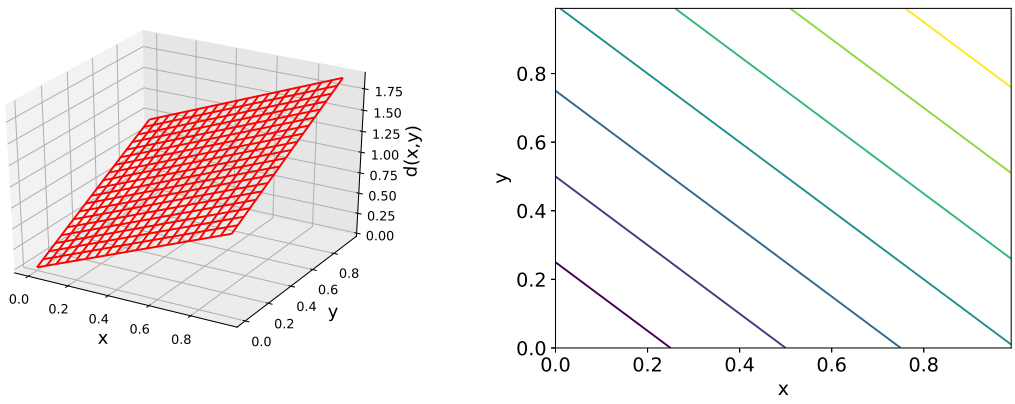
The function to approximate has hence four parameters. The question now is: how do scale the models trained before? From this simple example, it can be inferred which model is the most suitable to approximate the NP-hard problem. Looking at Table 7-6, some considerations can be made. First of all, consider the NNs; to achieve an error of the same order of magnitude as in the previous case, the complexity of the network must increase. This was expected since the number of input parameters doubled. On the other hand, such result can be obtained at the price of much slower training time, due to the higher number of neurons. It is worth noticing that a NN with 159 hidden neurons slightly overcomes the performance of a net with 99 neurons, but the training time is greater than twice. A faster training time is obtained with the sigmoid activation function; in addition, the number of neurons required to obtain an error of the same magnitude compared to the previous case is smaller ($h = 12$). In this case, the advantages of sigmoid activation function are more evident indeed: when the dimension of

Table 7-6: Performance of the models approximating Eq. (7-8). For the NN, h stands for the number of *hidden* neuros. For the SVR, C is the penalty term associated to the error.

Model	Error	Hyper-parameter	Training time (s)
NN with ReLu	4.38×10^{-3}	$h = 99$	0.92
NN with ReLu	4.33×10^{-3}	$h = 159$	2.12
NN with sigmoid	5.6×10^{-2}	$h = 12$	0.21
SVR	4.16×10^{-3}	$C = 20$	5.82

the NN input increases, the sigmoid activation function provide acceptable performance with a smaller number of neurons, compared to other series expansion. Finally, the SVR model achieves the same error as in the previous case with a training time 10 times higher.
From such experiments, we inferred that the scalability is compromised, as expected, even if in this simple scenario. We expect also that in a Euclidean combinatorial optimization problem the performance would worsen more. From the investigations performed in this section, we concluded that the coordinates are not the most suitable candidate as input feature to supervised learning models.

7-4 Alternative distance measure



(a) 3d representation. **(b)** Contour plot.

Figure 7-7: 3D visualization of the $\mathcal{L}_1$-norm of a 2-dimensional vector and its contour plot.

For the sake of completeness, consider now a different measure, that is the $\mathcal{L}_1$-distance. Consider first the norm; as the Euclidean norm, the $\mathcal{L}_1$-norm is a mapping from $\mathbb{R}^m$ to $\mathbb{R}$ defined as follows:

Definition 7-4.1. Given the vector $\mathbf{x} \in \mathbb{R}^m$, the $\mathcal{L}_1$-norm is:

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{m} |x_i| \tag{7-9}$$

Table 7-7: Performance of the Neural Networks predicting the norm.

	Error	Neurons	Training time (s)
$\mathcal{L}_1$-norm	9.37×10^{-5}	7	0.0035
Euclidean norm	3×10^{-3}	74	0.13
$\mathcal{L}_1$-distance	7.3×10^{-4}	9	0.15
Euclidean distance	4.38×10^{-3}	99	0.92

From such definition, the $\mathcal{L}_1$-distance between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^m$ is

$$d_{\mathcal{L}_1}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{m} |x_i - y_i| \tag{7-10}$$

Therefore, the $\mathcal{L}_1$-distance between two-dimensional vectors is:

$$d_{\mathcal{L}_1}(\mathbf{p}_1, \mathbf{p}_2) = |x_1 - x_2| + |y_1 - y_2| \tag{7-11}$$

The vector $\mathbf{p}_i$ can be considered as a point in the two-dimensional plane and its components (x_i, y_i) are the coordinates.

Compared to the Euclidean distance, the $\mathcal{L}_1$-distance complexity grows linearly, while the former has a quadratic relation. Since the relation is then simpler, the NN achieves better performance in predicting $\mathcal{L}_1$-distance. Figure 7-7 shows the 3D representation of Eq. (7-9). The shape is flatter than the Euclidean norm (see Figure 7-3) and thus the accuracy of the NN is expected to be better in predicting the two-points distance.

We trained a NN with ReLu activation function for predicting both the distance and the norm. The validation errors resulted smaller than the Euclidean approximation model, as can be observed in Table 7-7.

7-5 The Euclidean Pick-up and Delivery Problem

In the previous section, we investigated the complexity related to the supervised learning approaches to predict the Euclidean distance. In this section, we will focus on how to find the predictor $\bar{f}(\varphi(R))$ which estimated the optimal cost $\mathcal{F}(R)$ of the simplest PDP, that is a PDP with $n = 2$ requests.

7-5-1 Generate target data

Since we use a supervised learning approach, the target training data are required. In this section we will show the performance of the PDP optimization formulation shown in Chapter 6, emphasizing the results that will be useful for the training. For a matter of simplicity, we will show the validation data: we collected data for $n = 1, \dots, 8$ and for each n we did $m_{val} = 300$ experiments. In each experiment, we sampled n requests with a random distribution. The points have coordinates within the range $[0, 1]$. Figure 7-8 shows how the optimal cost varies by changing the number of requests. We added a jitter along the x-axis

to make the Figure more readable. We also drew a line connecting the average for each n and a two bound indicating the mean plus/minus the standard deviation. We noticed that the trajectory of the mean follows a square-root-like function. It has been proved that if n pick-up and delivery pairs are sampled randomly with a uniform distribution on a square and on a Euclidean region of area a, the length Y_n of an optimal tour satisfies [68]:

$$\lim_{n\to\infty} \frac{Y_n}{\sqrt{n}} = \frac{4}{3}\sqrt{2ab} \tag{7-12}$$

where b is the well-known Travelling Salesman Problem (TSP) constant [69], that is $b \approx 0.713$.

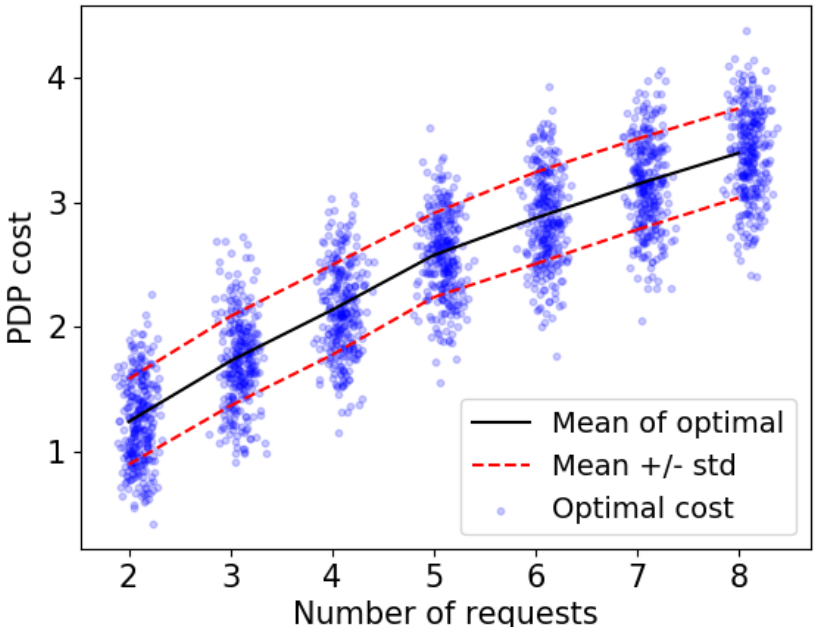


Figure 7-8: Plot of the experiments performed on several PDP instances. You can notice the black line connecting the averages for different n.

We empirically tried to demonstrate that the length of an optimal tour of a PDP problem with n requests is equal in average to:

$$\bar{Y}_n = \alpha\sqrt{n} + \beta \tag{7-13}$$

for all n. We thus fitted the curve with the input data:

$$X = \left[\underbrace{2,\dots,2}_{m_{val}\text{ times}}, \underbrace{3,\dots,3}_{m_{val}\text{ times}}, \dots, \underbrace{8,\dots,8}_{m_{val}\text{ times}}\right],$$

$$Y = \left[\mathcal{F}_1(R_1^2),\dots,\mathcal{F}_{m_{val}}(R_{m_{val}}^2),\dots,\mathcal{F}_1(R_1^8),\dots,\mathcal{F}_{m_{val}}(R_{m_{val}}^8)\right],$$

where the function $\mathcal{F}_i(R_i^j)$ is, with some abuse of notation, the function of Eq. (4-10), which return the optimum value of the PDP i-th training example with j requests. The resulting parameters were:

$$\alpha = 1.54, \quad \beta = -0.9254 \tag{7-14}$$

Figure 7-9 shows the performance of the fitting. Such result will be useful further in the thesis.

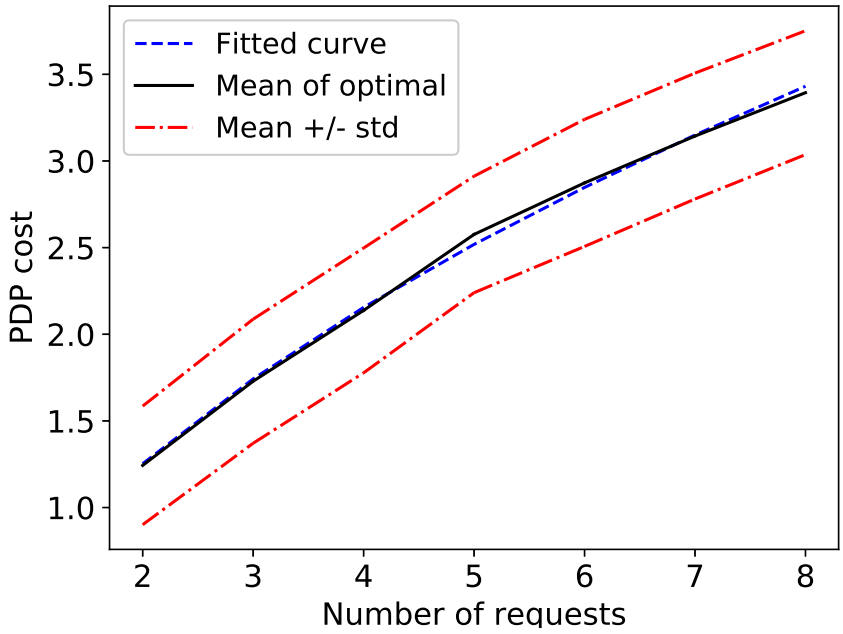


Figure 7-9: Fitting of the function $\bar{Y}_n = \alpha\sqrt{n} + \beta$ over the line connecting the averages of optimal PDP costs for different n.

7-5-2 Coordinates as input features

Having a PDP with two requests means that there are in total four points to connect, where each point has two coordinates:

$$r_0 = (x_0, y_0), \quad r_1 = (x_1, y_1), \quad r_2 = (x_2, y_2), \quad r_3 = (x_3, y_3).$$

Hence, considering the PDP I-th instance, the argument of the machine learning model applied to predict the optimal length $v(I)$ is a vector $\varphi(R_I) \in \mathbb{R}^8$ containing the coordinates of the requests r belonging to the i-th set of requests R_I:

$$\varphi(R_I) = \begin{bmatrix} x_0^I & y_0^I & x_1^I & y_1^I & \dots & x_3^I & y_3^I \end{bmatrix},$$

and the training input is a matrix $X \in \mathbb{R}^{m \times 8}$ containing all the points coordinates:

$$X = \begin{bmatrix} \varphi(R_1) \\ \varphi(R_2) \\ \vdots \\ \varphi(R_m) \end{bmatrix}.$$

Table 7-8: Pick-up and Delivery Problem: performance of Neural Networks.

Activation Function	Error	Neurons	Training time (s)	Input features
ReLu	7.36×10^{-2}	99	2.74	(x, y) coordinates
ReLu	7.03×10^{-2}	143	3.08	(x, y) coordinates
ReLu	6.64×10^{-2}	$[109, 57]$	6.88	(x, y) coordinates

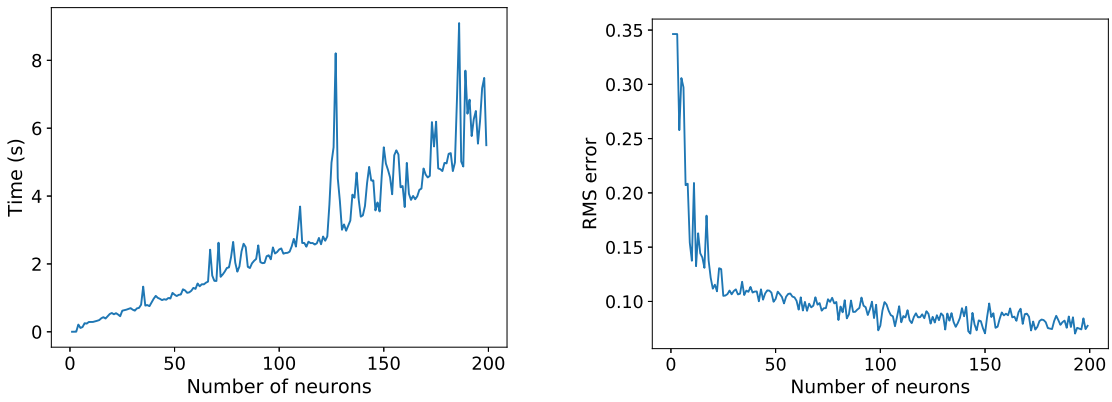
When the input features are the coordinates of the nodes in the PDP, we will refer to the generic input vector $\varphi_C \in \mathbb{R}^{2p}$ as:

$$\varphi_C(R_i) = \begin{bmatrix} \mathbf{X} & \mathbf{Y} \end{bmatrix} = \begin{bmatrix} x_1^i & y_1^i, & \dots, & x_p^i, & y_p^i \end{bmatrix}, \tag{7-15}$$

As can be easily deduced, the scalability is negatively affected: increasing the number of requests will lead to making the input vector four times larger. Therefore, other features input will be further investigated in order to show that another reliable prediction can be still obtained with a more scalable and yet faster model. The training output vector, that is the target $Y \in \mathbb{R}^m$ is the following:

$$Y = \begin{bmatrix} \mathcal{F}(R_1) \\ \mathcal{F}(R_2) \\ \vdots \\ \mathcal{F}(R_m) \end{bmatrix},$$

where $\mathcal{F}(R_i)$ is the function shown in Eq. (4-10), which returns the optimal length of the i-th PDP training example whose set of requests if R_i.



(a) Training time. **(b)** RMSE.

Figure 7-10: Analysis of the training time and the Root Mean Squared Error with respect to the number of neurons in a 2-requests PDP instance.

With the same approach described in the previous section, the performance of several machine learning models will be shown with respect to the validation error, complexity, and training time. In Table 7-8 such performance can be observed. From this Table, it emerges clearly that, compared with the Euclidean distance, the accuracy of the prediction worsens by a factor of ten. Furthermore, the training time is much slower compared to the Euclidean distance case;

moreover, increasing the complexity of the network by adding neurons or additional hidden layers does not improve the performance considerably. This can be observed in Figure 7-10. In particular, considering Figure 7-10a, one can notice that the training time increases almost linearly with the number of neurons. The spikes should be addressed to unpredictable routines which slow down the process. On the other hand, in Figure 7-10b it is shown that the RMSE saturates; therefore, after such saturation, it becomes superfluous to still increase the number of neurons. Another aspect that arises from the Table compared to the previous case is the absence of models with sigmoid activation function. As can be seen in Figure 7-11, when the activation is the sigmoid function, the prediction settles on the average of the PDP cost. This happens because the final goal of NN is to reduce the average error. The reason for such behavior is related to the shape of the activation function and to the optimization method. Recalling the Equation Eq. (2-8), the derivative of the sigmoid activation function is:

$$\sigma'(z) = \frac{e^{-z}}{(1+e^{-z})^2} = \sigma(z) \cdot (1 - \sigma(z)) \tag{7-16}$$

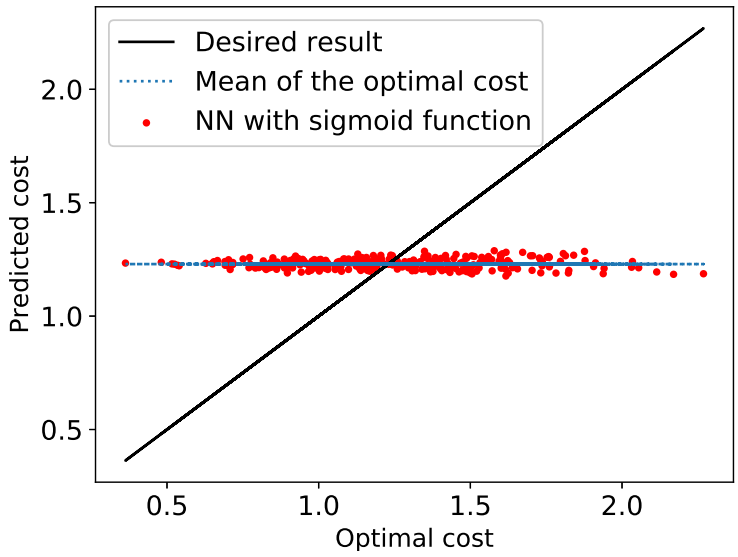


Figure 7-11: Performance of the model trained using sigmoid activation function in the 2-requests PDP with coordinates as input features. The scatter of the models lies below the mean of the optimal validation cost (the dotted line).

In Figure 7-12 is shown the derivative of the sigmoid activation function. Since such derivative is bounded by 0.25, it can happen very easily that the gradients of all the neurons are small if the algorithm results stuck in a local minimum. Hence, the product of such bounded values reaches zero rapidly, due to the vanishing gradient problem. But this means that the NN must train a model able to quickly leads to an averagely small error. The only way such goal can be achieved is by generalizing the model in such a way it predicts the mean of the training output. Therefore, with coordinates as input features, the sigmoid activation function is discarded from the options.

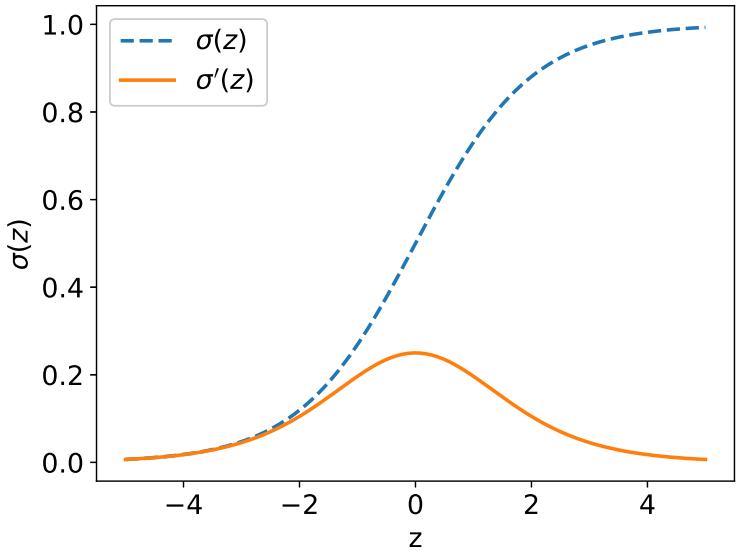


Figure 7-12: Plot of the sigmoid function with its derivative.

Regarding the SVR, from Table 7-9 is it possible to see how hard is finding an hyper-plane able to separate the different cases. In addition, the training time is considerably slow and therefore such model is not very suitable.

7-5-3 Heuristic as input features

In Chapter 5 we mentioned some possible additional input features for machine learning models solving the TSP and Vehicle Routing Problem (VRP). In this section the result of the greedy heuristic applied to the Euclidean PDP is used as input to the supervised learning model. The concept is explained in Figure 7-13. The input features vector for the supervised learning model with m training examples is:

$$X = \begin{bmatrix} G_1 \\ G_2 \\ \vdots \\ G_m \end{bmatrix} \tag{7-17}$$

where G_i is the function Eq. (2-4) computing the greedy route of the PDP similarly to Algorithm 1. The only difference in the algorithm is that now there are precedence constraints

Table 7-9: PDP: performance of Support Vector Regression.

C	Error	Training time (s)	Input features
99	1.33×10^{-1}	6.88	(x, y) coordinates
299	1.16×10^{-1}	7.34	(x, y) coordinates

to satisfy and therefore destination nodes cannot be inserted in the route before the corresponding origin nodes.

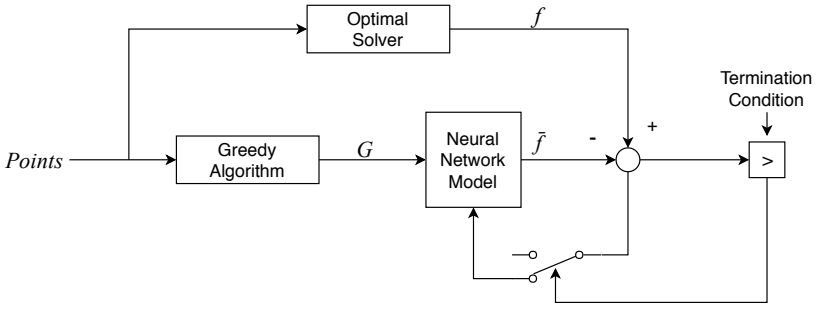


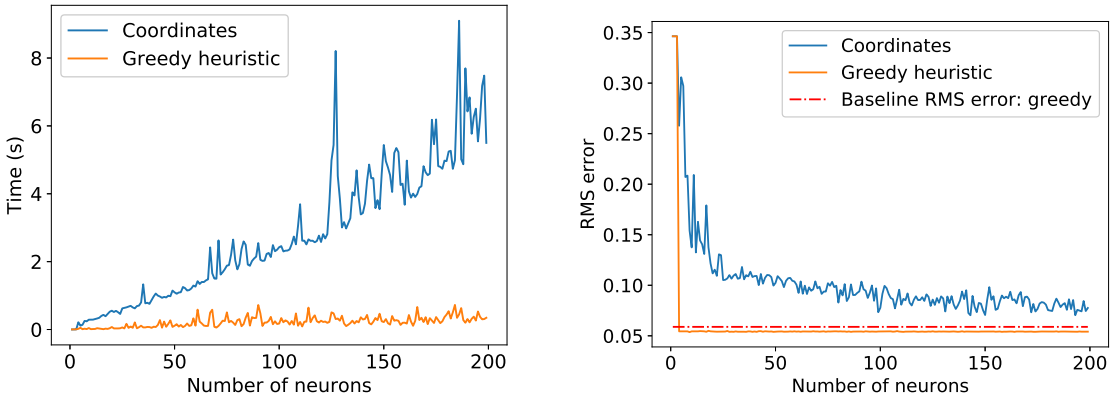
Figure 7-13: This Figure describes the training procedure when the input is the greedy heuristic. Given m experiments, in each experiment there are four points generated randomly. Imagine storing all the coordinates in a matrix called $Points$, where each row contains the coordinates of the four points. Therefore, for each row, we computed the greedy solution G_i and the optimal solution f_i and stored them in the input and output vectors. Until a termination condition is not satisfied, the learning algorithm adjusts the weights to overall minimize the error.

Table 7-10, ordered in training time ascent order, summarizes the performances. Even the slower models of the type $\bar{f}(G)$ outperform in terms of validation error the faster models trained with coordinates as input features; in addition, the structure of the network is much simpler. As in Figure 7-10, the same analysis can be applied to this case and the results are shown in Figure 7-14. Increasing the number of neurons does not affect the model trained with the greedy heuristic (see Figure 7-14a); the reason is that the optimization algorithm used to adjust the weights of the net achieves the sub-optimal result very soon and therefore the training time is considerably reduced. Furthermore, in Figure 7-14b it can be observed that the saturation of the RMSE is reached long before in case of greedy heuristic as input. In the same Figure, a baseline has been added in order to appreciate the performance of the models (the red dashed-dotted line). Such baseline is the validation error between the greedy heuristic and the optimal cost. We evaluate how the machine learning improves the baseline solution using the greedy algorithm. The baseline is $B_{greedy} = 5.89 \times 10^{-2}$, which is outperformed by all models in Table 7-10. It is worth noticing that the model with the greatest error still outperforms the baseline, improving it by 6%; therefore, we expect that the benefits would be much more relevant in a PDP with more requests.

Table 7-10: PDP: performance of machine learning models with other input features. For the NN, h stands for the number of hidden neurons. For the SVR, C is the penalty term associated to the error.

Model	Error	Hyper-parameter	Training time (s)	Input features
NN with ReLu	5.3×10^{-2}	$h = 9$	0.02	greedy heuristic
NN with ReLu	5.2×10^{-2}	$h = [1, 9]$	0.03	greedy heuristic
NN with Logistic	5.3×10^{-2}	$h = 29$	0.09	greedy heuristic
SVR	5.46×10^{-2}	$C = 0.18$	0.19	greedy heuristic
SVR	5.55×10^{-2}	$C = 8$	0.34	greedy heuristic

Regarding the SVR, while on one hand the models outperform their versions with coordinates as input features, the performance is poor compared to the NN. Therefore, from now on SVR models will be neglected. The models trained instead with the result of greedy heuristic as input is a one-to-one mapping. Hence, such approach is more promising for the scalability: the input is mono-dimensional.



(a) Training time. **(b)** RMSE.

Figure 7-14: Analysis of the NN models with different input features.

7-5-4 Comparisons

The performance of the trained models will be qualitatively evaluated in this subsection. We will compare the model trained with coordinates as input features and the model trained with the greedy heuristic. In both cases, the model is a NN with ReLu activation function, as summarized in Table 7-11. Moreover, both models are also compared with a baseline, that is the length of the route estimated with a greedy heuristic algorithm.

The performances are evaluated using a set of validation experiments. Each experiment consists of a batch of $2n = 4$ points generated randomly within a Gaussian distribution. The coordinates are in the range $[0, 1]$. There are 4 computed costs of the related PDP:

- The optimal cost computed using the Optimization formulation of Eq. (6-11), that is the value of the function $\mathcal{F}(R)$ (see Eq. (4-8)).
- The heuristic baseline cost $G(\mathbf{P})$ computed with the greedy algorithm (see Chapter 2 and Algorithm 1).
- The cost predicted with the learnt model $\bar{f}(\varphi_C)$ which has as input the coordinates of the $2n$ points, $\varphi_C = [\mathbf{X}, \mathbf{Y}]$.
- The cost predicted with the learnt model $\bar{f}(\varphi_G)$ whose input is the result of the greedy algorithm applied to the batch of points, $\varphi_G = G(\mathbf{P})$.

Table 7-11: Performance of the NN models with ReLu activation function.

Input Features	Error	Neurons	Training time (s)	Figure
Greedy heuristic	5.33×10^{-2}	9	0.2	Figure 7-15
(x, y) coordinates	6.64×10^{-2}	$[109, 57]$	6.88	Figure 7-16a

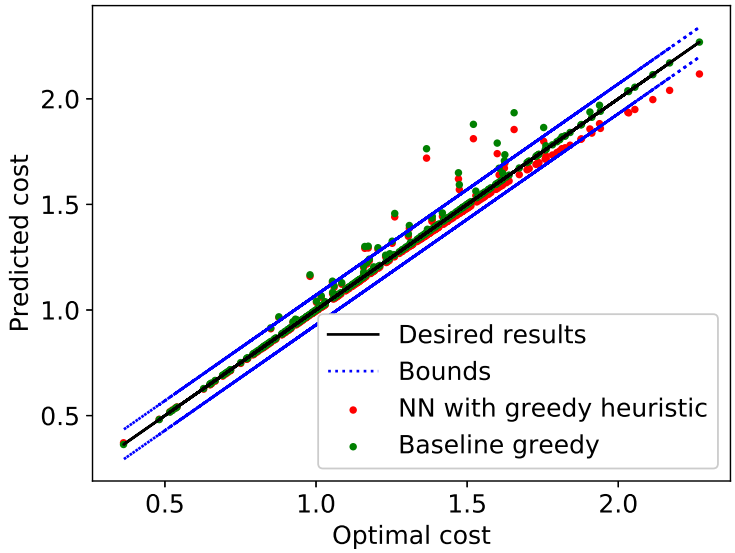
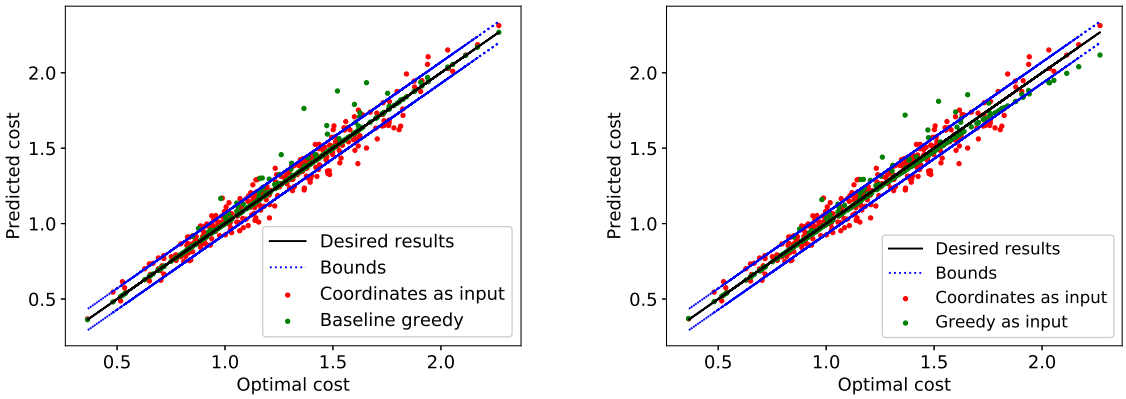


Figure 7-15: Performance of the Neural Network model trained with greedy heuristic as input feature.

The predicted costs are plotted in a 2-dimensional plane as a function of the optimal cost. Hence, in Figure 7-15 and Figure 7-16 the $x-$axis represents the optimal cost, the $y-$axis the predicted cost. We did in this way since we aimed to emphasize how the predictions change over the optimal costs. In each Figure, three lines are drawn to better visualize the models' performance. Firstly, the bisector of the quadrant (in the (x, y)-plane, it is $y = x$): when the prediction is accurate, the scattered points lie on such line. Clearly, points above that line are an over-approximation of the optimal cost, while points below under-approximated it. Since it is very unlikely to train a model in such a way that it predicts exactly the optimal cost, one should have a confidence bound in which the predicted cost is expected to lie in. To academic purpose, such bound is set using the RMSE between the optimal and the predicted costs applied to all the validation data set experiments. Hence, in Figure 7-15 and Figure 7-16 the dotted lines represent the bisector shifted positively and negatively of a factor equal to the RMSE. Model results thus to be more reliable if the result of its prediction is between these bounds. Such bound can be interpreted as the uncertainty associated with the prediction.

In Figure 7-15 can be observed the performance of the models trained with the results of the greedy algorithm as input features. When the greedy solution coincides with the optimal solution, the predicted output drifts away from the bisector for high value of the cost. Such behaviour can be explained with the support of Figure 7-17. In that Figure, it is possible to observe the curve learnt from the model. In the learning process, such curve tries to fit the

function when the over-approximation given by the greedy algorithm is far from the optimal cost. In addition, higher optimal costs are more unlikely to happen. Therefore, during the learning process, there are few examples and larger over-approximations to compensate. A solution could be increasing the number of training examples with higher optimal cost.
From Figure 7-15, one can infer that for a small PDP instance, a heuristic solution provides good results. On the other hand, Figure 7-16 shows that the model trained with greedy heuristic ensures better performance. Furthermore, one expects that the benefits of the learning become more evident in larger instances.



(a) Comparison between the model trained with coordinates as input features and the baseline.

(b) Comparison of the two trained models for different input features.

Figure 7-16: Analysis of the Neural Network models for the PDP with two requests.

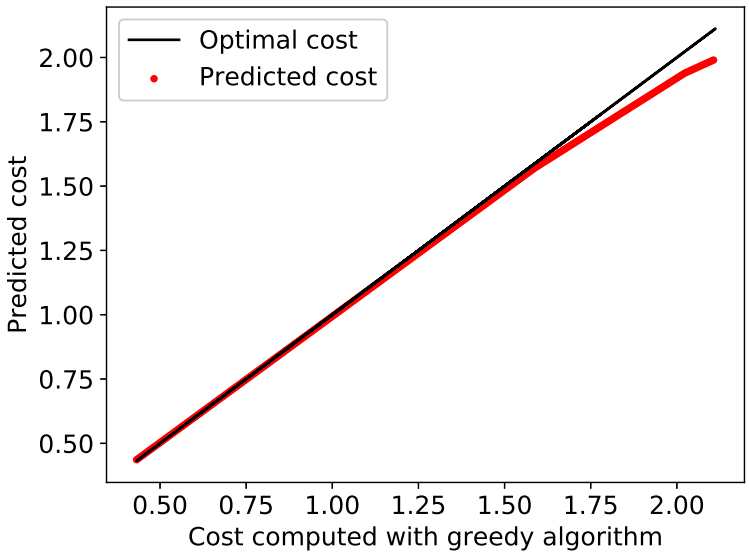


Figure 7-17: Output of the model training with the greedy heuristic algorithm plotted as function of the greedy cost.

7-5-5 Analysis of the models

To help visualizing the models performance, a higher-dimensional plot can be useful, like in section 7-2 (see also Figure 7-6). However, the PDP problem with 2 requests depends on 8 parameters. Therefore, a solution could be to maintaining 3 of the 4 points fixed and observe how the cost varies when the 4-th free point moves on the plane. Such experiment is carried out to interpret the ability of trained models to predict the optimal cost. Since the smallest error was achieved with the greedy heuristic input, we aim to show the accuracy only of the model trained with that input features. Some example are in Figure 7-18. The 3D representation plots are of the left, while the contour plots are on the right. The shape of the function, of course, depends on the fixed points. The slider in these Figures indicates different positions of three fixed points.

From these Figures, we inferred that the NN model has good prediction performance except when the function abruptly changes, or when the free point is farther. This could be since it is an unlikely situation and it is probably that there were not enough training examples in such sense. From what we experienced, NN has better performance to predict very highly non-linear non-convex model, especially when there are not any prior information. Moreover, there are the time-related issues to take into account: training a SVR model is slower compared to NN and this is a great disadvantage when projecting the learning in larger dimensional scenarios.

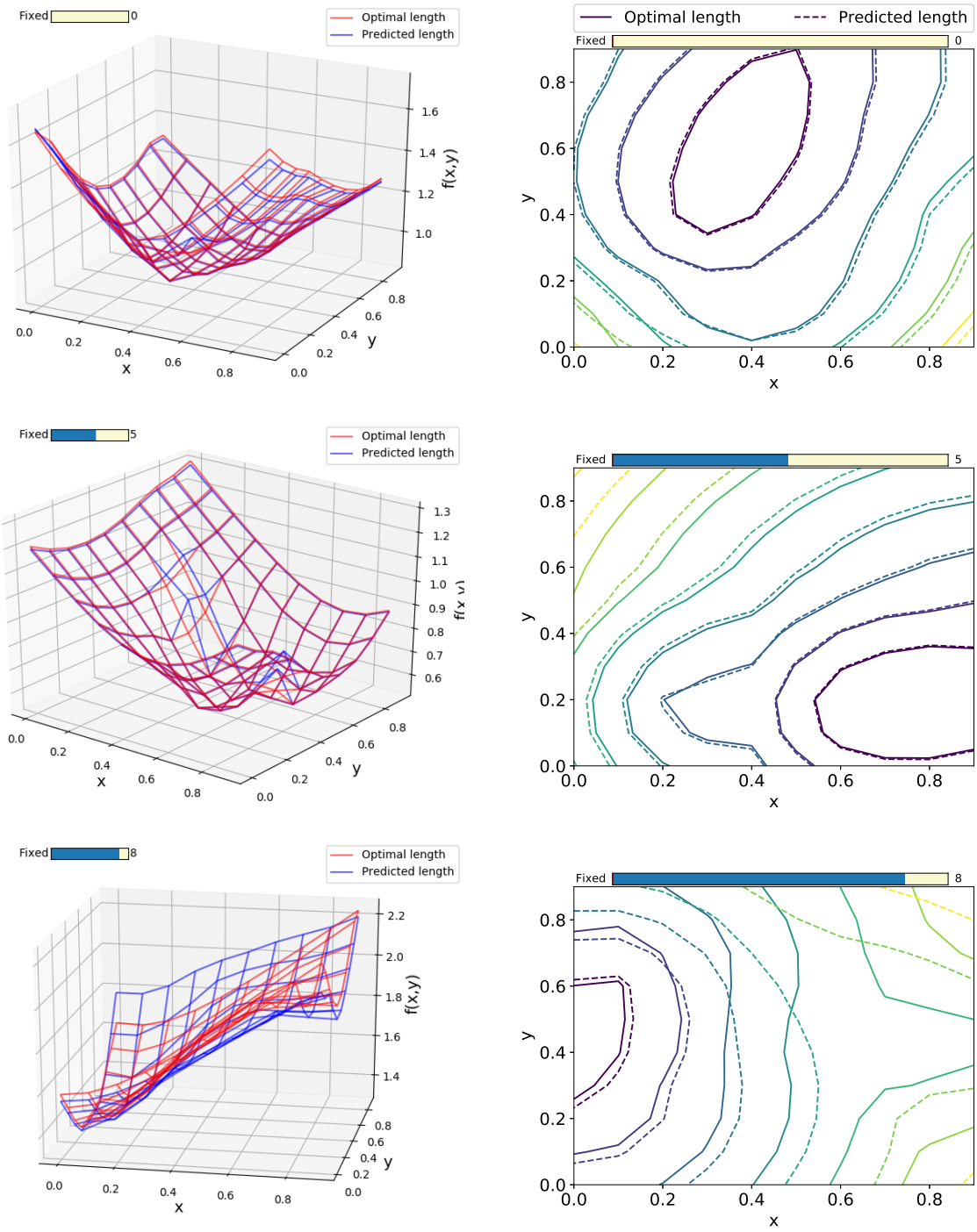


Figure 7-18: 3D plots and contour plots for different 3-fixed points. In the right column, the solid line is the optimum of the associated PDP, the dashed line is the predicted length using NN model trained with greedy heuristic. The slider indicates a different configuration of the three fixed points.

Chapter 8

Tests, simulations and results

In this chapter, we analyze the performance of supervised learning to predict the optimal cost in larger instances of Pick-up and Delivery Problem (PDP), following the same approach as the previous chapter. We will use the results we obtained to do an educated guess about the design of the network.

8-1 Experiments on larger-size

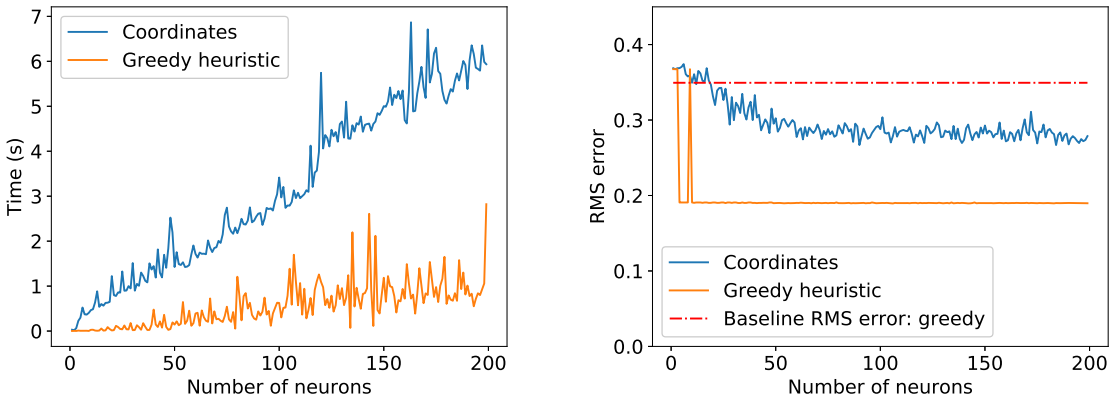
In this section, we consider the PDP with more than two requests. Our goal is showing how this increase will be detrimental to the performance of the network trained in the previous chapter. We chose six requests, leading to PDP training examples still solvable by the optimization algorithm in a reasonable time. Thus, there are $n = 6$ requests and $2n = 12$ points to connect. As in the previous chapter, the performance of trained models are compared with the result of greedy heuristic, showing the benefits of supervised learning over such baseline.

8-1-1 Euclidean distance

Consider the predictor $\bar{f}(\varphi_C)$, where $\varphi_C = [\mathbf{X}, \mathbf{Y}]$. In a PDP with six requests, the input is thus $\varphi_C \in \mathbb{R}^{24}$. In the previous chapter we saw that a single-layer Neural Network (NN) model achieves the smallest validation error with 143 neurons (see Table 7-8). Since the complexity increases, we expect a greater number of neurons to obtain a small error. On the other hand, consider now the predictor $\bar{f}(\varphi_G)$, where the input is the result of the greedy heuristic algorithm, $\varphi_G = G(\mathbf{P})$; such input feature remains a scalar as in the previous chapter even if the number of requests increases; therefore the complexity of the network is expected to remain the same.

Figure 8-1 shows the relations between neurons, training time and Root Mean Squared Error (RMSE) during the training of model able to predict the optimal cost of a PDP with six requests. Let us analyze separately the greedy heuristic value as input features and the

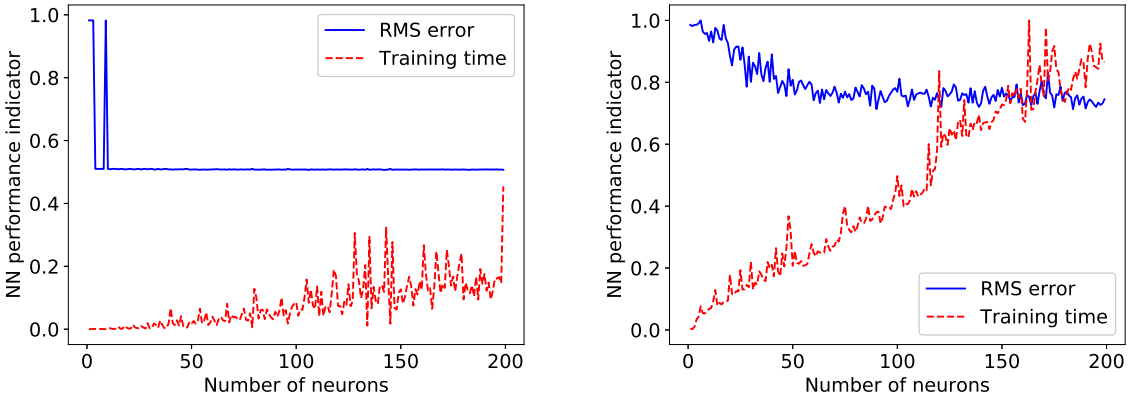
coordinates as input features. Firstly, consider the training time; when the input is φ_C, the training time increases almost linearly with the number of neurons (see Figure 8-1a). The RMSE saturated instead after 80 neurons (see Figure 8-1b). The benefits of the supervised learning are more evident compared to the two-requests PDP: NN models here improve substantially the greedy heuristic baseline. However, it is unnecessary to design a higher number of neurons, since the error does not show any significant improvement after 80 neurons. Training time, on the other hand, would be negatively affected.



(a) Training time. **(b)** RMSE.

Figure 8-1: PDP with $n = 6$ requests: analysis of the neural network models with different input features.

Models trained with greedy heuristic maintain the performance highlighted in the last chapter. The error reaches low values with small network structure and the training time remains low compared to the coordinate input features case. Table 8-1 summarizes the models per-



(a) Greedy heuristic as input feature. **(b)** Coordinates as input feature.

Figure 8-2: Comparison between the models trained with different input features and the greedy baseline. Both training time and RMS error have been scaled with respect to their maximum. The y-label indicates the performance of the network: the higher, the worse.

formances. In the following experiments, we fixed the number of neurons equal to $h = 90$

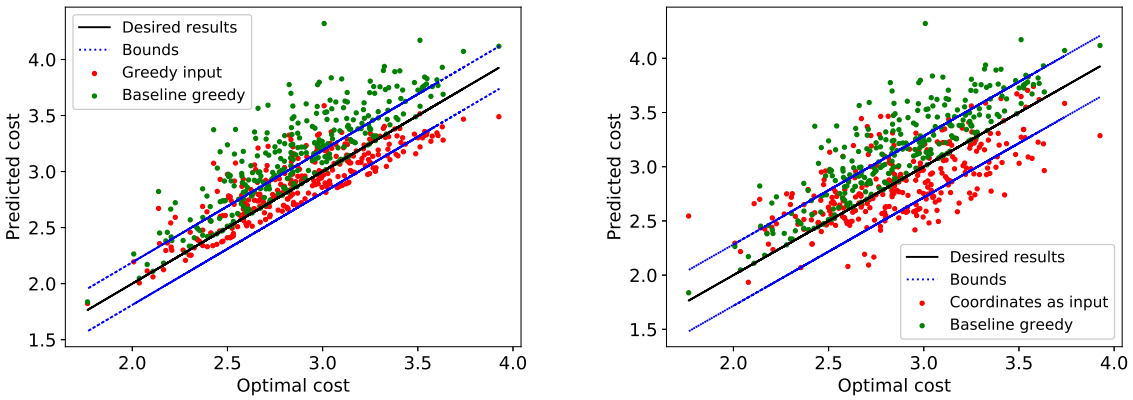
Table 8-1: Performance of the NN models. The fifth column underlines the improvement in predicting the cost compared to greedy heuristic baseline.

Input Features	Error	Neurons	Training time (s)	Improved accuracy	Figure
Greedy heuristic	0.189	106	0.59	40.6%	-
(x, y) coordinates	0.2668	184	5.49	23.6%	Figure 8-3b
(x, y) coordinates	0.267	90	2.42	23.6%	-
Greedy heuristic	0.189	90	0.43	40.6%	Figure 8-3a

Table 8-2: $\mathcal{L}_i$-norm: performance of the NN models. The fifth column underlines the improvement in predicting the cost compared to greedy heuristic baseline.

Input Features	Error	Neurons	Training time (s)	Improved accuracy	Figure
Greedy heuristic	0.19	64	0.14	40%	Figure 8-4a
(x, y) coordinates	0.26	89	2.47	19%	Figure 8-4b

for all models. Such choice is a trade-off for the network trained with coordinates, while it is irrelevant for the models trained with the greedy. Figure 8-2 helps to understand this concept: we scaled both RMSE and training time with respect to their maximum. Therefore, while Figure 8-2a shows that the minimum RMSE can be achieved with a small network, in Figure 8-2b the minimum is reached after the intersection of the two functions.
A graphical representation of the models performances is in Figure 8-3. Although the uncertainty range of Figure 8-3a is smaller, there are less outliers than in Figure 8-3b. This confirms our initial guess that such kind of model satisfies all the requirements for a good combinatorial optimization result predictor: scalability, reliability, fast training time. As a consequence, from now on we discard the coordinates as potential input features.



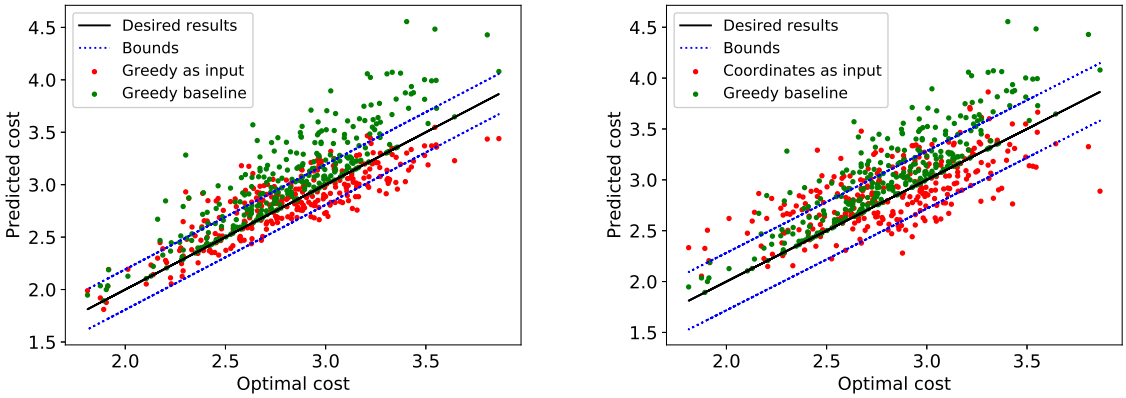
(a) Greedy heuristic as input feature.

(b) Coordinates as input feature.

Figure 8-3: Comparison between the models trained with different input features and the greedy baseline.

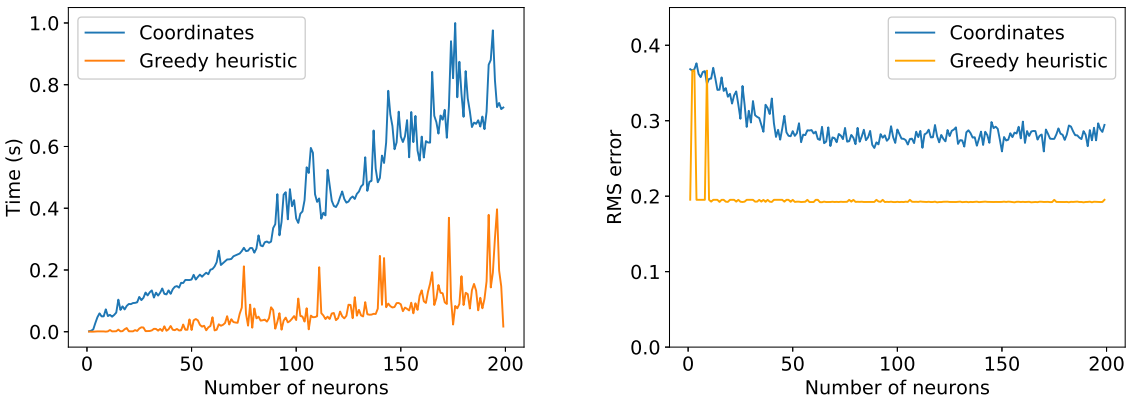
8-1-2 $\mathcal{L}_1$-distance

The approach of the previous subsection can be applied in the same way with a different distance measure, i.e. the $\mathcal{L}_1$-distance. Table 8-2 shows that with such measure the improved accuracy remains the same when the greedy heuristic is the input, while it improves with coordinates. The training time is smaller and the net has a smaller number of neurons since the function to approximate has fewer non-linearities. Therefore, using $\mathcal{L}_1$-distance makes the learning process faster, but cannot be seen any evident improvements in the prediction accuracy. The overall performance of NN models listed in Table 8-2 can be observed in Figure 8-4. As expected, the networks maintain a good level of approximation within the bound in Figure 8-4a. In addition, Figure 8-5 shows the same behavior in the relation Neurons-RMSE and Neurons-Training time as the Euclidean distance case (see Figure 8-1).



(a) Greedy heuristic as input feature. **(b)** Coordinates as input feature.

Figure 8-4: $\mathcal{L}_i$-norm: comparison between the models trained with different input features and the greedy baseline.



(a) Training time. **(b)** RMSE.

Figure 8-5: PDP with $n = 6$ requests: analysis of the neural network models with different input features when the distance measure is the $\mathcal{L}_1$ distance.

8-2 Analysis of the approximation accuracy

Following the same approach of subsection 7-5-5, we draw 3D plots to graphically visualize how the model trained with greedy heuristic as input feature is able to approximate the function $\mathcal{F}(R)$ shown in Eq. (4-10). Therefore, we kept fixed 5 out of 6 points leaving the sixth free to move and then we computed the optimal and predicted cost. Figure 8-6 shows the approximation accuracy. Since the validation error is three orders of magnitude bigger than the 2-requests PDP, the worsening was expected (see Figure 7-18 as a comparison). Also in this case, the accuracy depends on the sparsity of the fixed points. In the Figures are visible the multiple local minima which make the combinatorial optimization problem hard to solve, even in this simplified configuration.

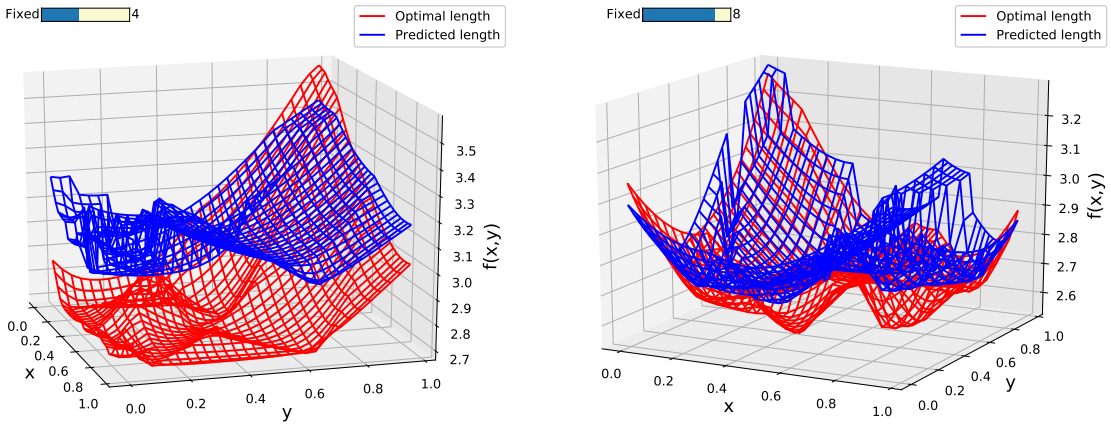


Figure 8-6: 3D plots for different configuration of the 5-fixed points. The slider on the top indicates different distribution of the fixed points.

8-3 Additional input features

In this section, we will investigate if other input features improve the prediction accuracy of the network we trained.

8-3-1 Lower bound

Chapter 2 referred to a lower bound approximation for the Travelling Salesman Problem (TSP), that is the Held-Karp lower bound (HK) [29, 30, 31]. We hereby propose a faster lower bound estimation.

The solution of any TSP instance with n cities is nothing but the sum of $n-1$ entries in the Adjacency matrix (see Chapter 6). As stated before, the adjacency matrix is symmetric and the diagonal is filled with zeros. Therefore, the number of entries to consider is $\frac{n(n-1)}{2}$. Imagine now to store all these entries in an array, shown below:

a_{11}	a_{12}	$\cdots$	a_{1n}	a_{22}	$\cdots$	$a_{n,n}$

Table 8-3: Sorting algorithms: time complexity.

	Best Case	Average Case	Worst Case
Bubble Sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Insertion Sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Merge Sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
Quick Sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$

which is a vector listing all the edges. Imagine now to sort the array in ascending order. The sum of the first $n-1$ elements will be the lower bound of the TSP. The computational effort depends only on the chosen sorting algorithm [70], as shown in Table 8-3. To summarize, the algorithm estimate the lower bound is the following:

Algorithm 2: Compute the TSP lower bound

```
// Input: Symmetric Adjacency matrix A
// Output: The lower bound L
// Initialization: E = [], i = 0, l = 0, c = 0
// Compute the upper-triangular part of the Adjacency matrix and store
   those entries in an array E
for i≤n-1 do
    j←i+1;
    for j≤n do
        E[l]← A[i,j];
        l←l+1;
    end
end
// Sort the array
E←mergeSort(E);
// Sum the first n − 1 entries
L← ∑_{i=0}^{n−1} E[i];
return L;
```

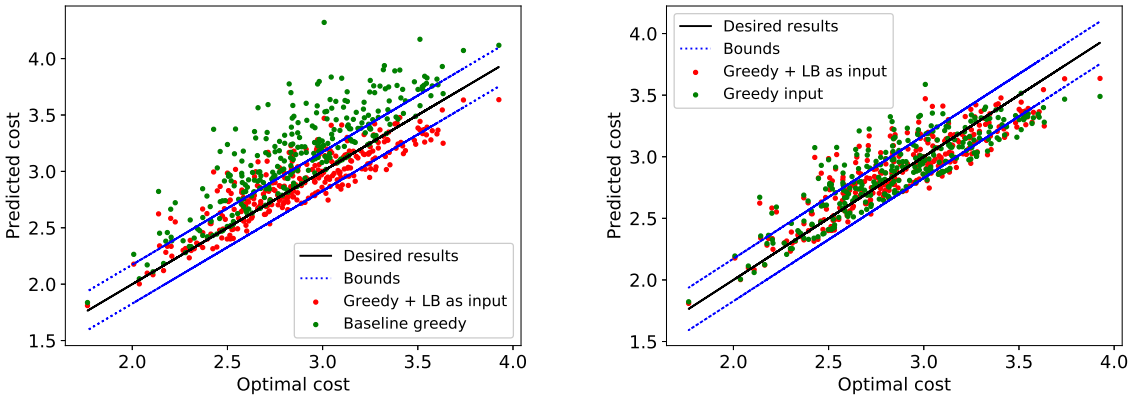
Similarly, the approach can be equally applied to estimate the lower bound of a PDP with n requests. We call E the sorted vector of the feasible edges, with cardinality $2n(2n-1)/2$, that is $n(2n-1)$. The number of edges in a route solving the PDP is $2n-1$. Furthermore, we called L_i the lower bound of the $i-$th instance, that is the sum of the first $2n-1$ feasible edges:

$$L_i = \sum_{j=0}^{2n-1} E[j] \tag{8-1}$$

The input features vector for the supervised learning with m training examples is now:

$$X = \begin{bmatrix} G_1 & L_1 \\ G_2 & L_2 \\ \vdots & \vdots \\ G_m & L_m \end{bmatrix} \tag{8-2}$$

The experiments are shown in the following Figures. We maintained the same structure of NN (a single layer with 90 hidden neurons) to predict the optimal length of a PDP with six requests. Figure 8-7 shows the effect of the additional input to the prediction accuracy. We want to emphasize, looking at Figure 8-7a, how the model with also the lower bound as input tends to predict cost closer to the black line, which is the line of optimality.



(a) Comparison with the baseline.

(b) Comparison with the model trained only with greedy heuristic.

Figure 8-7: Lower bound as additional input features to the model predicting the PDP outcome with $n = 6$ requests. The left Figure shows the comparison with the greedy heuristic baseline. The right Figure compares the model trained with only the greedy heuristic and the model with also the lower bound.

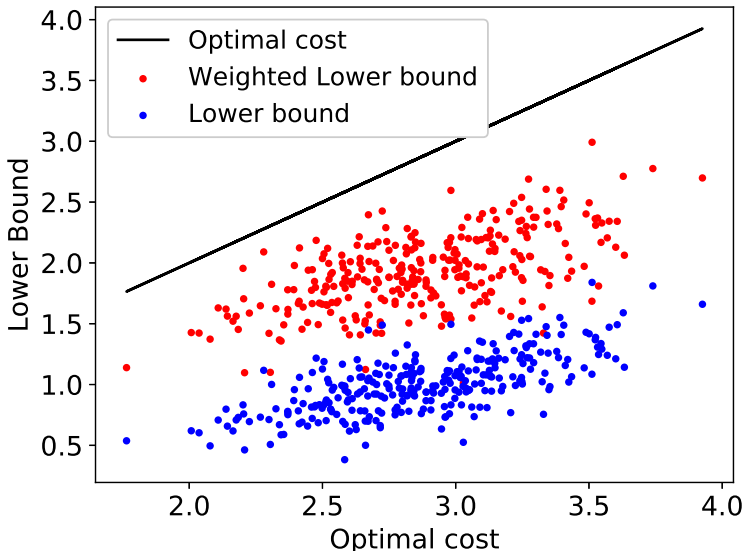


Figure 8-8: Plot of L and $\bar{L}$. Both values lie always below the optimal cost.

Even more interesting is when the lower bound is weighted with the mean $\bar{E}$ of the edges

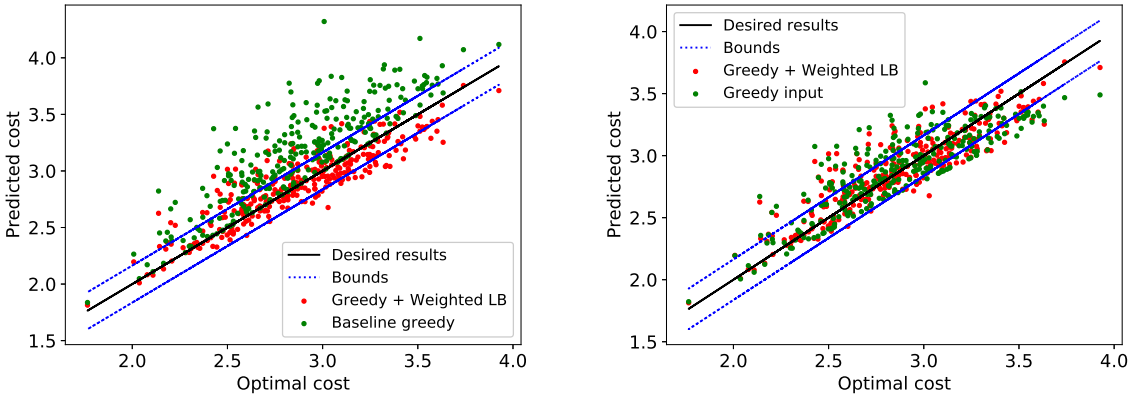
Table 8-4: Performance of the NN models with additional input. The fourth column underlines the improvement in predicting the cost compared to the model trained with only greedy heuristic.

Input Features	Error	Training time (s)	Improved accuracy	Figure
G, L	0.1725	1.24	8%	Figure 8-7b
$G, \bar{L}$	0.1639	1.17	13%	Figure 8-9b

vector E:

$$\bar{L}_i = L_i \cdot \bar{E}_i. \tag{8-3}$$

The weight is a measure to correct possible misleading measures of the lower bound. In fact, it could happen that there are $2n-1$ points very close and therefore the estimation of the lower bound would be too small. But if it is adjusted by a factor equal to the mean of the edges, we can provide a more reliable measure of the edges. Figure 8-8 emphasizes this concept. With such input, the performance slightly improves, as reported in Table 8-4. In addition, the reader can notice in Figure 8-7b how the number of outlier reduces compared to Figure 8-7b.



(a) Comparison with the baseline.

(b) Effect of weighted lower bound on the prediction accuracy.

Figure 8-9: Weighted lower bound as additional input features to the model predicting the PDP outcome with $n = 6$ requests. The left Figure shows the comparison with the greedy heuristic baseline. The right Figure compares the model trained with only the greedy heuristic and the model with the additional input.

8-3-2 Upper Bound

During the thesis, we often underlined that the greedy is one of the possible heuristic methods to find a solution to the routing problem. We also stressed out that there exists other heuristics with better accuracy performance. In this subsection, we aim to show how better heuristics can improve also the prediction accuracy of learnt models.
So far, we used greedy heuristic as a baseline to investigate how supervised learning improves the prediction of the cost. Moreover, the greedy result can be used as initialization for the

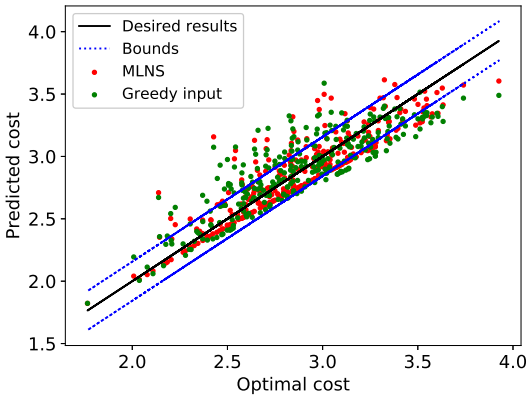
optimization formulation [3]. Furthermore, Chapter 2 referred to other heuristic procedures. Imagine defining thus another heuristic: starting from the greedy solution of the PDP, remove q nodes from the route. Then, iteratively reconstruct the route by adding in place of the removed nodes one of the possible permutations of such q nodes. This procedure is repeated until a time limit expires. For each reconstructed route, compute the length of the path and compare it with the shortest path computed up to that. If the former is smaller then the latter, update the shortest path. We will refer to such heuristic as Modified Local Neighborhood Search (MLNS). The pseudo-code is listed in Algorithm 3.

Algorithm 3: Compute the MLNS of PDP.

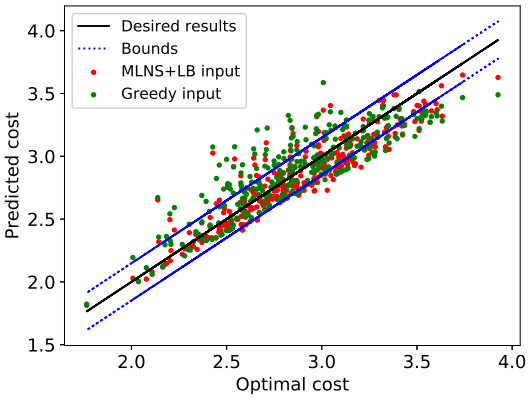
```
// Input: Matrix storing the points coordinates P, the number of nodes to
   remove q, the time limit T. The matrix P is in the form P ∈ R^{n×4}:
   each row corresponds to a single requests, thus comprising two points,
   origin and destination.
// Output: The heuristic cost shortest and the path path
// Initialization: path = [], i = 0, j = 0, c = 0

n ← len(P) // Compute the length of the P matrix. It return the size of
   the PDP, that is the number of requests.
shortest, route ← greedy(P);
newRoute ← route[0:q] // remove q elements from the route and store the
   reduced vector in a new variable
Q← route[q:end] // store the removed nodes into a new vector
perm ← permutation(Q) // generate all the permutation of Q and store them
while t < T do
    // Do until the time limit is not reached
    for p in perm do
        // In each iteration consider one of the permuations stored in perm
        r ← newRoute;
        append(r, p);
        // Reconstruct the route by appending the permutated version of the
           removed q nodes
        if precedenceConstraint(r) then
            // The function precedenceConstraint check if the destinations'
               nodes follow the corresponding origins' nodes
            c ← computeLength(r);
            if c<shortest then
                shortest← c;
                path←p;
            end
        end
    end
end
return shortest, path;
```

Giving the result of the MLNS algorithm as input features improves the prediction accuracy. The new heuristic over-performs (see Table 8-5 and Figure 8-10a) the models trained with



(a) Comparison between the model trained with the Modified Local Neighbourhood Search and model trained with the greedy heuristic.



(b) Comparison between the model trained with the Modified Local Neighbourhood Search and the lower bound and model trained with the greedy heuristic.

Figure 8-10: Performance of the model trained with the Modified Local Neighbourhood Search results as input feature.

Table 8-5: Performance of the NN models with a new input feature. The fourth column underlines the improvement in predicting the cost compared to the model trained with the corresponding model with the greedy heuristic.

Input Features	Error	Training time (s)	Improved accuracy	Figure
MLNS	0.157	0.278	17%	Figure 8-10a
MLNS, $\bar{L}$	0.146	0.09	10%	Figure 8-10b

the greedy heuristic. Adding the lower bound $\bar{L}$ also decrease the estimation error, as shown in Figure 8-10b. The Figure 8-10 refer to PDP with six requests. All the supervised learning models are NN with a single layer and 90 neurons. We set the time limit $T = 0.2s$, while a good choice of q is $q = n + 2$.

8-4 Design of a general model

All the experiments carried out in previous section used customized models trained for the PDP with six requests. The capacity of generalization is thus limited and larger instances would require other models. This means that for l PDP with different l size, l models are required. To avoid such loss of generalization, we aimed to derive a model which takes as input also the number of request n, that is the size of PDP. For instance, the input could be

$$X = \begin{bmatrix} \varphi_1 & \varphi_2 & \cdots & \varphi_l & n \end{bmatrix}$$

with φ_i, $i = \{1, \ldots, l\}$ a general input feature. In this way, a single model can be used for predicting the solution of PDP with different sizes.

The simpler implementation is the following: we trained a model for n going from 2 to 5 giving as input features the greedy heuristic and the number of requests n. For each

Table 8-6: Performance of the $M(n)$ models. The fourth column underlines the variation of $M(n)$ in predicting the cost compared to corresponding model M_6.

Input Features	Error	Training time (s)	Improved accuracy	Figure
MLNS	0.157	4.47	0%	Figure 8-11a
G	0.1949	5.3	-2.51%	Figure 8-11b
MLNS, $\bar{L}$	0.1431	6.8	2.05%	Figure 8-12

$n_i \in \{n_2, n_3, n_4, n_5\} = \{2, 3, 4, 5\}$, the training dataset has dimension $m_n = 3000$. Therefore, the training input is:

$$X = \begin{bmatrix} G_{1n_2} & n_2 \\ G_{2n_2} & n_2 \\ \vdots & n_2 \\ G_{mn_2} & n_2 \\ G_{2n_3} & n_3 \\ \vdots & n_3 \\ G_{mn_3} & n_3 \\ \vdots & \vdots \\ G_{mn_5} & n_5 \end{bmatrix}, \tag{8-4}$$

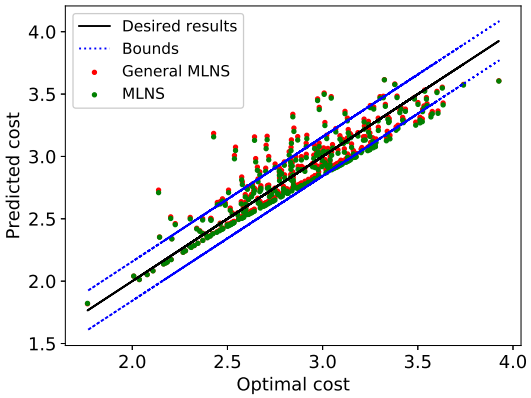
where G_{in_l} is the result of greedy heuristic applied to the training example i with $n = l$ requests. We will refer to the general model as $M(n)$ and to the customized model M_n; the latter is the model trained only for a specific n and thus trained with PDP examples with n requests. Since the input feature dimension increased, we enlarged the size of the network: instead of $h = 90$ neurons, we set the number of hidden neurons to $h = 190$.

8-4-1 Test with six requests

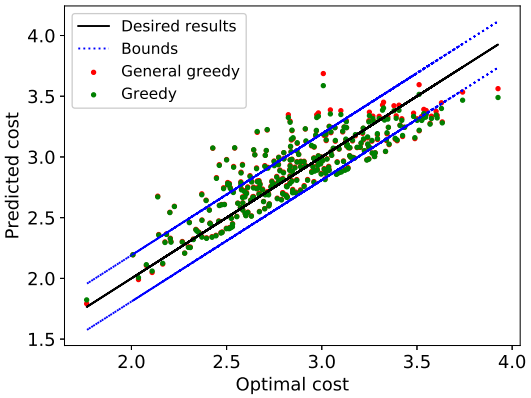
In this section, we will compare how the general model $M(n)$ and the model M_6 in the result prediction of a PDP with six requests. We consider two different scenarios: models trained with MLNS and models trained with the greedy heuristic G.

It is important to mention, as the reader can observe in Table 8-6, that even if the general model predicts the cost of unseen examples as the customized model, it slightly worsen the accuracy when the input is the greedy heuristic, while remains the same when the input is the MLNS. However, the deviation is limited. This means that the model has a good ability to generalize. The third column of such Table indicates a slower training time compared to the training of M_6 (see the previous section), but this result was expected since the size of training input is larger and the number of neurons is almost twice bigger.

In the previous section, we showed that MLNS along with the lower bound ensure the best prediction. We, therefore, tested if such accuracy is ensured in case of a generalized model. From the last row of Table 8-6 and from Figure 8-12 we can appreciate how the generalization does not lead to accuracy loss and instead improves it.



(a) Comparison between the general model and the model trained on 6 requests PDP when the input feature is the result of the MLNS algorithm.



(b) Comparison between general model $M(n)$ and customized model M_6 when the input feature is the result of the greedy algorithm.

Figure 8-11: Comparison between general and customized models.

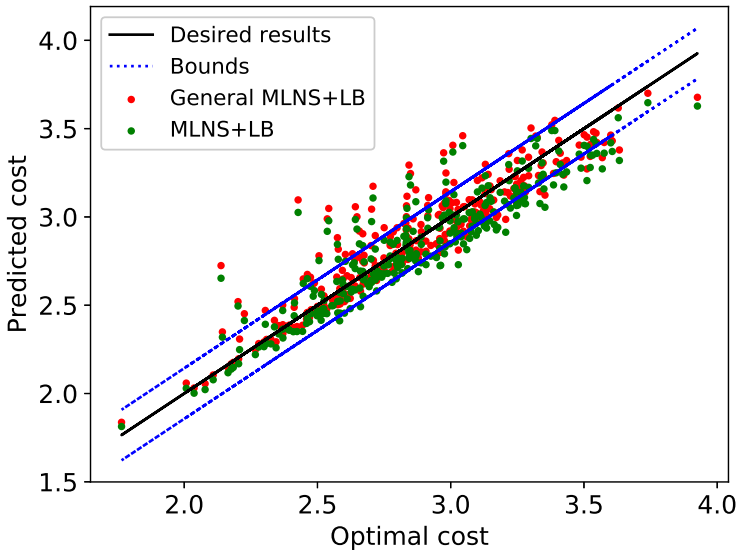


Figure 8-12: Prediction of the cost of a PDP: comparison between general model $M(n)$ and customized model M_6 with the lower bound as input.

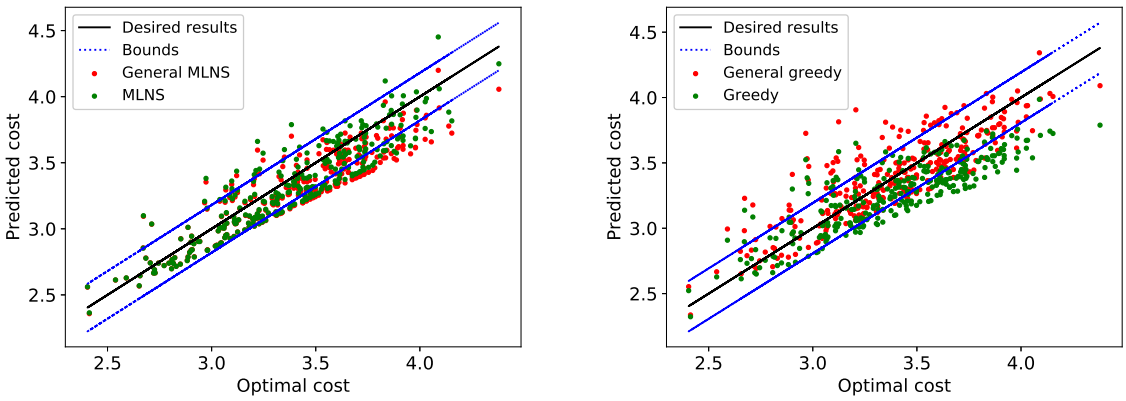
8-4-2 Tests with eight requests

The next step is to apply the generalized models to larger PDP instances. Consider the PDP with $n = 8$ requests and the models $M(n)$ and M_6. Table 8-7 shows that, except for when the input feature is the MLNS, the model $M(n)$ trained on instances with $n = 2, \ldots, 5$ generalizes better than the models trained only on the $6-$ requests instances. We can explain such result since the $8-$requests instance is an example closer to the training set of $n = 6$ and thus the generalization is more accurate when the input is an over-approximation solution which could

Table 8-7: Performance of the $M(n)$ models and M_6 in terms of prediction the cost of PDP with eight requests.

Input Features	Error of $M(n)$	Error of M_6	Figure
MLNS	0.18	0.166	Figure 8-13a
G	0.19	0.23	Figure 8-13b
MLNS, $\bar{L}$	0.1361	0.195	Figure 8-14

be very close to the optimal solution.



(a) MLNS as input feature. **(b)** Greedy heuristic as input feature.

Figure 8-13: Comparison of models generalization capability when the PDP has $n = 8$ requests.

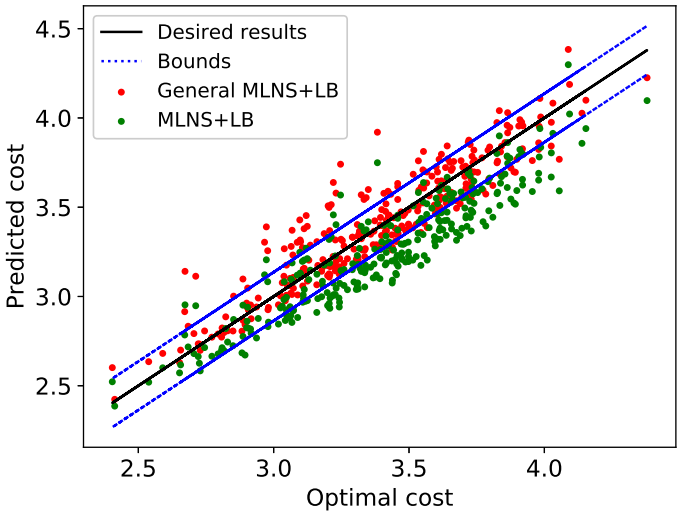
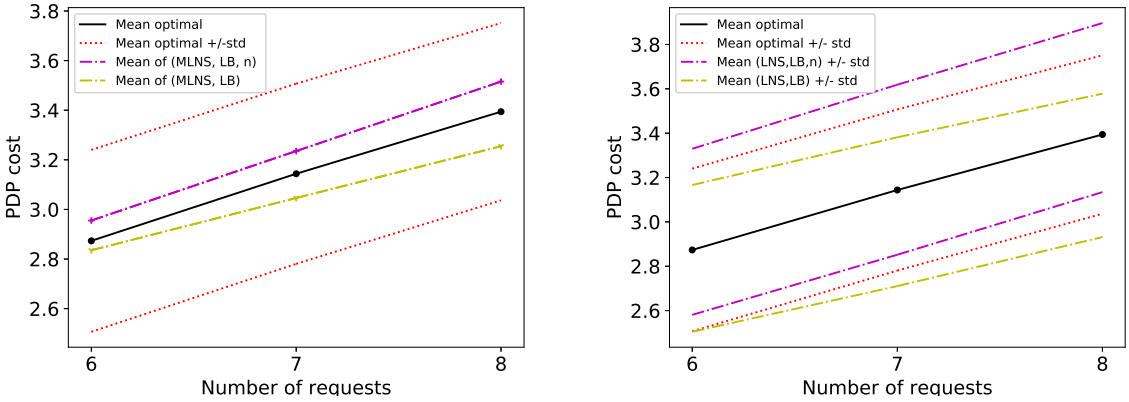


Figure 8-14: Comparison of general and customized model when also the lower bound is part of the input features.

Table 8-8: Performance of the $M(\bar{Y})$ models and M_6 in terms of prediction the cost of PDP with eight requests.

Input Features	Error of $M(\bar{Y})$	Error of M_6	Figure
MLNS	0.18	0.166	Figure 8-13a
G	0.234	0.23	Figure 8-13b
MLNS, $\bar{L}$	0.1373	0.19	Figure 8-14

Despite the positive results we obtained, we wondered about the capability of generalization in greater instances, i.e. number of requests greater than eight. We thus investigated how the two models deviate from the behavior we expected. Given the $m = 300$ validation examples, we computed the average of the optimal cost, the cost predicted with $M(n)$ and the cost predicted with the model M_6. We repeat the procedure for $n = \{6, 7, 8\}$. We noticed that the average of the two models followed a different evolution when the number of requests increased, as can be observed in Figure 8-15. Therefore, we expected that the general model



(a) Averages trajectories.

(b) Standard deviation trajectories.

Figure 8-15: Trajectories of averages and standard deviations. The purple line and the yellow lines connect the average of the trained models for different n.

would lose his generalization capability with the increasing of the requests. In order to ensure the generalization property, we need to define another input features that can track how the optimal cost is expected to change when the number of requests increases. We use the result found in Chapter 6, where we experimentally confirmed the asymptotic results of [69] for small n. We indeed found that the expectation $\bar{Y}$ of the optimal cost $f(\lambda)$ applied to a PDP with n requests generated randomly in the domain $[0, 1] \times [0, 1]$ is:

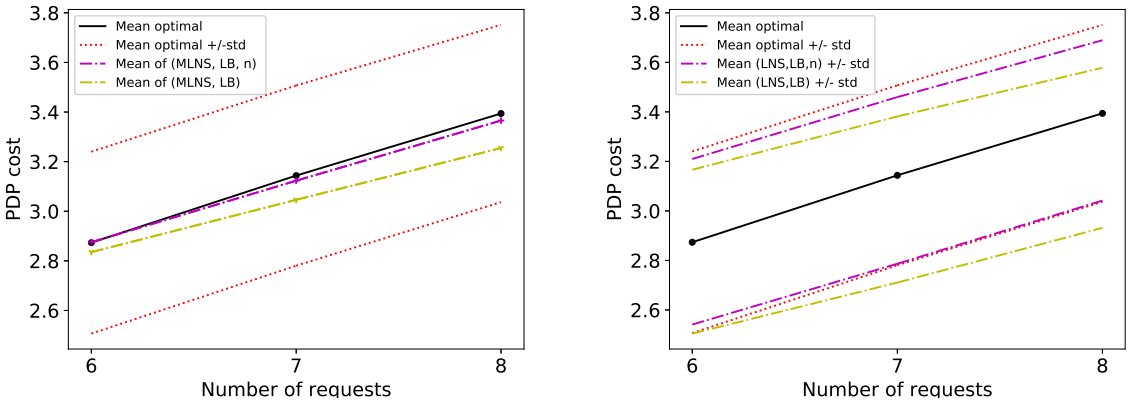
$$\bar{Y}_n = \alpha\sqrt{n} + \beta \tag{8-5}$$

where we derived that $\alpha = 1.54, \quad \beta = -0.9254$. With this value, we have still a dependence on the number of requests, but the relation follows the variations of the expectation value.

We re-trained the general models substituting the n feature with $\bar{Y}_n$. The results can be observed in Table 8-8 and Figure 8-16. In the Table, we can notice that the validation error

worsens especially for the model trained with the greedy heuristic, but the best model has not been affected dramatically. On the other hand, in Figure 8-16 we can appreciate how such best model follows the same trajectories of the optimal values.

.



(a) Average trajectories.

(b) Standard deviation trajectories.

Figure 8-16: Trajectories of averages and standard deviations. It can be noticed how the purple line now follows the trajectories of the average of optimal costs.

8-5 Summary

In this chapter we applied the investigations and results of the previous chapter to PDP with more than two requests. We used such results to do an educated guess about the structure of the NN predicting the optimal length of a PDP with six requests and we found the input features parameters which predict with the best accuracy. Then, we design a general model able to predict the optimal length of PDPs with different number of requests. We achieve such generalization given as additional input feature during the training the number of requests n. We inferred that the generalization capability can be ensured given as additional input feature not the number of requests, but the average optimal length of the PDP for different n.

Chapter 9

Conclusions and future research

Supervised learning is a powerful tool to estimate functions that can replicate complex and unknown dynamics. This thesis aimed to derive a machine learning model able to predict the cost of a combinatorial optimization problem: the single-vehicle pick-up and delivery problem. In literature, methods based on combinatorial neural networks and deep learning are used to solve routing problems, but such methods could result cumbersome to apply and they do not consider precedence constraints. We instead designed a neural network model with one hidden layer and ReLu activation function which predicts the optimal length of a routing problem with precedence constraints. We firstly investigated which input features ensure accurate prediction and we showed that the coordinates of the nodes to connect are not the most suitable input feature due to scalability issues. Moreover, we found that giving as input to the model the result of heuristic algorithms improves by 40% the performance compared to baseline prediction strategy that returns the length of the route computed by the greedy heuristic. In addition, we obtained even better performance giving as input also a lower bound estimation of the optimal length, leading to a further 13% improvement over the model trained with only the heuristic results. We also found that the model generalizes better to problems with a higher number of requests if it uses the average optimal length as an additional input feature.

As future research, we believe that it is possible to extend our method to other routing problems with time windows and cost function different from the minimum length tour, for instance the dial-a-ride problem that uses passenger travel delay as cost function. Furthermore, it would be worth inspecting if the predictions can indeed speed-up the algorithms which solve the routing problems (sub-)optimally in a ride-sharing scenario.

Bibliography

[1] A. Santos, N. McGuckin, H. Y. Nakamoto, D. Gray, and S. Liss, "Summary of travel trends: 2009 national household travel survey," tech. rep., 2011.

[2] D. o. E. United Nations and P. D. . Social Affairs, "World population prospects: The 2017 revision, key findings and advance tables," pp. 1382–1389, 2016.

[3] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences*, vol. 114, no. 3, pp. 462–467, 2017.

[4] M. Ota, H. Vo, C. Silva, and J. Freire, "Stars: Simulating taxi ride sharing at scale," *IEEE Transactions on Big Data*, vol. 3, pp. 349–361, Sept 2017.

[5] S. Ma, Y. Zheng, and O. Wolfson, "Real-time city-scale taxi ridesharing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1782–1795, 2015.

[6] A. Y. Lam, Y.-W. Leung, and X. Chu, "Autonomous-vehicle public transportation system: scheduling and admission control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 5, pp. 1210–1226, 2016.

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

[8] G. Berbeglia, J.-F. Cordeau, and G. Laporte, "Dynamic pickup and delivery problems," *European journal of operational research*, vol. 202, no. 1, pp. 8–15, 2010.

[9] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem: models and algorithms," *Annals of operations research*, vol. 153, no. 1, p. 29, 2007.

[10] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *Proceedings of the National Academy of Sciences*, vol. 111, no. 37, pp. 13290–13294, 2014.

[11] R. Zhang, F. Rossi, and M. Pavone, "Model predictive control of autonomous mobility-on-demand systems," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 1382–1389, IEEE, 2016.

[12] F. Miao, S. Han, S. Lin, J. A. Stankovic, D. Zhang, S. Munir, H. Huang, T. He, and G. J. Pappas, "Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach," *IEEE Transactions on Automation Science and Engineering*, vol. 13, pp. 463–478, April 2016.

[13] J. K. Lenstra and A. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981.

[14] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton university press, 2011.

[15] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pp. 410–421, April 2013.

[16] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Advances in Neural Information Processing Systems*, pp. 2692–2700, 2015.

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

[18] W. Kool and M. Welling, "Attention solves your tsp," *arXiv preprint arXiv:1803.08475*, 2018.

[19] J. A. Bondy, U. S. R. Murty, *et al.*, *Graph theory with applications*, vol. 290. Citeseer, 1976.

[20] F. L. Lewis, H. Zhang, K. Hengster-Movric, and A. Das, *Cooperative control of multi-agent systems: optimal and adaptive design approaches*. Springer Science & Business Media, 2013.

[21] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[22] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[23] K. K. B. Treleaven, *Probabilistic on-line transportation problems with carrying-capacity constraints*. PhD thesis, Massachusetts Institute of Technology, 2014.

[24] A. G. Percus and O. C. Martin, "Finite size and dimensional dependence in the euclidean traveling salesman problem," *Physical Review Letters*, vol. 76, no. 8, p. 1188, 1996.

[25] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," *Theoretical computer science*, vol. 4, no. 3, pp. 237–244, 1977.

[26] G. Laporte, "The traveling salesman problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[27] G. Clarke and J. W. Wright, "Scheduling of vehicles from a central depot to a number of delivery points," *Operations research*, vol. 12, no. 4, pp. 568–581, 1964.

[28] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[29] M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees," *Operations Research*, vol. 18, no. 6, pp. 1138–1162, 1970.

[30] M. Held and R. M. Karp, "The traveling-salesman problem and minimum spanning trees: Part ii," *Mathematical programming*, vol. 1, no. 1, pp. 6–25, 1971.

[31] C. L. Valenzuela and A. J. Jones, "Estimating the held-karp lower bound for the geometric tsp," *European journal of operational research*, vol. 102, no. 1, pp. 157–175, 1997.

[32] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013.

[33] H. Lei, G. Laporte, and B. Guo, "The capacitated vehicle routing problem with stochastic demands and time windows," *Computers & Operations Research*, vol. 38, no. 12, pp. 1775–1783, 2011.

[34] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, part i: Route construction and local search algorithms," *Transportation science*, vol. 39, no. 1, pp. 104–118, 2005.

[35] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, part ii: Metaheuristics," *Transportation science*, vol. 39, no. 1, pp. 119–139, 2005.

[36] J. R. Montoya-Torres, J. L. Franco, S. N. Isaza, H. F. Jiménez, and N. Herazo-Padilla, "A literature review on the vehicle routing problem with multiple depots," *Computers & Industrial Engineering*, vol. 79, pp. 115–129, 2015.

[37] M. Gendreau, G. Laporte, and R. Séguin, "Stochastic vehicle routing," *European Journal of Operational Research*, vol. 88, no. 1, pp. 3–12, 1996.

[38] E. Berhan, B. Beshah, D. Kitaw, and A. Abraham, "Stochastic vehicle routing problem: a literature survey," *Journal of Information & Knowledge Management*, vol. 13, no. 03, p. 1450022, 2014.

[39] U. Ritzinger, J. Puchinger, and R. F. Hartl, "A survey on dynamic and stochastic vehicle routing problems," *International Journal of Production Research*, vol. 54, no. 1, pp. 215–231, 2016.

[40] B.-H. Ahn and J.-Y. Shin, "Vehicle-routeing with time windows and time-varying congestion," *Journal of the Operational Research Society*, pp. 393–400, 1991.

[41] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte, "Static pickup and delivery problems: a classification scheme and survey," *Top*, vol. 15, no. 1, pp. 1–31, 2007.

[42] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *European journal of operational research*, vol. 59, no. 3, pp. 345–358, 1992.

[43] T. Vidal, T. G. Crainic, M. Gendreau, and C. Prins, "Heuristics for multi-attribute vehicle routing problems: A survey and synthesis," *European Journal of Operational Research*, vol. 231, no. 1, pp. 1–21, 2013.

[44] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[45] B. Eksioglu, A. V. Vural, and A. Reisman, "The vehicle routing problem: A taxonomic review," *Computers & Industrial Engineering*, vol. 57, no. 4, pp. 1472–1483, 2009.

[46] G. Laporte and I. H. Osman, "Routing problems: A bibliography," *Annals of Operations Research*, vol. 61, no. 1, pp. 227–262, 1995.

[47] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines*, vol. 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[48] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, pp. 65–93, Elsevier, 1992.

[49] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[50] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[51] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[52] R. Zhang and M. Pavone, "Control of robotic mobility-on-demand systems: a queueing-theoretical perspective," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 186–203, 2016.

[53] M. Pavone, S. L. Smith, E. Frazzoli, and D. Rus, "Robotic load balancing for mobility-on-demand systems," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 839–854, 2012.

[54] J. Alonso-Mora, A. Wallar, and D. Rus, "Predictive routing for autonomous mobility-on-demand systems with ride-sharing,"

[55] D. Delling, P. Sanders, D. Schultes, and D. Wagner, "Engineering route planning algorithms.," *Algorithmics of large and complex networks*, vol. 5515, pp. 117–139, 2009.

[56] K. Ruland and E. Rodin, "The pickup and delivery problem: Faces and branch-and-cut algorithm," *Computers & mathematics with applications*, vol. 33, no. 12, pp. 1–13, 1997.

[57] J. J. Hopfield and D. W. Tank, "âĂIJneuralâĂİ computation of decisions in optimization problems," *Biological cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[58] K. A. Smith, "Neural networks for combinatorial optimization: a review of more than a decade of research," *INFORMS Journal on Computing*, vol. 11, no. 1, pp. 15–34, 1999.

[59] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

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

[61] M. Nazari, A. Oroojlooy, L. V. Snyder, and M. Takáč, "Deep reinforcement learning for solving the vehicle routing problem," *arXiv preprint arXiv:1802.04240*, 2018.

[62] J. Rasku, T. Kärkkäinen, and N. Musliu, "Feature extractors for describing vehicle routing problem instances," *OASICS; 50*, 2016.

[63] Q. Lu and M. Dessouky, "An exact algorithm for the multiple vehicle pickup and delivery problem," *Transportation Science*, vol. 38, no. 4, pp. 503–514, 2004.

[64] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[65] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, 2013.

[66] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information theory*, vol. 39, no. 3, pp. 930–945, 1993.

[67] A. J. Smola *et al.*, *Regression estimation with support vector learning machines*. PhD thesis, MasterâĂŹs thesis, Technische Universität München, 1996.

[68] D. M. Stein, "An asymptotic, probabilistic analysis of a routing problem," *Mathematics of Operations Research*, vol. 3, no. 2, pp. 89–101, 1978.

[69] J. Beardwood, J. H. Halton, and J. M. Hammersley, "The shortest path through many points," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 55, pp. 299–327, Cambridge University Press, 1959.

[70] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

Glossary

List of Acronyms

AMOD	Autonomous Mobility On Demand
CVRP	Capacitated Vehicle Routing Problem
DARP	Dial-A-Ride Problem
DPDP	Dynamic Pick-up and Delivery Problem
DSVRP	Dynamic Stochastic Vehicle Routing Problem
DVRP	Dynamic Vehicle Routing Problem
HK	Held-Karp lower bound
ILP	Integer Linear Programming
LNS	Local Neighborhood Search
LP	Linear Programming
MILP	Mixed-Integer Linear Programming
MLNS	Modified Local Neighborhood Search
MLP	Multi-layer Perceptron
MOD	Mobility on Demand
MST	Minimum Spanning Tree
MDVRP	Multi-Depot Vehicle Routing Problem
NN	Neural Network
NP	Non-deterministic Polynomial-time
PDP	Pick-up and Delivery Problem

PDP	Pick-up and Delivery Problem
RBF	Radial Basis Function
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SVM	Support Vector Machine
SVR	Support Vector Regression
SVRP	Stochastic Vehicle Routing Problem
TSP	Travelling Salesman Problem
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows

List of Symbols

$\bar{L}_i$	Weighted lower bound of a Pick-up and Delivery Problem (PDP) istance.
$\bar{p}$	Cardinality of the set of solution of a Travelling Salesman Problem (TSP).
$\bar{Y}_n$	Average length of an optimal tour with n requests.
$\lambda(\rho)$	Length of the route ρ.
$\mathcal{E}$	Set of edges of graph $\mathcal{E}$.
$\mathcal{F}(R)$	Function which computes the otpimal length of a PDP given the set of requests R.
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	Graph.
$\mathcal{V}$	Set of nodes of graph $\mathcal{G}$.
$\mathcal{V}_D$	Set of destination nodes of graph $\mathcal{G}$.
$\mathcal{V}_O$	Set of origin nodes of graph $\mathcal{G}$.
ρ	Route of a PDP problem.
$\sigma(x)$	Activation function.
$\varphi(R)$	Input features related to the set of requests R.
A	Adjacency matrix.
a_{ij}	Entry of adjacency matrix.
C	Penalty parameter associated to the error of Support Vector Regression (SVR) model.
$d(\cdot)$	Function which computes a distance.

D_r	Destination of request r.
$d_{\mathcal{L}_1}$	Function which computes the $\mathcal{L}_1$ distance.
E	Array containg the edges of a routing problem.
e_{ij}	Edge connecting nodes v_i and v_j.
$f(\lambda)$	Function which returns the optimal length over all the routes.
G	Greedy function solving heuristically routing problems.
h	Number of hidden neurons.
L_i	Lower bound of a PDP istance.
m	Dimension of training data-set.
$M(n)$	General machine learning model to predict the PDP cost.
M_n	Machine learning model trained with examples of PDP with n requests.
m_{val}	Dimension of validation data-set.
O_r	Origin of request r.
R	Set of requests of a PDP.
$r = (O_r, D_r)$	Request for transportation.
v_i	Nodes belonging to $\mathcal{V}$.
x_{ij}	Optimization variable.
Y_n	Length of an optimal tour with $n \to \infty$.