Università degli Studi di Padova

Sede Amministrativa: Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

CORSO DI DOTTORATO DI RICERCA IN SCIENZE MATEMATICHE

CURRICULUM MATEMATICA COMPUTAZIONALE

CICLO XXXII

Solving a Multi-Attribute Vehicle Routing Problem in the freight delivery industry

Tesi redatta con il contributo finanziario dell'Azienda Trans-Cel Autotrasporti S.n.c. con contratto di Apprendistato di Alta Formazione e Ricerca

Coordinatore del Corso: Ch.mo Prof. Martino Bardi

Supervisore: Ch.mo Prof. Luigi De Giovanni

Supervisore aziendale: Filippo Sottovia

Dottorando: Nicola Gastaldon

Abstract

L'industria del trasporto merci è caratterizzata da diversi problemi decisionali che devono essere affrontati dagli operatori al traffico. Non solo si deve realizzare la pianificazione delle rotte in anticipo, ma devono prenderesi anche altri tipi di decisioni, in modo da far fronte ad eventi che possono dinamicamente presentarsi durante le operazioni, come ad esempio congestioni dovute al traffico o un guasto ad un veicolo. Ciascuna decisione può coinvolgere diversi aspetti: ad esempio, la negoziazione del prezzo di un ordine just-in-time dovrebbe tenere in considerazione lo stato corrente delle rotte e la loro pianificazione. I software di supporto alle decisioni disponibili sul mercato, seppur capaci di supportare il decisore in ciascuna area, tendono a mantenere i processi separati.
Trans-Cel, una piccola azienda di trasporto merci a Padova (Italia), ha un ramo di Ricerca e Sviluppo dedicato alla creazione di una piattaforma cloud, chiamata Chainment, che contenga diversi sistemi di supporto alle decisioni comunicanti fra loro attraverso un sistema di condivisione dati. Questi sistemi si affidano a un motore algoritmico che include un ottimizzatore per il routing dei veicoli e sistemi di intelligenza artificiale. In particolare, il problema di routing unisce le esigenze delle consegne espresse, studiate solitamente in contesti urbani, a caratteristiche dei veicoli e delle rotte tipiche di trasporti su distanze medio-lunghe, presentando caratteristiche peculiari e di interesse nel contesto della Ricerca Operativa.
In questa tesi ci concentriamo sullo sviluppo di un algoritmo di ottimizzazione capace di restituire una soluzione ad un nuovo Vehicle Routing Problem (VRP) ispirato dallo scenario di Trans-Cel, e che chiamiamo Express Pickup and Delivery in freight Trucking problem (EPDT).
La formulazione classica del VRP prevede un insieme di clienti e una flotta di mezzi e vuole definire un insieme di rotte tali che tutti i clienti siano visitati esattamente una volta e allo stesso tempo la distanza complessiva delle rotte sia minimizzata. Nella letteratura scientifica, la definizione base del problema è stata generalizzata in modo da considerare ulteriori attributi che spesso nascono dagli scenari reali, come ad esempio la capacità dei mezzi, finestre temporali e ordini che prevedono operazioni di carico e scarico. Spesso, nei casi reali, i decisori devono affrontare problemi con molti attributi da considerare simultaneamente, dando origine a una classe di problemi di routing chiamata Multi-Attribute VRP (MAVRP), che include il problema EPDT.
La tesi propone un algoritmo meta-euristico per la soluzione di EPDT, con lo scopo di integrarlo nel motore algoritmico di Chainment. Ai fini di essere compatibile con i requisiti della piattaforma, l'algoritmo è ideato in maniera che la soluzione venga restituita in pochi secondi.
Il metodo proposto consiste di un'euristica a due livelli: nel primo livello, un algoritmo Tabu Search ibridato con una Variable Neighborhood Descent

esplora le assegnazioni degli ordini ai veicoli, mentre il secondo livello fa uso di una Local Search per determinare la sequenza di clienti visitati e ottenere una valutazione delle rotte.
L'efficienza dell'algoritmo è migliorata dall'uso di filtri nell'esplorazione dei vicinati, da procedure per la valutazione rapida delle soluzioni, e dall'implementazione parallela di alcune componenti algoritmiche. Questi elementi sono adattati agli specifici attributi di EPDT e sono tra i contributi della tesi. Il miglioramento in termini di tempi di calcolo è stato validato dai risultati sperimentali, che verificano i requisiti desiderati per l'integrazione nella piattaforma.
La qualità delle soluzioni ottenute dall'algoritmo meta-euristico proposto è stata valutato sia sul campo, attraverso il confronto con operatori presso Trans-Cel, sia attraverso i bound ottenuti con metodi di programmazione matematica. A tale scopo, la tesi propone una formulazione di programmazione lineare intera per EPDT e un metodo di soluzione del suo rilassamento continuo basato su generazione di colonne. In particolare, la tesi presenta delle nuove procedure di pricing adatte ai diversi attributi di EPDT. I bound disponibili mostrano l'ottimalità o la quasi ottimalità delle soluzioni fornite dall'algoritmo euristico per le istanze reali. Inoltre, l'algoritmo è stato testato su benchmark di letteratura riguardanti il problema Pickup and Delivery Problem with Time Windows (PDPTW), mostrando soluzioni competitive con lo stato dell'arte. La tesi include anche uno studio preliminare di nuovi approcci per problemi di vehicle routing in contesti dinamici. In particolare, la tesi esplora la possibilità di trarre vantaggio dalla disponibilità di dati storici sugli ordini attraverso la predisposizione di opportune strategie anticipatorie (anticipatory algorithms). Una prima strategia si basa su metodi di clustering degli ordini per definire dei punti spazio-temporali che riassumono le informazioni sulla domanda futura. Una seconda strategia si basa sul concetto di accessibilità, come definito nella teoria della scelta discreta e della logistica territoriale, per rappresentare la capacità di una rotta di intercettare ordini futuri.
L'algoritmo euristico proposto per EPDT è stato integrato nel motore algoritmico della piattaforma Chainment di Trans-Cel. La tesi descrive le modalità di integrazione e gli adattamenti apportati agli algoritmi di ottimizzazione per una corretta interazione con i diversi moduli nel contesto delle operazioni gestite dalla piattaforma, come, ad esempio, la pianificazione iniziale delle rotte dei veicoli, la risposta a eventi dinamici o la contrattazione dei prezzi degli ordini.

Abstract

Freight transportation industry is characterized by several decisional problems that operations managers have to cope with. Not only the routes planning must be realized before their execution, but also other types of decisions must be taken, in order to answer events that may dynamically occur during operations, as for instance road network congestion or vehicle failures. Each decision can involve different aspects: for instance, the price negotiation of a just-in-time order should take into consideration the current routes status and planning. Off-the-shelf decision support software, although able to independently support the decision makers in each area, tend to keep tasks compartmentalized.
Trans-Cel, a small trucking company in Padova (Italy), has a Research and Development branch developing a cloud-based platform, called Chainment, able to host different decision support tools that can communicate through a data sharing system. These tools rely on an algorithmic engine that includes a routing optimization algorithm and artificial intelligence systems. In particular, the routing problem combines express couriers requirements, generally studied in urban contexts, with routes and vehicle features typical of medium- and long-haul trips, showing interesting characteristics that are worth of study in the Operation Research field.
In this thesis, we focus on the design of an optimization algorithm able to provide a solution to a Vehicle Routing Problem (VRP) inspired by the Trans-Cel scenario, that we name Express Pickup and Delivery in freight Trucking problem (EPDT).
The classical VRP definition includes a set of customers and a fleet of vehicles and aims to define a set of routes such that all customers are visited exactly once while minimizing the overall distance traveled. In the scientific literature, the basic definition of the problem has been generalized in order to consider additional attributes, often rising from real-world scenarios, as for instance capacity of vehicles, time windows and orders with both pickup and delivery operations. Often, in real-world cases, decision makers must simultaneously deal with a large number of attributes, thus defining a class of routing problems called Multi-Attribute VRP (MAVRP), which includes EPDT.
The thesis proposes a meta-heuristic algorithm for the solution of EPDT, with the aim of embedding it in the algorithmic engine of Chainment. In order to comply with the platform requirements, the algorithm is designed so that a solution is returned within few seconds.
The solution method we propose consists of a two-level heuristic: at the first level, a Tabu Search algorithm hybridized with a Variable Neighborhood Descent explores the order-to-vehicle assignments, while, at the second level, it makes use of a Local Search to determine the sequence of customers visited and obtain an evaluation of routes.

The algorithm efficiency is enhanced by the use of a granular exploration, by procedures for fast evaluation of solutions in the neighborhoods, and parallel implementation of specific algorithmic components. These elements are adapted to the specific attributes of EPDT and represent some of the thesis contributions. The improvement in computational times have been validated by the experimental results, verifying the desired requirements for the platform integration.
The quality of the solutions obtained with the proposed meta-heuristic algorithm has been assessed both on the field, by comparison with Trans-Cel operations managers, and through bounds obtained with mathematical programming methods. To this purpose, the thesis proposes an Integer Linear Programming formulation for EPDT and a solution method for its continuous relaxation based on Column Generation. In particular, the thesis presents new pricing procedures suitable for the specific EPDT attributes. The available bounds show optimality or near-optimality of solutions provided by the heuristic algorithm for real instances. Moreover, the algorithm has been tested on literature benchmarks related to the Pickup and Delivery Problem with Time Windows (PDPTW), providing solutions that are competitive with the state-of-the-art.
The thesis also proposes a preliminary study of new approaches for vehicle routing problems in dynamic contexts. In particular, the thesis explores the possibility of taking advantage from the availability of historical data on orders by means of anticipatory strategies. The first strategy is based on clustering methods that are applied to the orders to define space-time points that aggregate the information on future demand. A second strategy is based on the concept of accessibility, as defined in the discrete choice theory and urban logistic, to represent the route capability of intercepting future orders.
The heuristic algorithm proposed for EPDT has been integrated in the algorithmic engine of the Chainment platform at Trans-Cel. The thesis describes integration and the adaptation of the proposed optimization algorithms for a proper interaction with the different modules in the operational context handled by the platform, as, for instance, initial routes planning, reacting to dynamic events or order price negotiation.

Contents

1 Introduction **1**

1.1 Problem Overview 3

1.2 Structure of the thesis and contributions 4

2 Methodologies **7**

2.1 Linear Programming 8

2.1.1 The Simplex Algorithm 11

2.1.2 The Dantzig-Wolfe Reformulation 12

2.1.3 The Column Generation Algorithm 13

2.2 Algorithmic approaches to integrality constraint 15

2.2.1 Cutting Plane 16

2.2.2 Branch-and-Bound 17

2.2.3 Branch-and-Price 19

2.3 Dynamic Programming 19

2.3.1 Bellman-Ford Algorithm 19

2.3.2 Elementary Shortest Path Problem with Resource Constraints 20

2.4 Heuristic and Meta-heuristic approach 22

2.4.1 Greedy Algorithm 23

2.4.2 Neighborhood Search 24

2.4.3 Other popular Meta-Heuristic 27

2.4.4 Core characteristics of heuristic strategies 29

2.5 Machine Learning tools 30

2.5.1 Decision Tree Classifier 31

2.5.2 Support Vector Machine 32

2.5.3 K-Means 34

2.6 Implementation Framework 34

2.6.1 SCIP 34

2.6.2 SciKit-Learn 35

3 Description of the problem **37**
3.1 Introduction to Vehicle Routing Problems 37
3.2 A Multi-Attribute Express Freight Transportation Problem . 38
3.3 Entities and attributes 42
3.3.1 Positions 42
3.3.2 Vehicles 42
3.3.3 Orders 43
3.3.4 Routes 43
3.4 Problem definition 44

4 State-of-the-Art for MAVRPs **49**
4.1 VRP definition and basic models 49
4.1.1 Mathematical formulations 50
4.2 VRP extensions in the literature 53
4.2.1 VRP attributes classification 53
4.2.2 Extension of the objective function definition 56
4.3 Comparison with EPDT 57
4.4 Exact Methods for MAVRPs 58
4.5 Basic Exact Framework for VRP 59
4.5.1 The Pricing Problem 60
4.5.2 Branching strategies 61
4.6 Classical Heuristics for VRPs 62
4.7 Heuristic Approaches for MAVRPs 63
4.7.1 General approaches to MAVRPs 64
4.7.2 Specific approaches for pickup and delivery problems . 67
4.8 Approaches to Dynamic and Stochastic VRPs 68
4.8.1 Dynamic setting 68
4.8.2 Stochastic setting 69
4.8.3 Dynamic and Stochastic VRP 70

5 Design of a heuristic algorithm **73**
5.1 A two-level heuristic 74
5.2 The score function 75
5.3 Solution evaluation and second-level heuristic 77
5.3.1 Initial task sequence 79
5.3.2 Second-level neighborhoods 79
5.3.3 Overall Outline of the second level heuristic 79
5.4 Construction of the first-level Initial Solution 81
5.5 First-level Tabu Search neighborhoods 82
5.6 Exploration strategy: Tabu Variable Neighborhood Search . . 83
5.6.1 Sequential neighborhood switch 83
5.6.2 Cyclic neighborhood switch 84
5.6.3 Deterministic and random exploration 84
5.6.4 Tabu list and termination criteria 84

5.7 Overall outline of the first level heuristic 85
5.8 Granular Tabu Search 86
5.9 Destroy and Repair phase 86
5.10 Parallel Design 88
5.10.1 Parallelization on CPU 89
5.10.2 Parallelization on GPU 89

6 Bounding through a Column Generation Algorithm 93
6.1 An Integer Linear Programming formulation 93
6.2 The Master Problem 96
6.3 The Pricing Problem 96
6.3.1 Labels 97
6.3.2 Label extension 98
6.3.3 Dominance 99

7 Toward new Data-Driven approaches for DSVRPs 101
7.1 Dynamic and Stochastic VRP context 102
7.2 Data 104
7.3 Data-driven instance augmentation 105
7.3.1 Overall Procedure 105
7.3.2 Representative Orders 106
7.3.3 Choosing relocation positions and waiting times . . . 108
7.3.4 Analytical insight of waiting times 109
7.3.5 Towards an application to EPDT 115
7.4 Accessibility Approach 116
7.4.1 Overall Procedure 116
7.4.2 Data-driven accessibility measure 117
7.4.3 Towards an application to EPDT 119

8 Integration in a decision support system 121
8.1 Integrated support tool 121
8.2 The Orders Portal 122
8.3 The Driver App 124
8.4 The Planning Module 125
8.5 The Demand Forecast Tool 126
8.6 The Algorithmic Engine 127
8.6.1 The routing optimizer 127
8.6.2 Predictive models 128
8.7 Implementation technologies 129

9 Computational Results 135
9.1 General settings 135
9.2 Real-world Benchmarks definitions 137
9.3 Results of basic algorithm on real-world benchmarks 138

9.3.1 Selecting the initial solution procedure 138
9.3.2 Improvements from basic algorithmic components . . 139
9.4 Advanced algorithm settings on real world instance 140
9.4.1 Effect of filtering and parallel explorations 141
9.4.2 Impact of alternative exploration strategies 144
9.4.3 Statistical significance in algorithm selection 144
9.5 Assessment through optimality bounds 145
9.6 Tests on dynamic settings 147
9.6.1 Price estimation 147
9.6.2 Marginal cost estimation 148
9.6.3 Marginal cost estimation with 3 orders 148
9.7 Comparison with Literature Benchmarks 149

10 Conclusions **151**

Chapter 1

Introduction

Operations managers from the freight transportation industry have to deal with difficult decision problems arising during the work day. The planning of vehicles routes for satisfying customers requests is the most straightforward one, but many more issues take place during the operations, as for instance reacting to traffic jams, truck failures, delays on the load and unload at customer facilities. Not only the choices on the routing context play a role, but also other contour decisions contribute to the quality of the transportation service, with an indirect impact on the routing, as for example the price to assign to an incumbent customer request, as well as a marketing strategy based on defining low- and high-request rate areas. Off-the-shelf software tools are widely used by transportation companies in order to assist the operations managers in these difficult tasks: tools for real-time fleet management (e.g. TomTom Telematics [102]), vehicle route planning support (e.g. Driver [36]) as well as route optimizers (e.g. Workwave [115]) and on-line reservation of delivery time windows (e.g. Transporeon [107]).
Although these products are able to support the decision maker in each specific contexts, they keep tasks compartmentalized. As a consequence, the user will not likely see the broadest view of his supply-chain, missing potential benefits from coupling the systems. For instance, the price of an incumbent just-in-time request may be better decided if compared to the marginal cost in the current routes: pricing and routing are then interlaced by this point of view. Areas with high density of requests may support the routing design, e.g. asking for routes that traverse regions with higher requests rate: marketing and routing are then connected.
Trans-Cel, a small freight transportation company located in Padova (Italy), carries out the development of Chainment (see Figure 1.1), a cloud-based platform that interconnects various management tools through a data-sharing system. The input form contains different views (customer-side and operations manager-side) for handling requests: the customer issues the transportation requests and is notified about the status of the order, which in turn

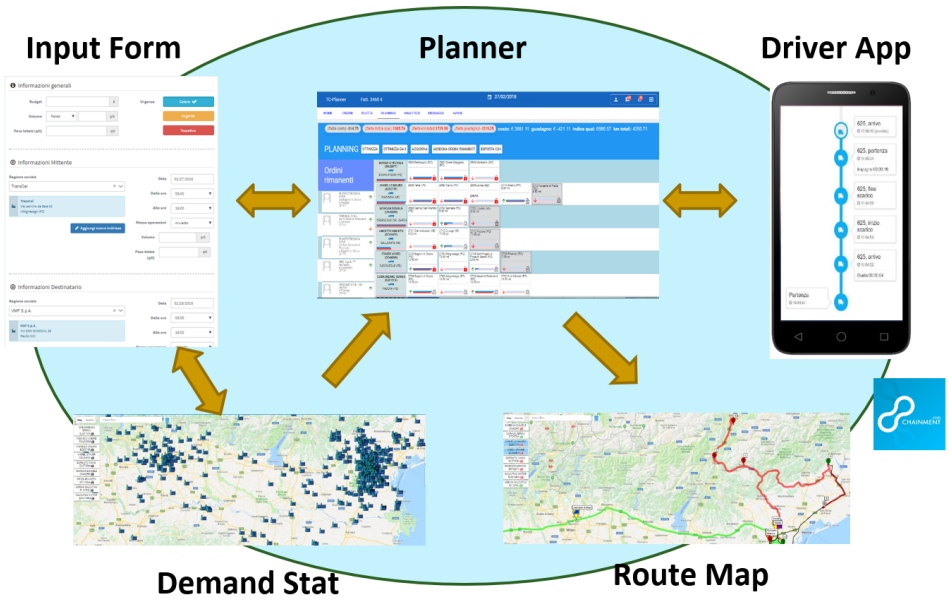

Figure 1.1: The inter-connection of Chainment modules.

is tracked by the Driver App. The operations manager proposes a price for the order that is negotiated on-line. The driver app is meant to truck drivers for receiving information on the route to execute (or any change to the current one), and transmitting to the central system feedbacks on operations completion, delays and truck maintenance issues. Further modules provide a geographical interface to the routes and locations of customers (Route Map), as well as statistical analysis of customers distribution over the areas of interest (Demand Stat).

The main support functionalities in the platform rely on an optimization engine and on artificial intelligence models. In particular, the optimization engine has to solve a vehicle routing problem that, in the incumbent context, issues interesting questions in the Operation Research field. In this research project we focus on the design of an optimization algorithm able to provide a solution to a vehicle routing problem inspired by the Trans-Cel scenario, that we named Express Pickup and Delivery in freight Trucking problem (EPDT). Besides solving EPDT, the algorithm is meant to provide the framework for the procedures to be implemented in the Chainment optimization engine, so it must comply with specific requirements dictated by the platform modules and the trucking operational context.

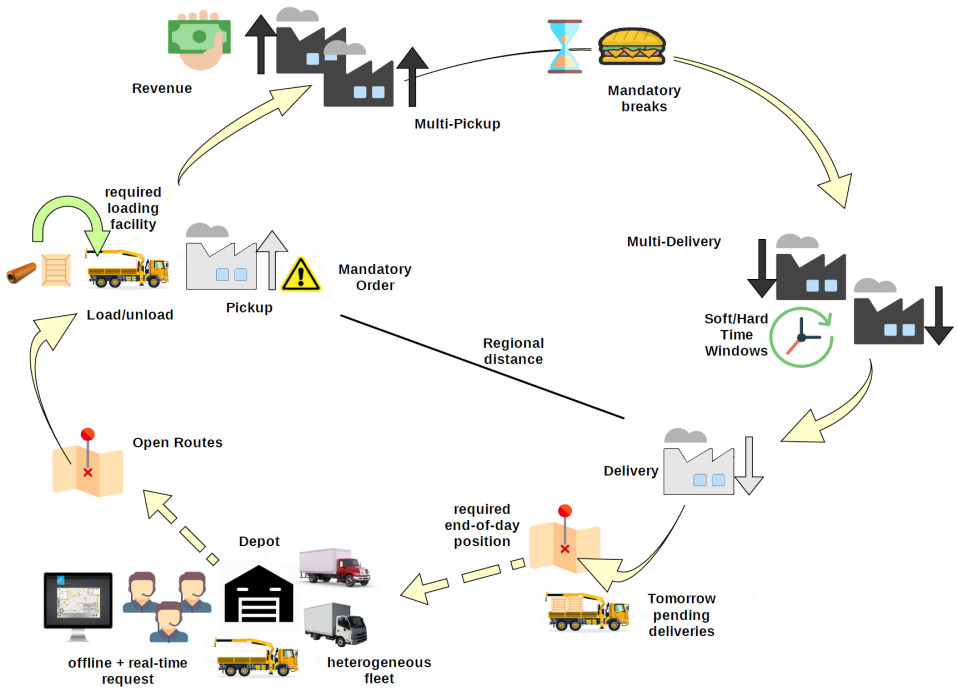

Figure 1.2: The MAVRP under study

1.1 Problem Overview

EPDT is modeled as an optimization problem belonging to the class of Vehicle Routing Problems (VRP, [104, 106] among others). The VRP classic definition consists of a set of customers and a fleet of identical vehicles, starting at a special location called depot, which have to visit each customer location while minimizing the overall traveled distance.
Most of the times, in real-world cases, a VRP comes with extra attributes that must be taken into account by operations managers while finding a solution, as for instance the capacity of vehicles, or the definition of pickup and delivery tasks, or the requirement of specific time windows.
EPDT comes with a large amount of attributes that play a role in the decision phase. As a consequence, EPDT belongs to the class of the so called Multi-Attribute VRPs (MAVRPs) [111, 113]. In Figure 1.2, we report the main features of EPDT. As we will detail in Chapter 3, in EPDT we consider a heterogeneous fleet, with different weight and volume capacities and loading tools. Drivers are subject to the European Hours of Service Regulation setting rules for maximum drive and work time as well as mandatory breaks. Customers ask for a just-in-time service to be fulfilled in the same working day or two consecutive days. Orders consist of multiple pickup and multiple delivery tasks, and all pickups must be visited before any delivery.

Moreover, time windows are specified for each task, as well as a revenue for completing the order and a urgency definition. Routes may or may not contain the depot and can contain pending orders from the day before and for the day after. In addition, a preferred or mandatory location can be required as last visit in the route, as well as a preferred maximum route length. Further attributes taken into account are a preferred minimum number of vehicles available at the depot, subsets of orders possibly assigned to a same vehicle and preferred order-to-vehicle assignment. The problem goal is the maximization of the net profit, so that orders may be discarded at the end of the solution procedure if not economically convenient.
In the context under study, order features are typical of intercity less-than-truckload couriers, however, the presence of urgent just-in-time requests reduces the impact of freight consolidation, typical of this setting, and introduces features similar to express couriers operations in urban contexts. As a consequence, the MAVRP under consideration shows some peculiar features that motivate the design of solution methods tailored on EPDT. For instance, the presence of multi-pickup and multi-delivery tasks as described above is, to the best of our knowledge, an attribute which is not treated in the literature and that needs to be suitably handled.
Researchers have investigated several exact and heuristic approaches to solve different classes of VRPs. The former are mainly based on mathematical programming, the latter on neighborhood search or genetic algorithms. Because of the high complexity of EPDT, together with the platform requirements for an acceptable computational time, we will consider a heuristic approach and, in particular, a solution method that hybridizes a Tabu Search algorithm with a Variable Neighborhood Descent, where multiple neighborhood are explored at each iteration.
In our context, the presence of software and hardware tools for real-time fleet management and data tracking fosters our research toward dynamic and stochastic versions of the VRP: in the former case, the requests is not completely known beforehand, but customer may ask for some operation during the execution of the planning; the latter case consists of the presence of parameters in the VRP problem affected by uncertainty and asking for suitable algorithmic techniques to cope with it.

1.2 Structure of the thesis and contributions

The thesis is divided into the following chapters:

- Chapter 2 provides the interested reader with a general and preliminary outline of the theoretical and methodological context supporting the research developed throughout this work. In particular, the first part gives the definition of a mathematical programming model, focusing on the Linear Programming case. We give a review of the Sim-

plex algorithm and of decomposition techniques, as the Dantzig-Wolfe formulation and the related Column Generation algorithm. We also provide a description of the main methods to deal with Mixed Integer Linear Programming problems, like the Branch-and-Bound or the Branch-and-Price algorithms. A quick description of Dynamic Programming algorithms for the Shortest Path Problem is given. Moreover, we explore the main heuristic approaches to Combinatorial Optimization problems, as well as the machine learning tools of interest for this thesis;

- Chapter 3 is devoted to the description of EPDT, the core problem of the thesis. A short introduction to Vehicle Routing Problems (VRPs) is provided, with a focus on Multi-Attribute VRP (MAVRP). In particular, we give a formal definition of EPDT as well as a model that formalizes all its specific attributes. The EPDT definition outlines a general problem statement that targets several vehicle routing scenarios among freight transportation companies and, to the best of our knowledge, is new to Operations Research literature;

- in Chapter 4 we review a VRP classification of the main extensions studied by researchers. Moreover, we describe some of the state-of-the-art solution methods that are proposed by literature to tackle several types of MAVRPs, both with exact and heuristic approaches, along with a more detailed description of popular algorithms for VRPs;

- Chapter 5 focuses on devising a meta-heuristic algorithm for the solution of EPDT. We explain in details all the components of the methods, that is based on a Tabu Search algorithm embedded in a Variable Neighborhood Descent framework. The algorithm presented in this chapter and its components represent one of the main contribution of the thesis. In fact, the algorithm includes specific features that are designed to match all the attributes of EPDT, as well as efficiency requirements: a fast evaluation phase for routes containing requests over two consecutive days; a special insertion operator to handle the multi-pickup and multi-delivery orders; a granular neighborhood exploration that filters less-promising neighbor solutions based on cliques of a tailored graph for multi-pickup and multi-delivery orders; parallel implementations of neighborhood exploration and evaluation to reduce the computational time of the overall procedure;

- Chapter 6 describes the algorithm that provides optimality bounds to the solution of EPDT. We implemented a Column Generation algorithm able to deal with EPDT instances. We formulate the Master Problem as a Set-Covering model, and the Pricing Problem is defined as a tailored Elementary Shortest Path Problem with Resource

Constraints (ESPPRC). The main contribution of the chapter is the development of a specific label correcting algorithm for ESPPRC, able to cope with all the attributes of EPDT. In particular, in order to deal with multi-pickup and multi-delivery orders, the algorithm relies on a brand-new label definition and on adapted label extension function and dominance rules;

- in Chapter 7, we make a preliminary exploration of two original data-driven approaches to tackle EPDT in dynamic and stochastic settings. The chapter focuses on special EPDT cases and on anticipatory techniques that, with the aim of keeping the computational effort low, do not make use of simulation. The first method is based on clustering the past orders to obtain space-time strategical points to summarize information on future requests. The second method assigns an accessibility measure to each point in space and time in order to represent the potential of each point and lead routes toward areas with high accessibility, possibly anticipating the future demand;

- Chapter 8 describes the integration of the algorithm defined at Chapter 5 within Chainment, the software platform that supports operations at Trans-Cel, the small freight transportation company whose business model inspired the definition of EPDT. We step into each module of Chainment, depicting how the optimization algorithm has been adapted to meet the platform requirements and how the interaction with predictive models comes into play; we remark the capability of the algorithm we devised to match the platform requirements in terms of efficiency, reaction to dynamic events and marginal cost evaluation;

- Chapter 9 contains all the computational results we collected through several testing on both literature benchmarks and real instances from Trans-Cel. Results assess the quality of the solutions proposed by the algorithm by means of a comparison with the optimality bounds returned by the Column Generation algorithm. Moreover, the efficiency gain achieved with neighborhood filtering and parallel implementations with respect to the plain solution method has been validated by our experimental results. Concerning literature benchmarks, computational tests consider instances of the Pickup and Delivery Problems with Time Windows (PDPTW) and the comparison with state-of-the-art methods shows the competitiveness of the solution approach proposed in this thesis in terms of quality of the solutions found;

- finally, in Chapter 10 we conclude this thesis with final observations and future research perspective.

Chapter 2

Methodologies

Optimization Problems have been largely studied by researchers, and different methods have been devised in order to efficiently solve them. An optimization problem is defined by a set of *variables* inside a *domain* and subject to some *constraints*, and an *objective function* that we want to *optimize*, that is minimize or maximize. In particular, a solution that optimizes the problem is called *optimal solution*. We will refer to the minimization case, an analogous theory holds for maximization.
There exist different solution methods to obtain one or more solutions to optimization problems. The algorithms that can find the optimal solution to the incumbent problem are called *exact algorithms*. Sometimes one must deal with problems that are computationally "difficult" to solve, mainly because of the large size of the instance and the complexity of the problem itself. In these cases, quite common in real applications, one can rely on algorithms that aim to find a feasible solution that is close to the optimum within a short (polynomial at most) amount of time. These type of solution methods are called *heuristic algorithms*.
A wide class of optimization problems can be formulated as mathematical programming problems and, in particular, Linear Programming (LP) problems. Below we describe important exact methods for Linear Programming problems: the *Simplex* algorithm and *Column Generation*. Then we outline two main strategies to deal with Mixed Integer Linear Programming (MILP) problems, where the domain of some variables is limited fo integer numbers: the *Cutting Plane* method *Branch-and-Bound*. A section is dedicated to a popular exact approach for the solution of the Shortest Path Problem based on dynamic programming, the *Bellman-Ford* algorithm.
The chapter gives also a description of some popular heuristic techniques, in particular *Greedy* algorithms, *Local Search*, *Variable Neighborhood Descent* and *Tabu Search*. Finally, the main Machine Learning tools we used are illustrated, both for regression and classification purposes: the *Support Vector Regression*, the *Decision Tree Classifier* and the *K-Means* clustering algo-

rithm. A complete description about LP and MILP and exact algorithms can be found at [17, 22, 61, 34]. Refer to [101] for a detailed characterization of heuristic and meta-heuristic methods.

2.1 Linear Programming

Mathematical Programming is a technique to handle an optimization problem by means of a mathematical formulation. In this context, the basic definition of a Minimization Mathematical Programming Problem is given by:

$$\min_x f(x) \tag{2.1}$$

s.t:

$$g_i(x) \leq 0 \qquad i = 1, \ldots, I \tag{2.2}$$

$$h_j(x) = 0 \qquad j = 1, \ldots, J \tag{2.3}$$

$$x \in X(\subseteq \mathbb{R}^n), \tag{2.4}$$

where $x \in \mathbb{R}^n$ is a vector of n variables, f is the objective function, g_i and h_j are, respectively, inequality and equality constraints and X is the domain. The domain together with the constraints define a subspace of $\mathbb{R}^n$ called *feasible region*, and a *feasible solution* (or, simply, *solution*) to the minimization problem is any vector of variable x which belongs to the feasible region.
Depending on the the domain and the constraints, a minimization problem can be:

- *infeasible:* if the feasible region is empty;
- *unbounded:* if the objective function can be arbitrarily decreased in the feasible region;
- *feasible:* in all the other cases.

An optimal solution x_s to a minimization problem is characterized by the fact that for any other solution x_r of the same problem, we have that $f(x_s) \leq f(x_r)$.
Mathematical Programming Problems can be classified by the nature of their objective function, domain and constraints. Below we report a list of the main types:

1. *Convex Optimization* studies the case where both the objective function and the feasible region are convex;
2. *Linear Programming* is a special case of the convex optimization class where the objective function and the constraint functions are all linear;

3. *Integer Programming* deals with optimization problems with integer variables;

4. *Quadratic Programming* admits quadratic terms in the objective function whereas the constraints are linear;

5. *Non-Linear Programming* allows both the objective function and the constraint functions to contain non-linear terms.

Linear Programming (LP) Problems are characterized by the fact that both the constraints and the objective functions are linear functions. In this context every LP problem can be rewritten as

$$\min_x c^T x \tag{2.5}$$

s.t:

$$Ax = b \tag{2.6}$$

$$x \geq 0, \tag{2.7}$$

where $x, c \in \mathbb{R}^n$ are, respectively, the vector of (decision) variables and the vector of costs, $b \in \mathbb{R}^m$ is the vector of constant terms for each constraint and $A \in \mathbb{R}^{m \times n}$ is the matrix of constraints. We call the form (2.5 - 2.7) of the problem the *standard form*.
The linearity of the constraints endows the domain with special geometrical properties: the region identified by the constraints is called *Polyhedron*. The points that belong to a polyhedron P can be represented by the linear combination of the extreme points (vertices) and the extreme rays of P, as stated by the following theorem:

Theorem 2.1 (Minkowski-Weyl Polyhedron representation)**.** *Given a polyhedron $P \subseteq \mathbb{R}^n$ with extreme points $v_1, \ldots, v_s \in \mathbb{R}^n$ and extreme rays $\rho_1, \ldots, \rho_t \in \mathbb{R}^n$, for any $x \in P$ we have that there exist $\lambda_1 \ldots \lambda_s \in \mathbb{R}$ and $\theta_1, \ldots, \theta_t \in \mathbb{R}$ such that*

$$x = \sum_{i=1}^{s} \lambda_i v_i + \sum_{j=1}^{t} \theta_j \rho_j, \quad \sum_{i=1}^{s} \lambda_i = 1, \lambda_i, \theta_j \geq 0 \; \forall i = 1, \ldots, s \text{ and } j = 1, \ldots, t.$$

Thanks to the linearity of the objective function, combined with Theorem 2.1, we obtain the theorem below:

Theorem 2.2. *Given an LP problem $\min_x \{c^T x, x \in P\}$ if P is non empty and bounded, then the problem is feasible and there exists at least one optimal solution corresponding to a vertex.*

This theorem tells us that the optimal solution of an LP problem can be searched in the set of the vertices of the polyhedron (see an example in

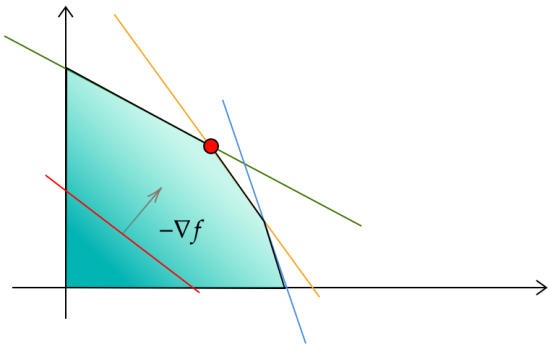

Figure 2.1: Example of optimal solution (in red) in a polyhedron of a LP with objective function f. Green, orange and blue lines correspond to the constraint linear functions.

Figure 2.1). Nevertheless, in large-size problems, the number of vertices can be exponentially large, then we cannot rely on exhaustive methods. Therefore, we need an algorithm that performs a clever exploration of such vertices, in order to reduce the computational time.
Such algorithm, called the Simplex algorithm, makes use of a link between the geometrical properties pointed out above and an algebraic point of view of the same concepts, that we briefly describe hereinafter.

We recall that a *basis of a matrix* A is a square sub-matrix of A with full rank. Given a system of equations of a LP problem in standard form $Ax = b$ and B basis of A, we can rewrite it as

$$\begin{bmatrix} B & F \end{bmatrix} \begin{bmatrix} x_B \\ x_F \end{bmatrix} = b$$

A solution to the system can be obtained by setting $x_F = 0$. Such solution is called *basic solution*, and it is feasible if and only if $B^{-1}b \geq 0$. The theorem stated below guarantees a correspondence between the basic solutions and the vertices of a polyhedron.

Theorem 2.3. *Given an LP problem* $\min_x\{c^T x, x \in P\}$ *with* P *non empty bounded and defined by the system of equations* $Ax = b, x \geq 0$*, we have that* x *is a feasible basic solution of the system of equations if and only if it is a vertex of the polyhedron* P*.*

This theorem suggests that exploring the vertices while searching the optimal solution is equivalent, from an algebraic point of view, to visiting the basic solutions, and this latter routine can be exploited by a solution method, as the Simplex Algorithm described in the following.

2.1.1 The Simplex Algorithm

The Simplex Algorithm is an exact method to find the optimal solution of an LP problem by moving through the vertices of the polyhedron defined by the constraint matrix. The main idea behind this algorithm is starting from a vertex and moving to adjacent vertices so that the objective function improves till either reaching optimality, if the problem is feasible, or detect the unboundness of the problem.
A fundamental observation about the simplex method is that moving from one vertex to another is equivalent to swap one column in the basis B with a column out of the basis in F, while preserving the non-negativity condition for a basic solution to be feasible and ensuring that the new basis is full-rank.
We rewrite once again the system of equations $Ax = b$ in a more convenient form in order to let suitable columns enter and exit the basis:

Definition 2.1. *Given an LP problem $P = \min_x\{Ax = b, x \geq 0\}$ and a basis $B \in \mathbb{R}^{m\times m}$ of a full-rank matrix $A \in \mathbb{R}^{m\times n}$, then P is in* canonical form *if the objective function $z := c^T x$ and the variables in basis are expressed as functions of the variables out of basis.*

$$z = \bar{z}_B + \sum_{j\in J_F} \bar{c}_j x_j \tag{2.8}$$

$$x_{B_i} = \bar{b}_i - \sum_{j\in J_F} \bar{a}_{ij} x_j \qquad \forall\, i = 1, \dots, m \tag{2.9}$$

where $\bar{z}_B$ and $\bar{b}_i$ are the values of, respectively, the objective function and the variables of the basic solution with basis B, J_F is the set of the indices out of base B, $\bar{c}_j$ and $\bar{a}_{ij}$ are, respectively, the cost coefficients (also called *reduced cost*), and the constraint coefficient at row i, relative to the j-th variable out of basis.
We can collect some useful information by the canonical form about

1. the value of the variable x_j that sets to 0 the variable leaving the basis;
2. the impact on the objective function;
3. the new values of the variables in the basic solution.

In particular, we observe in (2.8) that the absence of negative reduced costs implies that any variable that enters the basis will just deteriorate the objective value, so we can use it as an *optimality test* to stop the run of the algorithm and return the current (optimal) solution.
Therefore, by (2.8) we can choose any variable x_j such that $\bar{c}_j < 0$ to enter the basis. We then increase the value of x_j as much as possible so that (at

least) one variable in the basis reaches value 0, or equivalently, it has been selected to leave the basis. Notice that one cannot set x_j at any value that makes x_B negative, otherwise we obtain an infeasible solution. If we can infinitely increase the chosen x_j, then the problem is unbounded, and this corresponds (by 2.9) to the *unboundness condition*

$$\bar{a}_{ij} \leq 0 \quad \forall i = 1, \ldots, m.$$

When this condition is not met, one can see that the best value at which the variable enters the basis is

$$x_j = \min_i \left\{ \frac{\bar{b}_i}{\bar{a}_{ij}} : i = 1, \ldots, m, \bar{a}_{ij} > 0 \; \forall \; i \right\}.$$

We sum up the steps of the simplex algorithm in Algorithm 1.

Algorithm 1: Simplex Algorithm

1 *Initialization*: start from a feasible basic solution of the basis B;
2 set the LP in the canonical form with respect to B and compute the reduced costs $\bar{c}_j \; \forall j \in J_F$
3 **if** $\bar{c}_j \geq 0 \; \forall j \in J_F$ **then**
4 | STOP: optimal solution found;
5 **end**
6 Select any x_h s.t: $h \in J_F$ and $\bar{c}_h < 0$
7 **if** $\bar{a}_{ih} \leq 0 \; \forall i = 1, \ldots, m$ **then**
8 | STOP: problem unbounded;
9 **end**
10 compute $\min \left\{ \frac{\bar{b}_i}{\bar{a}_{ij}} : i = 1, \ldots, m, \bar{a}_{ij} > 0 \; \forall \; i \right\}$;
11 update the next basic solution.
12 `goto` Step 2.

2.1.2 The Dantzig-Wolfe Reformulation

In the previous section we have found a relation between the solutions of a system of inequalities $Ax = b$ and a polyhedron $P \subseteq \mathbb{R}^n$. In particular, an LP problem $\min_x \{c^T x, Ax = b, x \geq 0\}$, can be rewritten as

$$\min_x \{c^T x, x \in P\}.$$

By Theorem 2.1, we know that any point x of a non empty convex polyhedron P can be expressed by a convex combination of its extreme points $\{v_1, \ldots, v_s\}$ and a weighted combination of its extreme rays $\{\rho_1, \ldots, \rho_t\}$.

Hence, replacing each variable of the LP with its representation by extreme points and extreme rays, we obtain the *Dantzig-Wolfe reformulation*:

$$\min \sum_{i=1}^{s}(c^T v_i)\lambda_i + \sum_{j=1}^{t}(c^T \rho_j)\theta_j \tag{2.10}$$

s.t:

$$\sum_{i=1}^{s}(Av_i)\lambda_i + \sum_{j=1}^{t}(A\rho_j)\theta_j = b \tag{2.11}$$

$$\sum_{i=1}^{s}\lambda_i = 1 \tag{2.12}$$

$$\lambda_i, \theta_j \geq 0. \tag{2.13}$$

We observe that this reformulation may be very useful, e.g. when there are "difficult" constraints in the LP problem, since they are all implicitly respected, as we will se for the case of VRP problems and its extensions. Nevertheless, the number of variables is exponentially large, so we must make use of some suitable algorithm that can deal with such type of LP problems, as for instance the Column Generation algorithm.

2.1.3 The Column Generation Algorithm

The *Column Generation (CG)* algorithm decomposes the original problems into two sub-problems:

- The *Master Problem (MP)*: its solution corresponds to the optimal solution of the original problem;
- The *Slave Problem (SP)*: it generates variables to add into the *Reduced Master Problem (RMP)*, that is a version of the Master Problem defined on a subset of variables.

Before the description of the algorithm in Algorithm 2, we need some concepts of duality theory in linear programming.
Every LP problem P (called *Primal Problem*) is associated with another LP problem D (called *Dual Problem*) with some special relation between its characteristics, mainly the variables and the constraints, the inequality signs and the domain, the objective function and the constant terms. The relations are summarized in Table 2.1, where the x_j and u_i are the primal and dual variables respectively.

The values of the objective function of the Primal and Dual problems in Linear Programming are in a particular relationship stated by the following theorems:

Primal Problem	Dual Problem
min	max
$x_j \geq 0$	$A_j^T u \leq c_j$
$x_j \in \mathbb{R}$	$A_j^T u = c_j$
$x_j \leq 0$	$A_j^T u \geq c_j$
$A_i x \geq b_i$	$u_i \geq 0$
$A_i x = b_i$	$u_i \in \mathbb{R}$
$A_i x \leq b_i$	$u_i \leq 0$

Table 2.1: Primal-Dual relations

Theorem 2.4 (Weak Duality)**.** *Given a primal problem* $P = \min\{c^T x : Ax \leq b, x \geq 0\}$ *and its dual problem* $D = \max\{b^T u : A^T u \geq c, u \geq 0\}$*, if the feasible regions are not empty, for any feasible solutions* x *of* P *and* u *of* D *it holds*

$$b^T u \leq c^T x.$$

Theorem 2.5 (Strong Duality)**.** *Given a primal problem* $P = \min\{c^T x : Ax \leq b, x \geq 0\}$ *and its dual problem* $D = \max\{b^T u : A^T u \geq c, u \geq 0\}$*, if the feasible regions are not empty and* x^* *is optimal for* P*, then there exists also an optimal solution* u^* *for* D *and it holds:*

$$b^T u^* = c^T x^*.$$

Corollary 2.1. *Given a feasible solution* x^* *to a primal problem* $P = \min\{c^T x : Ax \leq b, x \geq 0\}$ *and a feasible solution* u^* *to its dual problem* $D = \max\{b^T u : A^T u \geq c, u \geq 0\}$ *and* $b^T u^* = c^T x^*$*, then* x^* *and* u^* *are also optimal.*

The first step in the Column Generation is finding the primal and dual solutions x^* and u^* of the RMP by using the simplex algorithm. In particular, the dual solution u^* returned by the simplex is such that $b^T u^* = c^T x^*$. The primal solution x^* is feasible for the MP, and if also u^* is feasible for its dual, then by Corollary 2.1 we have optimally solved MP and the algorithm stops. If u^* is infeasible for the dual problem of MP, there exists at least one violated constraint, that is there exists a $j \in \{1, \ldots, n\}$ such that

$$c_j - u^T A_j < 0.$$

Notice that, since the solution u^* provided by the simplex is by construction $u^* = c_B B^{-1}$ with B the optimal basis of RMP, finding such violated constraint in the dual problem of MP corresponds to finding a variable in the primal MP with negative reduced cost $\bar{c}_j$. Finding such a variable or proving that no such variable exists, is a non-trivial task solved by SP. One way to solve SP is modeling it as an optimization problem

$$\min\{c_j - u^T A_j, j = 1, \ldots, n\}.$$

If this problem provides a negative solution, the corresponding variable is added to the RMP and the procedure is iterated.

Algorithm 2: Column Generation Algorithm

1 *Initialization*: start from RMP;
2 solve the RMP and obtain primal and dual solutions x^* and u^*;
3 solve the SP to find if $\exists\, j \in \{1, \ldots, n\}$ s.t: $\bar{c}_j < 0\}$;
4 **if** $\bar{c}_j \geq 0 \;\forall\; j = 1, \ldots, n$ **then**
5 | STOP: optimal solution found;
6 **end**
7 **else**
8 | add to RMP a variable x_j s.t: $\bar{c}_j < 0$.
9 | `goto` line 2.
10 **end**

2.2 Algorithmic approaches to integrality constraint

The Simplex algorithm and the Column Generation algorithm work on the assumption that all variables are continuous. Thus, they are not suitable for problems containing one or more integer variables (see Figure 2.2). Problems of this type are called *Mixed-Integer Linear Programming (MILP)* problems and they can be formulated as

$$\min_{(x,y)} c_1^T x + c_2^T y \tag{2.14}$$

s.t:

$$D_1 x + D_2 y \leq d \tag{2.15}$$

$$A_1 x \leq b_1 \tag{2.16}$$

$$A_2 y \leq b_2 \tag{2.17}$$

$$x_i \in \mathbb{R}_+, \quad i = 1, \ldots, n_1 \tag{2.18}$$

$$y_j \in \mathbb{Z}_+. \quad j = 1, \ldots, n_2 \tag{2.19}$$

where n_1 and n_2 are respectively the number of real and integer variables in the problem.

Several problems in Operations Research can be formulated as MILP, since integer and binary variables can represent a discrete choice or a logical selection.

A MILP can be relaxed by dropping the integrality constraint, then the

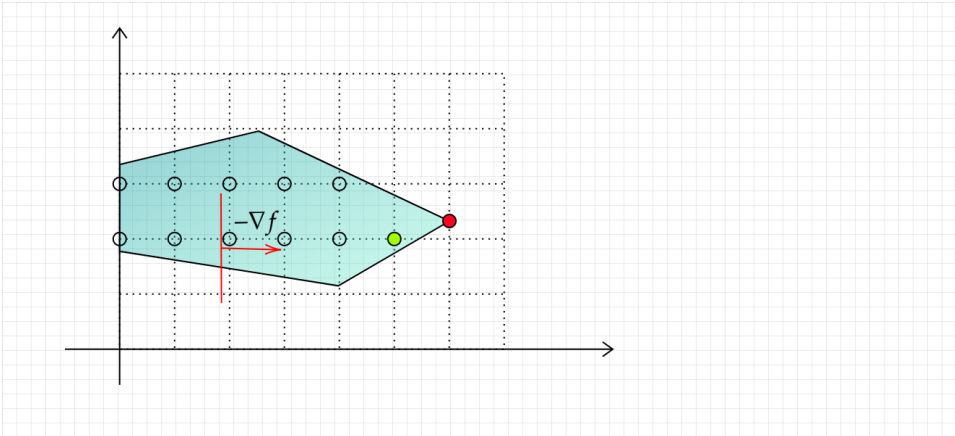

Figure 2.2: Example of a MILP with objective function f. The image shows the difference between the optimal solution found by the Simplex algorithm on the continuous relaxation of the problem (red point) and the optimal solution of the MILP (green point).

problem becomes

$$\min_{(x,y)} c_1^T x + c_2^T y \tag{2.20}$$

s.t:

$$D_1 x + D_2 y \leq d \tag{2.21}$$

$$A_1 x \leq b_1 \tag{2.22}$$

$$A_2 y \leq b_2 \tag{2.23}$$

$$x_i, y_j \in \mathbb{R}_+, \quad i = 1, \ldots, n_1 \text{ and } j = 1, \ldots, n_2 \tag{2.24}$$

The solution of the relaxed problem, may be infeasible to the original problem, because of the potential presence of fractional values in correspondence of the integer variables. However, the matrix of constraints may have some properties such that the solution to the continuous relaxation of the problem contains only integer variables: this is the case, for example, of *Totally Unimodular* constraint matrices.

Definition 2.2. *A matrix A is called* Totally Unimodular *if the determinant of every minor of A is 0, $+1$ or -1.*

2.2.1 Cutting Plane

One way to deal with MILPs is based on the concept of *convex hull*. The convex hull is the intersection of all convex polyhedra containing the feasible region. Loosely speaking, it is the "tightest" convex polyhedron enclosing

the feasible region. Given a MILP and the set F containing all its feasible points, we can define the following relaxation of the problem

$$\min_x c_1^T x + c_2^T y \tag{2.25}$$

s.t:

$$x, y \in conv(F), \tag{2.26}$$

$$\tag{2.27}$$

where $conv(F)$ is the convex hull of F. It can be proved that, given a MILP, the optimal solution of the LP above is also optimal for the original problem. Indeed, all the vertices of $conv(F)$ are contained in F, and this includes also the optimal solution of the LP.
This observation, though based on an easy concept, discloses the problem of finding the equations representing the convex hull of a set of points. As a matter of fact, this is often an impractical task.
Given a MILP with domain F and its LP relaxed version with feasible region P, a *cut* is an inequality constraint that, if added to the current set of constraint of the LP, does not exclude any point of F and it excludes at least one extreme point of P. The *cutting plane* approach iteratively adds cuts to the relaxed problem to tighten the feasible region until it reaches the optimal vertex of $conv(F)$. Hence, given a solution of the relaxed problem that is infeasible for the MILP, the effort of the algorithm is to find a cut that excludes the current solution from the domain. This is known as the *separation problem.*

2.2.2 Branch-and-Bound

A different solution method for a MILP, called Branch-and-Bound (B&B), is based on the fact that, in order to find the optimal solution of an optimization problem, we can partition its domain $X = \{X_1, \ldots, X_n\}$, solve the problem in each X_i, $i = 1, \ldots, n$ and return the best solution found among all the sub-problems. For instance, in a minimization MILP $\min_x\{c^T x, Ax = b, x \in X\}$, given a partition $\{X_1, \ldots, X_n\}$ we can find the optimal solution x_i^* in the restricted domain X_i for each i. Therefore, the optimal solution is

$$\arg\min_{i=1,\ldots,n}\{c^T x_i^*\}.$$

The branch-and-bound algorithm, shown in Figure 2.3, recursively partitions the domain of the continuous relaxation of the MILP into subsets associated with *nodes* of a tree structure, in order to find a solution that is at the same time integer and optimal. Without any particular strategy, the algorithm would just end up enumerating all feasible solutions, which would be totally inefficient.
In order to overcome this issue, the Branch-and-Bound algorithm exploits

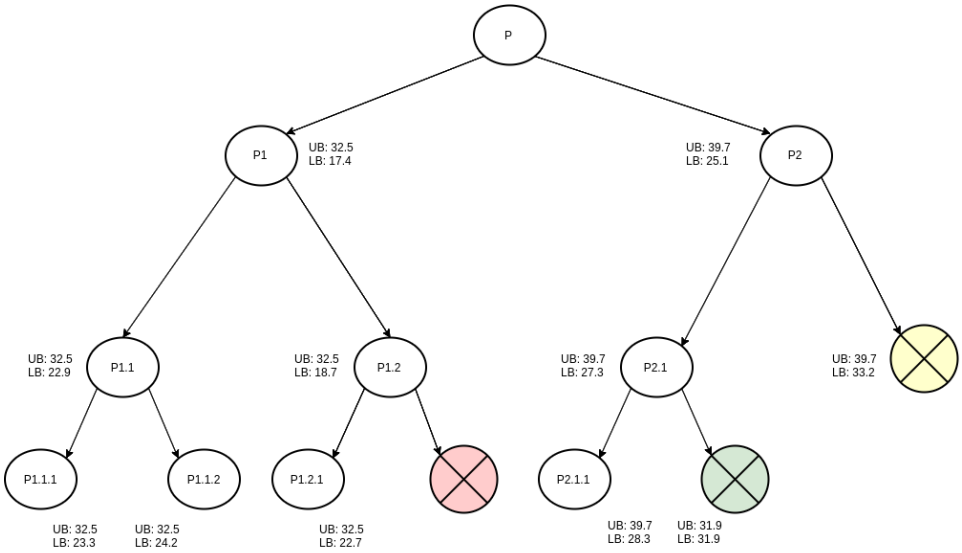

Figure 2.3: Example of a Branch-and-Bound tree. The green node is fathomed by integrality, the yellow is pruned by bound and the red by infeasibility.

information on bounds to the optimal value of the global solution that is collected at each exploration of a node, in order to a-priori dismiss the exploration of some branches that certainly do not contain the global optimal solution. This operation is called *pruning* of a branch. Let us consider a minimization problem. A fundamental observation is that given a node n and an upper and a lower bounds U_n, L_n to the solution of the restricted problem at n, any other node derived by branching on n cannot contain any optimal solution outside the interval $[L_n, U_n]$. The cases in which one can prune a branch on a node n are:

1. whenever a problem on n is infeasible (pruning by *infeasibility*);
2. whenever U_n is an integer solution (pruning by *integrality*);
3. whenever there exist another node n' such that $L_n \geq U_{n'}$ (pruning by *bound*).

The procedure stops when there are no non-pruned nodes to explore. Notice that, for minimization MILP problems, a lower bound can be obtained by solving the linear relaxation, leading to the B&B method for MILPs.
Different extensions can be embedded in a Branch-and-Bound framework. For instance, the *Branch-and-Cut* (B&C) algorithm mixes features from the B&B and the Cutting Plane methods: each time a sub-problem is solved at a node of the Branch-and-Bound tree, valid inequalities can be generated and added to the sub-problem constraints in order to improve the bound

corresponding to that node, with the aim of enhancing the pruning phase and speeding up the convergence.

2.2.3 Branch-and-Price

The *Branch-and-Price* (B&P) relies on a B&B strategy hybridized with a Column Generation algorithm: the solution of the LP sub-problems at the nodes of the B&B are obtained by means of a Master-Slave decomposition and new variables are added in order to find the sub-problems' optima. In particular, the Slave Problem, also called Pricing Problem (PP), must efficiently take into account the constraints generated by the branching rule at each node for the B&P to properly work. In fact, a good B&P design ensures the Pricing Problem to be subject to small-impact changes (or no changes at all) at each node of the tree. This is enforced by setting up a branching rule that involves the Pricing Problem variables.

2.3 Dynamic Programming

Dynamic programming is an algorithmic paradigm that divides a problem into multiple (easier) sub-problems so that the solutions of each of them concur to recover the solution of the initial problem.
In this context, the optimization procedure must be characterized by a *multi-stage* structure. Stages are sub-problems solved sequentially one at a time in such a way that the solution of sub-problems at some stage can exploit the solutions at previous stages. This type of optimization requires a *recursive procedure* that firstly solves one stage and then, on top of that, sequentially builds the optimal solution of the following stages.

2.3.1 Bellman-Ford Algorithm

Efficient applications of the Dynamic Programming technique appear in solving methods for the *Shortest-Path Problem (SPP)* with polynomial complexity. Given a graph $G = (N, A)$ with costs c_{ij} defined on each arc $(i, j) \in A$ and a source and destination nodes s and d respectively, the solution of the SPP is a path of minimum cost starting from s and ending in d. The *Bellman-Ford algorithm* is a dynamic programming algorithm that finds a solution to the SPP in polynomial time. The method defines a set of *labels* for each node $i \in N$, encoding the cost of the shortest paths from s to each node reached in h steps (the h-th stage of the dynamic programming procedure). The optimization strategy is based on the iterative correction of the labels, until the minimum cost path is found, as long as the graph does not contain negative-cost cycles. Algorithm 3 shows the main steps of the Bellman-Ford algorithm, where $p(n)$ represents the predecessor of node n in a path P and π_i are the labels for $i \in N$.

Algorithm 3: Bellman-Ford Algorithm

```
1  Initialization: π_s = 0, p(s) = NULL;
2  foreach v ∈ N \ {s} do
3  |   π_v = +∞;
4  |   p(v) = NULL;
5  end
6  for h = 1, ..., N do
7  |   π' = π;
8  |   updated = false;
9  |   foreach (i, j) ∈ A : π_j > π_i + c_ij do
10 |   |   π_j = π'_i + c_ij;
11 |   |   p(j) = i;
12 |   |   updated = true;
13 |   end
14 |   if not updated then
15 |   |   STOP: optimal solution found.
16 |   end
17 end
18 STOP: ∃ negative cost cycle.
```

2.3.2 Elementary Shortest Path Problem with Resource Constraints

In several context, a path is required that does not contain cycles.

Definition 2.3. *A path is* elementary *if it does not pass through the same node more than once.*

Moreover, a set of constraints can be defined over a set of resources. A resource refers to a quantity that changes incrementally along a path according to some functions, called *resource extension functions* [55].
This problem is known as the *Elementary Shortest Path Problem with Resource Constraints (ESPPRC)* [31]. In order to deal with ESPPRC, a dynamic programming technique exists, inspired by Bellman-Ford algorithm, that guarantees a solution to be feasible with respect to constraints that limit the use of the resources.
The basic solution method for the ESPPRC is a *label correcting* algorithm where at any node there can be more than one label, each representing a path starting from a source and ending at that node, and taking into account both the cost of the partial path and the consumption of resources (refer to [31] for a detailed discussion). In the procedure, labels are extended to every reachable node in the graph and *dominance rules* are introduced in order to fathom partial paths that surely cannot be extended to optimal solutions. The algorithm converges in pseudo-polynomial time and the efficiency is

strongly related to the design of the extension function and the domination rules. Moreover, according to the dynamic programming paradigm, at any iteration, label extension are performed *incrementally*, based on the values of resources computed at the previous iteration.

Definition 2.4. *Given a path p_i from the origin node s to the node i, we define the* state *as tuple $R_i = (T_1^i, \ldots, T_L^i, U_1^i, \ldots, U_{|N|}^i)$. Each entry of the tuple T_k^i, $k = 1, \ldots, L$ is called* resource*, whereas the vector U_m^i, $m = 1, \ldots, |N|$ is the vector of* unreachable nodes*.*

The value of each T_k^i represents the quantity of that resource consumed along the path p_i, while the variables U_m^i are flags for unreachable nodes, that are either nodes already contained in p_i, or nodes at which an extension of the current label would produce an infeasible path.

Definition 2.5 (label)**.** *Given a path p_i from the origin node s to the node i, we define as* label *the pair $\lambda_i = (R_i, C_i)$ where R_i is the state of p_i and C_i is the cost of p_i.*

In the following definition we introduce the concept of dominance between two labels.

Definition 2.6 (dominance)**.** *Let p_i^1 and p_i^2 be two distinct paths from s to i with associated labels λ_i^1 and λ_i^2 respectively. We say that p_i^1* dominates *p_i^2 if and only if $C_i^1 \leq C_i^2$, $T_i^{1m} \leq T_i^{2m}$, $m = 1, \ldots, L$ and $U_i^{1m} \leq U_i^{2m}$, $m = 1, \ldots, |N|$.*

In particular, it can be proved that any dominated label can be removed from the set of labels during the algorithm run. The reason is that, given a label λ_i^1 and a dominated label λ_i^2, the set of extensions of the path related to λ_i^1 contains the set of extensions of the path related to λ_i^2, therefore λ_i^2 can be discarded since all its extensions will be generated starting from λ_i^1. Algorithm 4 shows the label correcting method to exactly solve ESPPRC. We make use of the following notation:

- Λ_i: list of labels at node i;
- E: the list of to-be-processed nodes;
- $Extend(\lambda_i, j)$: the function that returns a singleton containing the label that results by the extension of label λ_i to node j if feasible, otherwise it returns the empty set;
- F_{ij}: set of labels extended from node i to node j;
- $filterDom(\Lambda_i)$: function that filters Λ_i removing all dominated labels;

The procedure initializes at 1-5 the set of labels in each node as an empty set, except from the source node, where a label with all resources at value 0 is added. Also, the set of unprocessed nodes is initialized with the source node s. At 7-19, an unprocessed node i is selected and all the possible extensions of the labels in i to $\delta^+(i)$ are generated. In particular, dominated labels are discarded at 15. Finally, at Step 20, the processed node is removed from E and the procedure is repeated as long as E is not empty. At the end of the algorithm we obtain the optimal paths from the source node to each other node of the graph.

Algorithm 4: Label-Correcting algorithm for ESPPRC

```
1  Λ_s = {(0, ..., 0)}
2  foreach i ∈ N \ {s} do
3  |   Λ_i = ∅;
4  end
5  E = {s};
6  while E ≠ ∅ do
7  |   choose i ∈ E;
8  |   foreach j ∈ δ⁺(i) do
9  |   |   F_ij = ∅;
10 |   |   foreach λ_i ∈ Λ_i do
11 |   |   |   if U_i^j = 0 then
12 |   |   |   |   F_ij = F_ij ∪ Extend(λ_i, j);
13 |   |   |   end
14 |   |   end
15 |   |   Λ_j = filterDom(F_ij ∪ Λ_j);
16 |   |   if Λ_j has changed then
17 |   |   |   E = E ∪ {j};
18 |   |   end
19 |   end
20 |   E = E \ {j};
21 end
```

2.4 Heuristic and Meta-heuristic approach

Sometimes optimization problems can be very hard to be optimally solved in practice. Exact algorithms may be not efficient enough at finding the optimal solution. This can happen for instance with *NP-Complete* and *NP-Hard* problems, especially with large-scale instances. *Heuristic* and *Meta-Heuristic* algorithms have been devised, in order to obtain a feasible solution of high quality within an acceptable amount of time.

We can divide heuristic algorithms into two main categories: *constructive heuristics* that build a feasible solution from scratch and *improving heuristics* that start from a given solution and try to increase its quality.
While heuristic algorithms are often referred to tailored strategies to solve specific problems, meta-heuristics represents general heuristic algorithmic frameworks to be applied to different contexts [101]. The main approaches are

- *Search-based methods*, that implement an exploration strategy through the solution space in the attempt of improving the initial solution;
- *Population-based methods*, which are inspired by evolutionary mechanisms of nature generating a large set of solutions, combining them and forwarding high quality patterns to next generations of solutions;
- *hybrid* algorithms that mix one or more techniques of different methods, even from different algorithm classes e.g. a search- and population-based methods, or meta-heuristic and mathematical programming solution methods (called *matheuristics*).

In meta-heuristic frameworks, additional phases can be inserted in order to enhance the exploration of the solution space:

- *Intensification* is a procedure that puts a focus on the exploration of attractive regions in the solution space. Such regions are identified by the presence of elite solutions;
- *Diversification* is used to let the search routine visit solutions that are substantially different, so that the search covers a broader area of the solution space.

Below we outline the main heuristic algorithm used throughout the thesis, we will refer to the minimization case (once again, the maximization case can be easily derived). We assume that the objective function is computed by the evaluation function $eval(\cdot)$. In addition, for each infeasible solution S the evaluation function is such that $eval(S) = \infty$.

2.4.1 Greedy Algorithm

Suppose that the solution of an optimization problem is made of a set of elements. The *Greedy* algorithm starts from an empty (or partial) solution and a list of elements called *candidates*, then it iteratively adds the candidates one by one till it builds a feasible solution. The strategy behind such method is adding at each iteration the candidate corresponding to the best partial solution. Some other constructive methods can be derived by this algorithm, as for instance the *Randomized Greedy* that, given a $k \in \mathbb{Z}_+$, at each iteration randomly chooses the best candidate within the best k. This

strategy, especially in a multi-start approach, is useful to obtain a heterogeneous set of initial solutions.

2.4.2 Neighborhood Search

The search-based algorithms are iterative methods relying on the concept of *Neighborhood*. Given a solution S called *center* of the neighborhood, we can define a set of perturbations of S that create new solutions with the aim of finding a better one in terms of objective value. The set of perturbed solutions is a Neighborhood $\mathcal{N}(S)$ of the solution S. The *complexity* of a neighborhood is related to the number of solutions it contains, as well as to the computational effort for the neighbors evaluation. Clearly, the complexity entails repercussions on the efficiency of the exploration of a neighborhood.

Local Search

The basic search-based algorithm is called *Local Search*. This algorithm starts from an incumbent solution S, explores the neighborhood $\mathcal{N}(S)$ and updates the solution S with the best feasible solution found that improves the objective value of S. Then it repeats this procedure as long as it is can find a feasible improving solution. We show the outline of this algorithm in Algorithm 5.
This schema that chooses the best improving solution in the neighborhood

Algorithm 5: Local Search Algorithm

```
1  Initialization: start from a given solution S and a neighborhood N;
2  improved = true;
3  while improved do
4      improved = false;
5      S̄ = arg min_{S'∈N(S)} eval(S')
6      if eval(S̄) < eval(S) then
7          S = S̄;
8          improved = true;
9      end
10 end
11 return S;
```

is called *Best Improvement strategy* or *Steepest Descent*. Sometimes the exploration of the neighborhood can be too time-consuming, so one can decide to stop the exploration as soon as an improving feasible solution is found. This is called *First Improvement strategy*.

With a similar routine, one can let the search explore more than a single neighborhood, with different strategies. For instance we can let the algorithm search through all the neighborhoods for an improving solution rather than one. For sake of efficiency, neighborhood filtering techniques, called *granular searches* have been created in order to discard unpromising moves [105]. Another scheme is to consider multiple neighborhoods and explore them according to some strategy to pass from a neighborhood to another. This is called a *Variable Neighborhood Search* [50, 70].

Variable Neighborhood Descent

The *Variable Neighborhood Descent (VND)* [50] is a variable neighborhood search where the neighborhoods are switched for exploration according to a fixed order. The algorithm keeps exploring the first neighborhood and, on no improvement, it switches to the second one. If an improving solution is found in the second neighborhood, then the best solution is updated and the procedure restarts from the first neighborhood, otherwise the third neighborhood is explored, and so on. We stop when no improving solution is found among any neighborhoods. We describe the algorithm in Algorithm 6.

Algorithm 6: Variable Neighborhood Descent

```
1  Initialization: start from a given solution S and a neighborhoods
     list (N_1, ..., N_L);
2  improved = true;
3  while improved do
4      improved = false;
5      k = 1;
6      while k ≤ L do
7          S̄ = arg min_{S' ∈ N_k(S)} eval(S')
8          if eval(S̄) < eval(S) then
9              S = S̄
10             improved = true
11             Break
12         end
13         else
14             k = k + 1;
15         end
16     end
17 end
18 return S
```

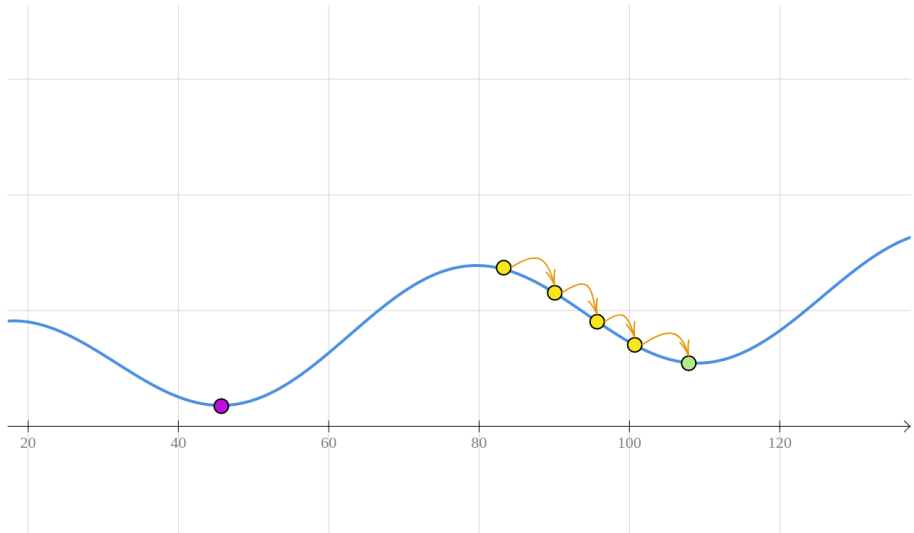

Figure 2.4: Local Search getting stuck in a local minimum (green point) instead of reaching the global minimum (purple point).

Tabu Search

The Local Search algorithm has the drawback of getting stuck at the first local minimum encountered in the neighborhood considered, as shown in Figure 2.4. Meta-heuristics often provide methods to prevent the search from an early-stage stop. The *Tabu Search* algorithm [48], whose main steps are described in Algorithm 7, pushes the exploration further, by admitting some deteriorating moves when a local minimum is encountered, in order to hopefully find better solutions later in the search. The algorithm, in order not to loop, exploits the memory to trace the last solutions visited, which are stored in a list, called *Tabu List*. The solutions in the tabu list are called *Tabu Solutions* and the exploration phase filters out them when encountered. The length of such list is called *Tabu Tenure* and is normally kept constant during the run by removing the first element of the list at each insertion of a new element. The tabu tenure must be calibrated so that cycling through some solutions is diminished, as well as avoiding to excessively narrow the neighborhoods. *Aspiration Criteria* are conditions used to prevent good solutions to be filtered out when detected as tabu. Classic aspiration criteria let the search accept tabu solutions when they are better than any other solution with some predefined attributes in common (attributes depend on the solution representation). The usual stopping criteria are quitting the algorithm after a total amount of iterations M_i or after reaching a threshold of non-improving iterations M_n. The function $insert(T, S)$ in Algorithm 7 pushes a solution S to the back of the tabu list T and at the same time pops

out the top element of T.

Algorithm 7: Tabu Search

```
Initialization: start from a given solution S, a neighborhood N and
  a tabu list T = [ ];
i = 0, n = 0;
S* = S;
while i < M_i or n < M_n do
    S̄ = arg min_{S' ∈ N(S)\T} eval(S')
    if eval(S̄) < eval(S) then
        S* = S̄;
        n = 0;
    end
    else
        n = n + 1;
    end
    T = insert(T, S̄)
    S = S̄;
    i = i + 1;
end
return S*;
```

2.4.3 Other popular Meta-Heuristic

We give a brief description of other widely-used meta-heuristic algorithms. Below a list of the most popular *search-based* methods:

- the *Simulated Annealing* (SA) algorithm [60] is based on a Local Search structure and implements a technique for escaping local optima so that deteriorating solutions are accepted with a certain probability. During the execution, such probability is more and more decreased until it falls down to 0 and only improving moves are accepted;

- the *Large Neighborhood Search* (LNS) technique [91] explores large-scale neighborhood. LNS method exploits destroy and repair heuristics in order to obtain an efficient exploration of the neighborhood. In particular, there exists a version of the LNS called the *Adaptive Large Neighborhood Search* that at every iteration selects destroy and repair heuristics from a set of heuristic algorithms based on past effectiveness;

- the *Iterated Local Search* (ILS) [67] alternates Local Search phases with perturbations phases any time the search is stuck in a local optimum,

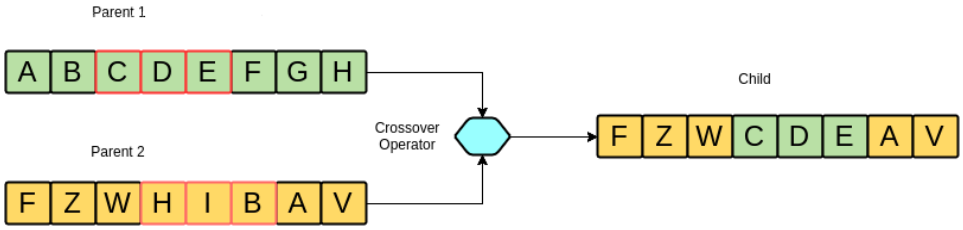

Figure 2.5: Example of crossover operator in a genetic algorithm. The crossover operator is responsible for mixing the solutions' components into a brand new individual of the next generation.

in order to make the solution jump to another region of the solution space.

Among the *population-based* meta-heuristic algorithm we mention the following:

- the *Genetic Algorithm* (GA) [62], inspired by the evolutionary law of nature, generates a population of solutions and evolves it through mutation and crossover operations that mix solutions' components, and passing over to the next-generation populations the most valuable patterns (see Figure 2.5);
- the *Ant Colony Optimization* (ACO) has been used for the solution of VRP extensions. This method reflects the ant-society system in its routine, where an ant represents a constructive heuristic guided by past information (represented by pheromone);
- in the *Scatter Search* (SS) algorithm [46], only a small subset of the population is selected for the solutions crossover. New individuals then undergo an improvement phase and then the new population is generated including high-quality and diversified solutions;
- the *Path Relinking* (PR) technique [47], originally applied to the SS, explores a trajectory in the solution space that connects two solutions. Hence, it introduces a notion of distance between solutions and, starting from an origin solution, it iteratively executes a guiding heuristic to visit new solution that minimize the distance between the target solution and the incumbent one.

Hybrid Meta-Heuristics and Matheuristics

Also *Hybrid methods* have been largely used by researchers. Hybridization has been realized between algorithms of the same class, for instance Tabu Search with Simulated Annealing, or Iterated Local Search with Variable

Neighborhood Search. Also, hybrid algorithms from different classes have been designed, one of the most wide-spread is Genetic Algorithm with Local Search or Tabu Search phase. Moreover, *Matheuristic* techniques have been used, to combine exact Mathematical Programming frameworks with meta-heuristic algorithms. In particular, a matheuristic algorithm comes from the inter-operation of metaheuristics and Mathematical Programming techniques, based on exploiting the features derived from a mathematical model of the problem, often specifically devised for its integration in a heuristic context [16]. Matheuristics mainly base the combination on two paradigms: either using Mathematical Programming to improve metaheuristics or using metaheuristics for enhancing Mathematical Programming techniques (e.g., designing a Branch-and-Price algorithm whose pricing algorithm makes use of some heuristic method).

2.4.4 Core characteristics of heuristic strategies

Here we show a list of features presented in [111] that are shared among many meta-heuristic algorithms and representing crucial characteristics for the method, in particular when VRPs are considered.

- *Solution Space*: this is the set of all solutions of the problem. Solutions may have an indirect representation, so efficient decoding algorithms are needed (see for instance [78]). Also, many algorithms make use of temporary acceptance of infeasible solutions in order to enhance the exploration of the solution space (see [110] for an analysis of infeasible solution admittance);

- *Neighborhoods*: the design of the neighborhoods explored in search-based algorithms is essential for obtaining high performance. Also, large neighborhoods, if efficiently explored, provide high-quality solutions [77]. Some methods rely on the use of multiple neighborhoods, as the Variable Neighborhood Search [70]. Moreover, granular searches is a key-feature to reduce the computational effort of the algorithm preventing the unfruitful visit of bad solutions [105], as well as neighborhood limitation based on characteristics of the solution that recently changed [74];

- *Search Trajectories*: randomization represents a core feature, for search- and population-based methods. Randomization phases increase the diversity of solutions, which lead the algorithm to a broader exploration of the solution space (e.g. see [77]). Random components may affect, for instance, the solution structure itself, or some parameters of the algorithm (i.e. the tabu tenure). The change from one solution to the next is also a peculiar characteristic related to the search trajectory. If the amount of change from one solution to the next is

relatively small, the change is called continuous, otherwise if the next solution originated is considerably different from the first one, it is a discontinuous change (as, e.g., in [5]). For instance, Iterated Local Search has a continuous search with discontinuities in correspondence with perturbation phases;

- *Memory*: collecting full or partial information on the solutions history is a very effective system to guide the algorithm towards more promising areas of the solution space. Short- and long-term memory has been widely used for such a purpose, as for example the Tabu List. In evolutionary algorithms, population is itself a memory of old solutions patterns. Moreover, memory can be centralized for parallel cooperative algorithms which need to share information with the aim of biasing each other toward better solutions [26], or adaptive, which depends on the past visited solutions [100]. Diversification and intensification can also be attained by memory usage, for instance introducing a metric on the solution space;

- *Parallelism*: parallel routines within optimization algorithms can be classified as low- or high-level parallelism. The former represents the decomposition of the algorithm into parallel independent tasks with no impact on the algorithm structure itself. This can be an asset for a more efficient evaluation phase, which is often time-consuming. The latter consists of cooperation schemes, where parallel algorithms share partial or full information through a central memory and influence each other search [26].

2.5 Machine Learning tools

Machine Learning is a field of Artificial Intelligence that aims at creating algorithms able to infer information from a set of given data and build up *predictive models* [15]. Every element of the data consists of a set of *features*, defined in a space called *feature space*, and representing the starting point to make the prediction. Moreover, the data may or may not contain, with reference to every element, the values that we want our model to predict. We call these two cases *Supervised* and *Unsupervised Machine Learning* respectively. Another distinction is relative to the domain of the predicted element: we talk about *Classification* when such elements belong to a discrete set, *Regression* if they are real numbers.
Several models have been used by researchers to obtain a high-quality solution from the predictions, such as linear model, polynomial models, logistic models, decision trees etc. In particular, Neural Networks are very complex and versatile models widely used in the sub-field of machine learning called *Deep Learning*. Each model has a set of parameters that are tuned by an

algorithm, called *learning algorithm*, the larger this set, the higher the complexity of the model. A subset of the parameters, called *hyper-parameters*, do not undergo the learning procedure, but they must be tuned before such phase.
In order to validate the prediction model, different performance indicators have been used both for classification (e.g. accuracy, precision, recall, etc.) and for regression (e.g. R-squared, mean squared error, etc.). The dataset is usually divided into two subset, the *training set*, on which we run the learning algorithm, and the *test set*, where we assess our predictions. The goal is to obtain a model with balanced error rate in the training set (*in-sample error*) and the test set (*out-sample error*). High values of the in-sample error are symptoms of the so called *under-fitting* models, whereas a small in-sample error together with a large out-sample error is defined as an *over-fitting* models. Under-fit and over-fit models are subject to bad performances in terms of prediction quality.
Among the many models available from the literature, we provide in the following a description of the ones that have been integrated in the procedures devised in this thesis as described in Chapter 7 and 8, namely Decision Tree Classifiers, Support Vector Machines and K-Means clustering.

2.5.1 Decision Tree Classifier

A well known classification algorithm is the *Decision Tree Classifier (DTC)* [97]. This algorithm repeatedly splits one component of the feature space into two components by a threshold value for a selected feature. Hence, the procedure goes on until one of the two following conditions are met:

- each component contains only elements of the same class;
- there are no more features available for further splits.

We call *impurity* the presence of elements of different classes in the same component. When choosing the feature used to split a component, the algorithm takes advantage of a measure of such impurity with the objective of decreasing it as much as possible. The main measures used to evaluate the impurity of a set of data x are:

- the *Entropy* $H(x) := -\sum_{i=1}^{K} p(x_i) \ln p(x_i)$;
- the *Gini index* $G(x) := \sum_{i=1}^{K} p(x_i)(1 - p(x_i))$.

where K is the number of element in the selected component of the feature space, x_i is the i-th element in such component and $p(x_i)$ is the probability of that element. Notice that the more homogeneous the data are in the component, the lower the value of the entropy is. DTCs have the advantage of being human readable prediction models, as they follow a binary

tree structure, and this is one of the reason why they are very popular in fields such as medical diagnosis and management. One drawback of DTCs is that the division of input space is based only on splits along the coordinate axes. Moreover, they are very sensitive to variations in the training data, so adding few new data to the training set may result in a highly different prediction model after the learning phase. In order to overcome such instability, one may adopt the *Random Forest* approach, where multiple DTCs are considered and, in the classical version, the mode or the mean prediction among all DTCs is selected.

2.5.2 Support Vector Machine

Let us consider a binary classification problem, and let us use a linear model to make a separation of the two classes. Assume that the two classes are linearly separable in the feature space, then there are infinitely many hyper-planes fulfilling the separation. A reasonable choice is to choose the hyper-plane with maximal distance from the two classes (as shown in Figure 2.6), since it is likely to be the more stable with respect to the out-sample error. This is the basic concept behind the *Support Vector Machine (SVM)* [24]. We know that a hyper-plane in the feature space is given by the equation

$$w^T x - b = 0.$$

Let us define the *label* $\lambda_i \in \{-1, 1\}$ denoting that the i-th element belongs to the first or second class. One can prove that, in a normalized data-set, the margin between the two closest elements in two different classes is equal to $2/||w||$, then finding the w that maximizes the margin is equivalent to finding the w minimum in norm such that the classes are separated. Therefore, given a data-set with I elements, we have the following optimization problem:

$$\min ||w|| \tag{2.28}$$

s.t:

$$\lambda_i(w^T x_i - b) \geq 1, \qquad i = 1, \ldots, I \tag{2.29}$$

More advanced techniques are used to manage non linearly separable data, such as the *kernel method*, where non-linear functions (called *kernel functions)* are used to map the points in the feature space in a higher dimensional space aiming for a linear separator that is then mapped back to the original space.
The Support Vector Machine can be applied also to the regression case, with a similar theory as the one developed for the classification. We have a

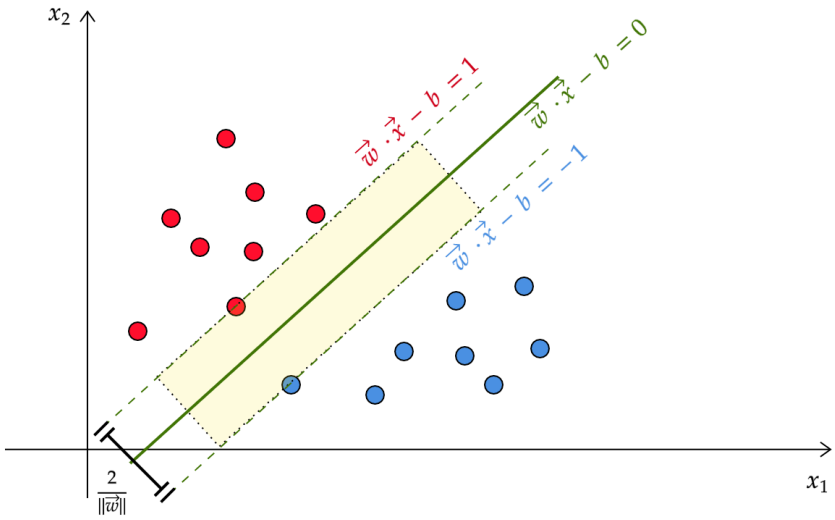

Figure 2.6: Example of SVM classification. The green line is the separating hyper-plane with largest margin between the two subsets of labelled (red and blue) data.

similar optimization problem to solve

$$\min ||w||^2 \tag{2.30}$$

s.t:

$$y_i - (w^T x_i - b) \leq \varepsilon \qquad i = 1, \ldots, I \tag{2.31}$$

$$-y_i + (w^T x_i - b) \leq \varepsilon \qquad i = 1, \ldots, I \tag{2.32}$$

where $y_i \in \{-1, 1\}$ are the target values and ε is a tolerance parameter that ensure the prediction to be at most ε far from the true value.
SVM presents high performance with linearly separable data and high-quality predictions even in high dimensional spaces. One of the main field of application is hand-written characters recognition. In the SVM training phase, outliers have a small impact as the maximal distance computation ignores them. Moreover, since the determination of the model parameters corresponds to a convex optimization problem, any local solution is also a global optimum. One drawback of the SVM is that for larger datasets, it requires a large amount of time to accomplish the training phase. Moreover, unlike other types of approach (i.e. Bayesian classifiers), SVM does not provide posterior probabilities, which are defined as the probabilities of the trained parameters in the prediction model.

2.5.3 K-Means

Clustering is an example of unsupervised machine learning. There are no labels in the data-set, but we can still find patterns in the feature space to make a classification based on the similarity in the data.
Among the clustering techniques, the *K-Means* [66] algorithm takes k as parameter that represents the desired number of clusters and partitions the feature space into k subspaces in such a way to minimize the overall squared distance between each point and the centroid of the corresponding cluster. Hence, the problem is to find a partition $\mathcal{C} := (C_1, \ldots, C_k)$ that minimizes the function

$$I(C) = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \bar{x}_i||^2,$$

where $\bar{x}_i$ the *centroid* of the the cluster C_i.
K-Means starts from a random partition and then runs a heuristic algorithm to solve this problem that repeats the following two steps:

1. compute the centroid of the each cluster C_i as the barycenter of the points in C_i, for each i;
2. assign each element x to the cluster with least squared distance from its centroid (this creates a new partition).

The problem is solved by the algorithm above within $O(tcnd)$ operations, where t is the fixed number of iterations set, c is the number of clusters, d is the dimension of the space, n is the number of points. Moreover, a technique called *k-means++* has been developed to create smart initial partitions that are proved to provide faster convergence rather than starting from a totally random set of clusters.
The K-Means clustering shows very good performance for isotropic data, which means that the extent of the variance of data is almost the same in any direction. Beside anisotropic data, another drawback of K-Means is the sensitivity to outliers. In this case, approaches such as DB-Scan [38] are more suitable. Moreover, K-Means may have convergence issues with any non-euclidean distance, whereas other clustering algorithms, such as Affinity Propagation [37], can consider any type of distance among pairs of data.

2.6 Implementation Framework

2.6.1 SCIP

SCIP (*Solving Constraint Integer Programs*) is a non-commercial suite of algorithms to solve constraint programs, mixed integer linear and non linear programs. The software has been developed by the Zube Institute of Berlin

(ZIB) and is available under academic license (for more information refer to [1]). The suite consists of the following modules:

- *SCIP:* mixed integer (linear and nonlinear) programming solver and constraint programming framework;
- *SOPLEX:* linear programming solver;
- *ZIMPL:* mathematical programming language;
- *UG:* parallel framework for mixed integer (linear and nonlinear) programs;
- *CGC:* generic branch-and-cut-and-price solver.

A user can write a model in the ZIMPL modeling language and pass it directly to one of the solvers available. As an alternative, one can interface the solvers to many programming languages through the available APIs, for instance the C++, Python and Java SCIP APIs. Moreover, SCIP supports the connection to external LP solvers, like CPLEX or Gurobi.
The framework offers a very versatile system of callbacks that let the user plug in its own code at different stages of the routines. For instance, one can dynamically create and add constraints and variables to a LP program, as well as setting branching rules for branch-and-bound approaches. In particular, one can add the variables generated by pricing algorithms in a column generation approach, as well as adopt different pricing strategies that are already implemented in the suite.

2.6.2 SciKit-Learn

SciKit-Learn is a free Python package that contains several Machine Learning algorithms [94]. The implementation of the algorithms are made on the top of NumPy [75] and SciPy [56], which are Python libraries optimized for miscellaneous linear algebra operations. SciKit-Learn offers machine learning tools in different machine learning settings:

- *Classification*: Decision Tree Classifier, Random Forests, Support Vector Classifier, Naive Bayes, K-NN etc;
- *Regression*: Linear Regressor, Polynomial Regressor, Support Vector Regressor, etc;
- *Clustering*: K-Means, DBScan, Affinity Propagation, etc.

Moreover, it offers utilities for common machine learning tasks, such as K-Fold Cross Validation according to some validation functions (e.g. accuracy, R-squared, etc.). A grid-search routine is available for hyper-parameters

tuning. Also, there are data pre-processing functions, for instance standardizers and normalizers of data, as well as encoders for categorical features. Moreover, also a Principal Component Analisys module is available, useful to compress data and obtain a dimensionality reduction of the model.

Chapter 3

Description of the problem

Freight and public transportation companies are characterized by very hard decision-making issues rising during the daily operations. The problem of planning the routes for the available fleet of vehicles is usually combined with a variety of requirements that must be simultaneously taken into account by the routing managers. On the one hand, decision makers ask for support systems able to include more and more attributes while requiring a fast solution to the incumbent routing problem. On the other hand, modern technologies providing for instance GPS, fleet tracking, real-time data sharing, create new opportunity for algorithms to enhance their performances. In this context, we consider a routing problem inspired by a small trucking company located in Padova (Venice Area, Italy), and targeting a broad class of similar cases in the transportation industry. Hereinafter, we give a verbal description of the problem, followed by a notation setting and a more formal statement.

3.1 Introduction to Vehicle Routing Problems

The problem we consider in this thesis belongs to the class of the *Vehicle Routing Problem (VRP)*. VRPs arise in public and freight transportation industries when operations managers have to organize the routes traveled by the fleet of vehicles that are available to fulfill some particular tasks (e.g. load or drop freight or people at specific locations).
In the freight context, we have a set of locations, representing customers' sites, where load or unload operations must take place according to the customers' demand. Given a set of customers and a fleet of vehicles, the basic VRP consists of finding the planning of routes starting and finishing at a specific location (called depot) so that each customer is visited by one and only one vehicle and the overall cost is minimized. Real world scenarios led the researchers to consider extra attributes of the VRP and to devise solution methods that are more and more suitable for the needs of the planning

phase. The *Capacitated VRP (CVRP)*, for instance, assigns to each vehicle a specific capacity while every customer demand comes with a required amount. Another well known extension of the VRP is the *VRP with Time Windows (VRPTW)*, where each customer must be visited within a particular time interval. Time windows are defined as *hard* if a late arrival is not accepted, whereas they are called *soft* when delay is feasible at the cost of a penalty related to the delay. The *Pickup and Delivery Problem (PDP)* is a VRP where customers demand consists of one loading (pickup) and one unloading (delivery) operation. This represents a constraint at route level that forces the pickup location to be visited before the corresponding delivery location. Also, pickup and delivery operations of a same order must be assigned to the same vehicle. Further extensions studied in literature concern the types of routes, for instance in the *Multi-Trip VRP* a vehicle can travel more than a single route, whereas the *Multi-Depot VRP* provides multiple locations as depots where a route can start and finish. The *Open VRP* allows the routes to begin and end not only at the depot but at any location.
Freight transportation companies ask for more and more attributes of the VRP to be simultaneously considered during the planning, giving rise to new problems of scientific interest. A VRP that takes into consideration at the same time multiple extensions is defined as a *Multi-Attribute VRP (MAVRP)*, or *Rich VRP*.
In this thesis we focus on a MAVRP statement that takes several attributes simultaneously into account, in order to comply with the real-life daily operations in small trucking companies offering express transportation services. This problem is the *Express Pickup and Delivery in freight Trucking problem (EPDT)* introduced at Chapter 1.

3.2 A Multi-Attribute Express Freight Transportation Problem

The problem under study rises in the routing context of small-medium trucking companies, where operations manager have to deal with daily planning of routes servicing customer requests arising during all the working day. The fleet consists of vehicles that can be different to one another. The main characteristics of vehicles are weight capacity, volume capacity, load/unload equipment (e.g. tail lift, side doors) and fuel consumption rate. A customer request comes with several characteristics that play a role in the routes planning. Each order requires pickup and delivery operations at different locations. In particular, each order may require one or more pickup operations and/or one or more delivery operations. Each operation is associated with one time interval and date, previously agreed between the two parties, within which such operation should be performed. Each order includes op-

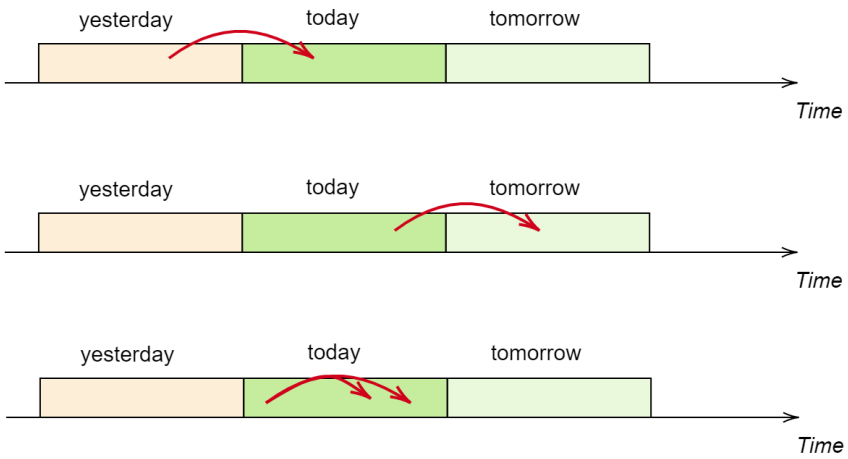

Figure 3.1: Examples of orders spanning multiple days.

erations that span a single day or two consecutive days. In Figure 3.1, we provide three examples where *today* is the occurring working day, which is the object of the planning. Tails of arrows represent pickup tasks, heads stand for deliveries, colors distinguish the working day duration for each day. From the top to the bottom of the figure, we show (i) an order with one operation to be executed yesterday and one to be executed today, (ii) an order with one operation to be executed today and one to be executed tomorrow, and (iii) an order with three operations for today.

A service time is associated with the loading operations and special loading requirements may be required (e.g. tail lift, crane, etc.). The requests come with a revenue that let the decision makers decide whether the order can be conveniently satisfied or is better discarded for cost-savings purposes. In some case, it may be possible to interact with the client in order to define a more convenient cost, if necessary. A urgency level is associated with orders: normal orders may be rejected at no cost and may be handled so that operations are not required to fall within the predefined time interval, but a delay is accepted at a specific cost for the customer inconvenience. Urgent orders are defined similarly to normal orders but rejecting them implies a cost. Finally, mandatory orders cannot be rejected and delays with respect to the time intervals are not allowed.

Due to the just-in-time nature of the orders, when an order is issued, customers are not always aware of where each item to delivery will be actually available for the pickup operation. For example, in the manufacturing industry, the production of multiple items will be performed in more than

one factory. The client may decide on how to distribute the production depending on factors that are not available at the time the order is issued, but will become clear later during the working day (e.g., production in one establishment has been faster, so machines for specific items are available there earlier than in the other facilities). Typically, a client knows the items to be delivered, divided by destination (delivery points) and, in order to get a reservation in time, it issues an order before the information on how the production will be exactly divided among the factories. This allows reserving the necessary capacity in time, whereas the specific items to pickup at each factory will be defined later. As a consequence, orders cannot be split into multiple orders containing a single pickup and a single delivery operation, nor they can be served by different vehicles. This nature of orders implies also that, by the transportation company point of view, no complete knowledge is available during the planning phase about the destination of each item at the pickup points, so that the routes must be planned in such a way that, for each order, all pickup operations are executed before any delivery operation, with benefits from the operations and from the accounting sides. In Figure 3.2, for example, the routes contain an order (in orange) with two pickups (P) and three delivery (D) tasks. In (a), the route is feasible, whereas in (b) the precedence policy defined above is violated since one delivery task (marked with the red cross) is performed before completing all pickups. Numbers represent the amount of items to be transported, supposing 100 items in total for that order. In the pickup nodes the vehicle must load 40 and 60 items, whereas in the delivery points the items to unload are 20, 30 and 50. Observe that, without complete information on the destination of each item, one ore more item among the 20 to be delivered in the first delivery of (b) may be part of the 60 items available at the following pickup node, or even spread between the 40 and 60 items in the two pickup nodes.

Drivers are normally not required to start nor to end their trip at the depot. Sometimes, for instance for maintenance purposes, the routes must finish at the depot, or also, in order to offer a better starting position for the day after, routes may preferably finish in some predefined location. The route design must be carried out taking into account the European Hours of Service Regulation, so that drivers must respect mandatory breaks along their assigned trips: the crew is subject to the *European Hours of Service Regulations* (No. 561/2006, Directive 2002/15/EC). These laws imply constraints on the rest duration and cumulative drive and work time of a driver. The regulation consists of rules (and consecutively constraints) based on the activity of drivers spanning different intervals of working days, from 4 months to 1 day before the incumbent day. Among all the rules, we define below the most significant for daily optimization purposes:

- after at most 4:30 hours of uninterrupted driving, a compulsory break

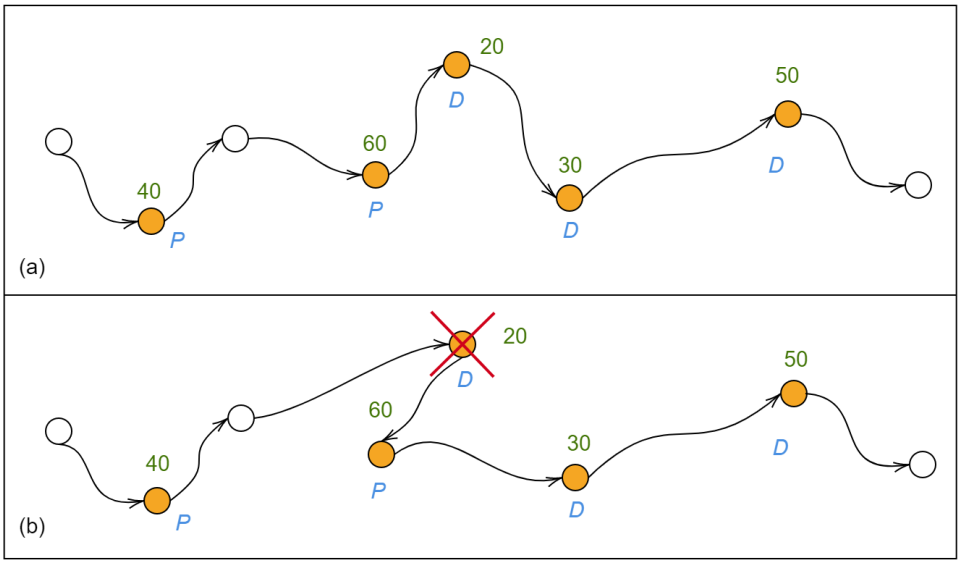

Figure 3.2: Example of valid and invalid route for an order (orange nodes).

must be taken;

- the compulsory break duration is 45 minutes. It can be split into a first break of 15 minutes and a second break of 30 minutes;
- the maximum amount of drive time in a day is 9 hours (twice a week it can be extended to 10);
- the maximum amount of work (driving and operational) time in a day is 13 hours (twice a week it can be extended to 14);
- the minimum amount of night rest is 11 hours (under some conditions it may be reduced to 9 hours);
- the minimum weekly rest period is 45 hours (if compensated by previous rests longer than the minimum required, it may be reduced to 24 hours).

From the transportation company point of view, the objective is the maximization of the net profit, defined as the sum of the revenues from satisfied orders minus the overall costs, together with preference matching. In addition, further preferences play a role with the aim of improving the quality of service. A maximum duration of route may be preferred, e.g., to accommodate specific drivers' requests, or balance drivers' workload in the long term. Moreover, a benefit for including predefined subsets of orders in the same route is considered, in order to preserve some effective order assignment combinations. For instance, groups of orders may be easier to pack,

load or unload if in the same vehicle, facilitating the operations executions. The assignment of orders to specific vehicles can also be expressed as a preferred outcome, aiming at helping pickup or delivery tasks by truck facilities, although not necessary. Finally, a preferred minimum number of vehicles available at the depot at the end of the planning horizon may be declared, as an advantage when dealing with just-in-time requests.
In the following, we provide a more detailed analysis of the problem described in this section, thus providing a more formal definition of EPDT.

3.3 Entities and attributes

The main attributes of EPDT are related to positions, vehicles, orders and route.

3.3.1 Positions

Positions or *locations* represent the pickup and delivery points and include the depot, corresponding to the main facility of the transportation company. For each ordered pair (i, j) of positions, a distance d_{ij} and a travel time t_{ij} are given. They depend on the road network status and up-to-date information can be retrieved from, e.g., web services or specialized applications like [49, 102].

3.3.2 Vehicles

The scenario described in Section 3.2 involves vehicles with different features, that is a *heterogeneous fleet*. For instance, vehicles may be characterized by multiple loading capacities (weight, volume) and operational costs. In particular, denoting by V the set of vehicles, each $v \in V$ defines a constraint on the maximum volume U_v^1 and a maximum weight U_v^2 that can be loaded on it. Since freights are normally trucked on pallets, U_v^1 and U_v^2 provide a reliable estimation of the actual vehicle loading capacity. We remark that, in our problem, capacity relates to on board freight, meaning that deliveries make capacity available again for following pickups. Moreover, vehicle-specific variable and fixed operational costs are provided. The former is a cost per distance unit C_v^U, taking into account several factors among which fuel consumption, vehicle mortgage, maintenance costs etc. The latter represents the fixed cost C_v^F incurred if vehicle v is deployed. Some vehicles are equipped with special loading facilities, such as crane, tail lift, side doors etc.

3.3.3 Orders

Customer *requests* (or *orders*) correspond to the demands of pickup and delivery services. Orders specify a set of pickup and delivery tasks. In particular, an order may include more than one pickup and more than one delivery task (multi-pickup / multi-delivery orders). In this case, more pickup tasks and/or more delivery tasks are defined, and all the pickup tasks must be executed before all the delivery tasks by the same vehicle. Each task is defined in terms of position, size of the freight (volume and weight), time window and service time. Time windows are specified by a time interval within which the operation should be performed. Time windows can be hard or soft. The former imply that the task must be executed within the time window, otherwise the solution is not feasible; the latter can be violated at the cost of a penalty L_i^w proportional to the time window violation at the task i, taking into account that early service is not allowed (in case a vehicle arrives early at the task position, it waits until the beginning of the time window). A revenue P_o is associated with each order $o \in O$, O denoting the set of orders, together with a priority. We have three possible priority levels: mandatory orders (subset O^M) must be fulfilled in any feasible solution; urgent orders (O^U) define a penalty in case they are note executed (the penalty for $o \in O^U$ is denoted by L_o); normal orders (O^N) can be rejected at no penalty. Finally, each order may specify the required loading facilities, if any, affecting the set of vehicles it can be assigned to. We remark that, due to the nature of customer requests demanding for an express delivery service, the tasks' time windows of one order fall in at most two consecutive days. As a consequence, the set of issued orders that are relevant for planning in a given day, hence included in O, are the ones issued either the same day or the day before. The time windows of the tasks in O fall in the day before (we call them *yesterday tasks*), in the same day (*today tasks*) or in the next day (*tomorrow tasks*).

3.3.4 Routes

A *route* is defined by a sequence of pickup and/or delivery tasks assigned to a specific vehicle. All the tasks of the same order must be assigned to the same vehicle. In order for a route to be feasible, for each assigned order, all pickups tasks must precede all deliveries in the sequence. The planning horizon is one working day and routes are open: each vehicle starts from the position of the last task executed the day before, in case with pending *yesterday* tasks, it services *today* tasks, and it finishes at the position of the last today task or, in case, of the last pending *tomorrow* task if any. Moreover, some routes are preferred or required to reach a prescribed *end-of-day* position, e.g. for maintenance issues. Further attributes are related to a preferred maximum duration of route, the inclusion of predefined subsets

of orders in the same route, the assignment of orders to specific vehicles and a preferred minimum number of vehicles available at the depot. Moreover, a *prospective route cost* is taken into account in assessing routes quality as a proxy of the variable next-day cost. In fact, although the planning horizon has a daily span, the succession of working days asks for good initial state as a consequence of the end-of-day vehicle status of the day before.

3.4 Problem definition

A schematic definition of the Express Pickup and Delivery in freight Trucking problem is stated as follows:

- *Input:*
  - the orders with at least one task in a one-day planning horizon;
  - the initial vehicle status in terms of position and on-board pending yesterday tasks;
  - the travel and time distance between positions;
- *Output:* a solution defined as a set of daily routes to service the today tasks of a subset of selected orders;
- *Hard Constraints:* a feasible solution must respect

  H1 vehicle capacity;
  H2 precedence between pickup and delivery tasks;
  H3 hard time windows;
  H4 execution of mandatory orders;
  H5 loading facilities requirements;
  H6 hours-of-service regulations;
  H7 prescribed route ending position (if provided).
- *Soft Constraints:* a feasible solution should preferably respect

  S1 maximum route duration;
  S2 preferred subset of orders in a same route;
  S3 preferred minimum number of vehicles at the depot;
  S4 preferred order-to-vehicle assignment;
  S5 preferred end-of-day position;
- *Objective function:* the goal is to optimize profits and revenues of different types by combining the following objectives:

  O1 total revenue of satisfied orders maximization;

O2 routes cost minimization;

O3 prospective routes cost minimization;

O4 missed urgent orders cost minimization;

O5 soft time windows violation cost minimization.

In Table 3.1 we summarize the notation related to EPDT.
By the definition of the problem, we remark that the cost $C(r)$ of a route r includes the fixed deployment cost C_v^F and the variable cost depending on C_v^U and on the length $l(r)$ of r up to the last *today* task

$$C(r) = C_v^F + C_v^U \cdot l(r). \tag{3.1}$$

We define $A(r)$ and $W(r)$ as, respectively, the prospective route cost function and the cost for time windows violation. Given the set of pickup tasks N_p and delivery tasks N_d, we introduce the complete graph $G = (N, A)$ where $N = N_p \cup N_d$ and A is the set of all arcs between each pair of nodes in N.
Denoting by s a solution of EPDT, we introduce the variables $y_r \in \{0, 1\}\ \forall\ r \in \Omega^v, v \in V$ that are equal to 1 if route r belongs to the solution s, 0 otherwise.
We can view EPDT as a maximization problem where we want to optimize multiple criteria O1 - O5. Because of the different nature of the criteria, the problem may be tackled, e.g., by searching for the Pareto frontier, thus providing the decision maker with a set of alternative solutions. Weights can be adjusted on different orders of magnitudes to represent a hierarchy among different criteria. In our model, we include a combination of such criteria into a single-objective function. In particular, following the suggestion of the transportation company, the combination represents a net profit given by the revenues from satisfied orders (objective O1) minus the costs of today routes, prospective tomorrow routes, missed urgent orders and soft time windows violations (objectives O2, O3, O4 and O5). Moreover, by denoting with s a solution of EPDT, we introduce a penalty term $I_k(s)$ for each soft constraint Sk, with $k \in K$ and $K = 1, \ldots, 5$, together with a weight coefficients w_k that give more or less importance to each term.
Summarizing, the value of the objective function for s is represented by the net profit, penalized by the missed preferences:

$$\sum_{o \in O(s)} P_o - \sum_{v \in V} \sum_{r \in \Omega^v} (C(r) + A(r) + W(r)) y_r - \sum_{o \in O^U \setminus O(s)} L_o - \sum_{k \in K} w_k I_k(s),$$

We search for a solution s and related variables y_r such that the number of routes does not exceed the fleet size:

$$\sum_{v \in V} \sum_{r \in \Omega^v} y_r \leq |V|,$$

Table 3.1: Notation for EPDT.

	Positions
N	set of positions
d_{ij}	space distance between positions $i \in N$ and $j \in N$
t_{ij}	time distance between positions $i \in N$ and $j \in N$
	Vehicles
V	set of vehicles
U_v^1	volume capacity of vehicle $v \in V$
U_v^2	weight capacity of vehicle $v \in V$
C_v^U	cost per distance unit associated with vehicle v
C_v^F	fixed cost associated with the deployment of vehicle v
	Orders
$o(i)$	order in O that contains the task i
O	the set of orders in the problem instance
O^N	subset of normal orders ($\subseteq O$)
O^M	subset of mandatory orders ($\subseteq O$)
O^U	subset of urgent orders ($\subseteq O$)
$O(s)$	set of orders in O satisfied by solution s
P_o	revenue of of an order $o \in O$
L_o	penalty for missed urgent orders $o \in O^U$
L_i^w	penalty for soft time window violation at task i
$\mathcal{T}(o)$	set of tasks related to order o
yesterday task	pickup or delivery operation with time window in the day before
today task	pickup or delivery operation with time window in the current day
tomorrow task	pickup or delivery operation with time window in the day after
	Routes
Ω_v	set of routes for vehicle v, feasible w.r.t. H1-H7;
$C(r)$	cost of route $r \in \Omega_v$
$A(r)$	prospective cost of route $r \in \Omega_v$
$W(r)$	cost for soft time windows violation of route $r \in \Omega_v$
D	maximum route duration
$I_k(s)$	penalty term for solution s associated with the k-th soft constraint
w_k	weight parameter associated with penalty term I_k

all mandatory orders are executed:

$$O^M \subseteq O(s),$$

and all the tasks in the orders $O(s)$ that are part of the solution are visited by exactly one route:

$$\sum_{v \in V} \sum_{r \in \Omega^v} a_{ir} y_r = 1 \quad \forall\, i \in N : o(i) \in O(s),$$

where parameters a_{ir} $\forall i \in N, r \in \Omega_v$ is equal to 1 if route r visits node i, 0 otherwise.
Based on the observation above, we introduce a preliminary model to formalize the definition of EPDT. To this aim, we define for each non-mandatory order $o \in O \setminus O^M$ the binary variables x_o equal to 1 if and only if o is assigned to no route in the solution. Furthermore, we define q_o as the objective coefficients of variables x_o, such that $q_o = P_o$, if $o \in O^N$, and $q_o = P_o + L_o$, if $o \in O^U$, being P_o and L_o the profit of order $o \in O$ and the cost for missed urgent order $o \in O^U$, respectively.In conclusion, we can model EPDT as a maximization problem:

$$\max \sum_{o \in O} P_o - \sum_{v \in V} \sum_{r \in \Omega^v} (C(r) + A(r) + W(r)) y_r - \sum_{o \in O \setminus O^M} q_o x_o - \sum_{k \in K} w_k I_k(s) \tag{3.2}$$

$$\text{s.t.} \sum_{v \in V} \sum_{r \in \Omega^v} a_{ir} y_r = 1 \quad \forall\, i \in N : o(i) \in O^M \tag{3.3}$$

$$\sum_{v \in V} \sum_{r \in \Omega^v} a_{ir} y_r + x_{o(i)} = 1 \quad \forall\, i \in N : o(i) \in O \setminus O^M \tag{3.4}$$

$$\sum_{v \in V} \sum_{r \in \Omega^v} y_r \leq |V| \tag{3.5}$$

$$y_r \in \{0, 1\} \quad \forall\, r \in \Omega^v, v \in V \tag{3.6}$$

$$x_o \in \{0, 1\}, \quad \forall\, o \in O \setminus O^M. \tag{3.7}$$

The solution s is of EPDT is determined by the values of variables y_r and x_o. The objective function (3.2) combines the net profit and soft constraints violations, constraints (3.3) require each task of a mandatory order to be served by exactly one vehicle, whereas constraint (3.4) allows tasks from non-mandatory orders to be either visited by exactly one vehicle, or considered as a rejected order. Finally, constraint (3.5) ensures that the solution does not involve a number of vehicle that exceeds the fleet size.
Notice that the proposed formalization makes use of feasible routes that are implicitly defined by the sets Ω_v. Moreover, the penalty terms related to soft constraints are not expressed in closed form, since they depend on s,

and a specific model to represent prospective cost is needed: in Chapters 5 and 6 we will discuss how they can be represented.
The definition of EPDT reported in this chapter encompasses several attributes coming from real-word contexts. As we will discuss in the following Chapter 4, EPDT presents, to the best of our knowledge, interesting innovative features. For example, multi-pickup and multi-delivery orders are defined as a combinations of decoupled pickup and delivery tasks, so that they cannot be split into single pickup-delivery orders and special precedence constraints have to be considered. Another interesting feature, is the span of orders that may include tasks occurring in different planning days, thus generating pending orders to be considered at the beginning and at the end of the planning horizon.
In the next chapter we make a literature review of exact and heuristic algorithms for several VRP extensions, with the aim of stating the relevance of EPDT and analyzing state-of-the-art approaches to cope with VRPs similar to it.

Chapter 4

State-of-the-Art for Multi-Attribute Vehicle Routing Problems

This chapter is devoted to the characterization of the VRP and its extensions, with an insight both in mathematical formulations and in the most popular solution methods offered by the scientific literature. In Section 4.1, we recall different models for the basic VRP and, in Section 4.2, the main VRP extensions, remarking different classes of attributes and their impact on the problem. When such additional features come in a significant number, the problem is a MAVRP (see definition at Section 3.1). In Section 4.4, we outline the state-of-the-art exact approaches to different types of MAVRPs. In Section 4.5, we give a description of an exact method for VRPs that has been the base for algorithms solving MAVRPs in several literature works, as well as for the Column Generation algorithm we devise to solve the problem stated in Chapter 3, as we will detail in Chapter 6. Section 4.7 contains a review of the main heuristic and meta-heuristic methods for the solution of MAVRPs, followed by a description of classic VRP heuristic algorithms in Section 4.6. Finally, in Section 4.8.3, we review approaches proposed by literature to handle Dynamic and Stochastic versions of VRPs.

4.1 VRP definition and basic models

In this section we give a description of the Vehicle Routing Problem in its basic form. According to the statement in [106], the problem can be defined as follows:

> given a set of *transportation requests* and a *fleet of vehicles*, the problem is to determine a set of vehicle *routes* to perform all (or some) transportation requests with given vehicle fleet at *mini-*

mum cost; in particular, decide *which vehicle handles which requests in which sequence* so that all vehicle routes can be feasibly executed.

The problem arises from several real-world scenarios from the freight transportation industry. The high complexity of the problem motivated a very intense and active research, and nowadays many challenges are still a fertile ground to new progresses.
A more formal statement of the problem considers a set of n_I points $I = \{1, \ldots, n_I\}$ called *customers*, each of which needs to be visited by a single vehicle, a particular point 0 called *depot* where a *fleet* of n_K identical vehicles $K = \{1, \ldots, n_K\}$ is available. The route traveled by a vehicle must start at the depot, visit the set of associated nodes in an order and finish at the depot, hence a route can be defined as a tour containing the depot. Finally, there is a fixed cost c_{ij} to move from point i to point j.
This description of the problem leads naturally to the same definition on a graph structure. In the following we report different mathematical models for the VRP [106]. We define $N = I \cup \{0\}$ as the set of nodes and $A = \{(i, j) : i, j \in N\}$ as the set of arcs, with a cost coefficient c_{ij} related to each arc. Therefore, a VRP instance is defined by a complete graph $G = (N, A, c_{ij})$ and the available fleet K. In this context, a route is defined as a sequence of nodes $r = (i_0, i_1, \ldots, i_s, i_{s+1})$ where $i_0 = i_{s+1} = 0$ and a set of s customers associated with the route r is visited. Such route is said *feasible* if $i_m \neq i_l$ for any $1 \leq l < m \leq s$. A *feasible solution* is defined by a set of feasible routes $S = \{r_1, \ldots, r_{|K|}\}$ such that the corresponding sets of customers' nodes visited by each single route give a partition of the set I.

4.1.1 Mathematical formulations

VRP can be formulated as a MILP problem. There are different types of models, based either on arc-flow on a network or on a set-partitioning formulation.

2-index Arc-Flow Formulation

The formulation relies on the concept of flow on a network, and consists of two sets of indices on the node set N. This is called *2-index Arc-Flow*

Formulation

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.1}$$

$$s.t.$$

$$\sum_{i\in\delta^-(\{j\})} x_{ij} = 1 \qquad \forall j \in N \setminus \{0\} \tag{4.2}$$

$$\sum_{j\in\delta^+(\{i\})} x_{ij} = 1 \qquad \forall i \in N \setminus \{0\} \tag{4.3}$$

$$\sum_{j\in\delta^+(\{0\})} x_{0j} \leq |K| \tag{4.4}$$

$$\sum_{j\in\delta^+(S)} x_{ij} \geq 1 \qquad \forall S \subseteq N \setminus \{0\}, S \neq \emptyset \tag{4.5}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A, \tag{4.6}$$

where the binary variable x_{ij} takes value 1 if the arc (i,j) belongs to the solution, 0 otherwise. We remind that given a node i, $\delta^+(S)$ and $\delta^-(S)$ are the *forward and backward cuts* of the subset of nodes S, that are the set of out-going and in-going arcs respectively with exactly one end in S.
The objective function (4.1) represents the minimization of the overall cost of the solution. Constraints (4.2) and (4.3) ensure that each node is visited by exactly one vehicle, and they are called *flow conservation constraints*. The number of vehicle used in the solution does not exceed the fleet size by constraint (4.4). Finally, constraints (4.5) are the *sub-tour elimination constraints*, necessary for each route to contain the depot node. The idea behind such constraints is that for each subset of nodes other than the depot, there must be at least one route that enters (and exits, by (4.2) and (4.3)) the subset $S \subseteq N \setminus \{0\}$. In other words, no route can loop on a subset of nodes if it does not visit the depot.

3-index Arc-Flow Formulation

The previous model does not need arcs in the solution to be explicitly related to some particular vehicle. As we will see, sometimes there may be other attributes to plug into the model that are vehicle-specific, for instance in the case of a heterogeneous fleet. When this is the case, a better problem

modeling is provided by the *3-index Arc-Flow Formulation*:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{4.7}$$

$$s.t. \sum_{k \in K} \sum_{i \in \delta^-(\{j\})} x_{ij}^k = 1 \qquad \forall j \in N \setminus \{0\} \tag{4.8}$$

$$\sum_{k \in K} \sum_{j \in \delta^+(\{i\})} x_{ij}^k = 1 \qquad \forall i \in N \setminus \{0\} \tag{4.9}$$

$$\sum_{k \in K} \sum_{j \in \delta^+(\{0\})} x_{0j}^k \leq |K| \tag{4.10}$$

$$\sum_{j \in \delta^+(S)} x_{ij}^k \geq 1 \qquad \forall S \subseteq N \setminus \{0\}, S \neq \emptyset, k \in K \tag{4.11}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall (i,j) \in A, k \in K, \tag{4.12}$$

where binary variables x_{ij}^k are equal to 1 if arc (i,j) is traversed by vehicle k in the solution, 0 otherwise. This type of formulation allows to enrich the model with attributes at vehicle level, e.g. a capacitated VRP where each vehicle has its own capacity. However, the size of the model has increased both in number of variables and in number of constraints, involving a loss of efficiency when solving the MILP.
Let us observe that both in 2- and in 3-index formulation the number of sub-tour elimination constraints is exponentially large, therefore a suitable solution technique is needed in order to efficiently solve this problem. A classically used method to handle these constraints is based on a Branch-and-Cut approach, where the problem is relaxed by removing all the sub-tour elimination constraints. Successively, we solve the LP relaxation in a Branch-and-Bound scheme. If one of the sub-tour elimination constraint is violated, the corresponding separating cut is added to the model. The procedure is repeated until a feasible solution is found.

Set Partitioning Formulation

A third model of the problem is based on a *Set Partitioning Formulation*. We define Ω as the set of *all* feasible routes. We associate a cost c_r to each route $r \in \Omega$. Then, we define the parameters a_{ir} that assume value 1 if node i is visited by route r, 0 otherwise. Finally, the binary variables x_r is 1 if

route r is part of the solution, 0 otherwise.

$$\min \sum_{r \in \Omega} c_r x_r \tag{4.13}$$

$$s.t. \sum_{r \in \Omega} a_{ir} x_r = 1 \qquad \forall i \in N \setminus \{0\} \tag{4.14}$$

$$\sum_{r \in \Omega} x_r \leq |K| \tag{4.15}$$

$$x_r \in \{0, 1\} \qquad \forall r \in \Omega \tag{4.16}$$

The objective function 4.13 represent the total cost of the solution, constraints (4.14) ensure that each node is covered by exactly one route and constraint (4.15) forces the number of routes to be less than or equal to the number of available vehicles.
This model is proved to be related to the arc-flow formulations by the Dantzig-Wolfe decomposition (see e.g. [32]). In fact, every vertex of the polyhedron defined by the continuous relaxation of the problem corresponds to a feasible route in the graph. Despite the simplicity of this formulation, one can observe two main issues:

- it is not trivial to know which are the feasible routes;
- the number of feasible routes (which in turn is the number of variables) is exponentially large.

Column Generation and Branch-and-Price approaches are suitable methods to deal with models with this structure. The Pricing Problem is related to the Shortest Path Problem and can be solved by Dynamic Programming in pseudo-polynomial time, as we will describe in Section 4.5.

4.2 VRP extensions in the literature

VRP attributes are intended as features that are embedded in the basic definition. In the following, we will se a classification for the main extensions of both the VRP and its objective, with reference to [106].

4.2.1 VRP attributes classification

Real-life problems show several attributes that have to be taken into consideration in the planning phase. Therefore, the basic VRP definition is very often extended according to one or more characteristics in play.
The VRP attributes can be grouped into different classes, according to the way they impact the definition of the problem. Researchers have provided different classifications, as for example in [7, 58, 106]. Based on the classification presented in [106], we gather the main MAVRP attributes in the following classes:

- Network structure;
- Fleet composition;
- Type of requests;
- Intra-route features;
- Inter-route features.

Hereinafter, we give a short description of the classes above, exploring examples of features rising from real-case scenarios that have been studied by researchers.

Network Structure

The nature of the network can influence the graph underlying the model. One main characteristic is whether the distances c_{ij} are symmetric with respect to the endpoints i and j. In the symmetric case we can model our problem on an undirected graph, since $c_{ij} = c_{ji}$, otherwise we need a directed graph.
Another feature concerns what has to be visited by the vehicles. We can represent our tasks as operations to be served at some specific locations, that is known as *Node Routing Problem* (this corresponds to the classical VRP). Sometimes it is more convenient to model our tasks as arcs to be visited, for example street cleaning, winter services with salt gritting, etc. In this case the problem is called *Arc Routing Problem*.
There exists also examples of mixed Node and Arc Routing Problems called *General Routing Problems* [106].

Fleet Composition

In Chapter 3 we have seen fleet features like heterogeneity, multiple capacities and cost per space distance (see Section 3.2). Further characteristics can be related, e.g., to the shape and size of each vehicle. Another peculiar feature is the presence of *multiple depots*. In this case vehicle can start and finish at different locations, as long as they correspond to a depot.
Moreover, in some applications the vehicles can be *compartmentalized*, so that different classes of items are allowed only within specific compartments. This indeed induces compatibility constraints between items transported by the same vehicle. As an example, think about waste collection, or food requiring different temperatures or dangerous goods.
Compatibility can rise also between vehicle and customer, for instance the vehicle size may be not suitable to enter or make maneuvers in some specific customer location.

Some more complex fleet variant is given by the presence of trailers, where vehicles need to be joint to a self-moving vehicle. The *truck-and-trailer* routing problem is composed by a class of customers that cannot be reached by trucks with trailers (called *truck customers*), therefore single trucks can perform sub-tours to reach trailers, transfer items and accomplish the operations at the truck customers. In the *trailers and transshipments* scenario a trailer may be assigned to different trucks on parts of the route.

Type of Requests

As we have seen in Section 3.2, operations related to a particular customer request can be *pickup* or *delivery* tasks, where items are, respectively, loaded to or unloaded from the vehicle. A particular case is the one of *Pickup and Delivery* routing problems, where customer requests consist of both a pickup operation and a delivery operation. This induces a precedence constraint, so that a pickup must be performed in a route before the corresponding delivery. Similar models are used in public transportation, and are known as *dial-a-ride* problems.
A different way of requests handling is the *VRP with backhauls*. In this case a subset of customers (linehauls customers) must be visited for delivery operations. Afterwards, pickup operations are performed at the remaining customers (backhauls customers).
Finally, in some contexts a request can be fulfilled by more than one vehicle: the items of a request can be transferred at a transit-point, where *consolidation* operations can be executed. Notice how the multi-pickup and multi-delivery type requests defined in 3.3 cannot be associated with any of the precedence constraints found in literature and therefore represent a brand-new attribute.

Intra-Route Features

Among the main intra-route characteristics, we mention the presence of (soft or hard) *time windows*, *on-board capacity*, the *service time* and the *European Hours of Service Regulations*, as described in Section 3.2.
In some scenarios, one can consider as decision variables the *waiting times*, defined as the amount of times the vehicle is kept idle at the locations along the route. Hence, waiting times can be used to schedule the arrival to the next assigned location at a more convenient time (as we will see in Section 4.8.2, this may be relevant with stochastic customers).
Another extension of the request type is related to the possibility to split or not split the goods that have to be delivered to some customer. Indeed, if visiting a customer with more than one vehicle or tour to accomplish the same request is allowed, a larger set of feasible plannings is available which may involve more cost-savings rather than denying splits. Not all scenarios

are suitable for splitting, for instance because of the lack of information on customer requests, as we have seen in Section 3.2.
Items can be also characterized by 2 or 3 dimensional size, that induces the capacity constraint to take into account a packing problem at each load and unload operation.
Also, goods can be subject to a *loading policy*, for instance a LIFO (Last In First Out) can be required when a truck is not equipped with side doors and therefore can be unloaded only from the rear. Also, as described in Section 3.2, loading policies may involve subset of orders which are required or preferred to be together in a vehicle for, e.g., packing compatibility.
In the classic VRP, routes are tour containing a depot location. The *Open VRP* allows vehicles to start and end at any location, so routes may be non-closed paths.
Constraints related to the maximum duration of a route in terms of space and time can be added. As an example, VRPs with deteriorating food delivery need these type of constraints.
In the *Multi-Trip* VRP, a vehicle can travel multiple routes, for example to refuel or reload operations at the depot.

Inter-Route Features

More complicating scenarios are based on *synchronization* issues. For instance, we can have a shared finite resource (i.e. trailers available, or maximum number of long routes), or a balancing requirements for fairness issue, so that the difference between the longest and shortest route cannot exceed some threshold. Synchronization also takes place in *scheduling and routing* problems, where crew must be assigned to a vehicle and, at the same time, a route planning must be decided.

4.2.2 Extension of the objective function definition

The most classical objective function is related to the overall distance minimization, but other criteria can be considered, like customer satisfaction, overall time duration, number of vehicles, environmental impact or maximum route duration. When dealing with soft time windows, also penalties for late arrivals are considered. Other type of preferences about specific attributes can be added to the evaluation phase of the solutions in the incumbent VRP [106].
The planning evaluation can be performed as a single objective function or modeled as a multi-objective problem. Also, a hierarchical evaluation can be performed on the objective function components. As an example, a well-known hierarchical approach is at a first level the minimization of the number of deployed vehicles and, at a second level, the cost of routes.
In the *VRP with profits* a revenue is associated with the fulfillment of each

request. It is worth to notice that in this extension of the VRP, a feasible solution does not require to accomplish all the requests, since the objective in such optimization problem is to maximize the overall profit, regardless of dismissing some requests.

4.3 Comparison with EPDT

The surveys cited in Section 4.2 show that several attributes of EPDT, the problem we described in Chapter 3, are the object of many studies in literature. For instance, a large number of works consider heterogeneous fleets, vehicles characterized by capacities and space distance costs. Concerning route attributes, several papers consider scheduling breaks to comply with the drivers' hours of service regulation, as well as open routes. Moreover, soft and hard time windows are popular type of attributes widely studied, as well as service times at customer facilities.
Although many of its attributes can be found in the VRP literature, EPDT presents some innovative aspects.
A first new attribute comes from the definition of the orders. EPDT defines, to the best of our knowledge, a new type of multi-pickup/multi-delivery orders, made of a combination of several pickup and several delivery operations where no direct correspondence between possible pairs of single pickup and delivery tasks can be established at the planning time. This is due to the particular context defined by express freight transportation with regional haul, where, as described in Section 3.2, the exact destination of the items may be defined at the time of the picking up and unknown at the planning time, which may take place several hours before.
This aspects reflects on a new route attribute in terms of precedence constraints. In Section 3.2, we have seen that feasible routes must be planned such that they visit all pickup nodes before visiting any delivery nodes: a route that does not comply with this precedence constraint may have already visited the destination node of an item, which is revealed the pickup time.
Another relevant innovative feature of EPDT is, to the best of our knowledge, the presence of pending tasks among the route attributes, as described in Section 3.2 and stated in Section 3.3.4. In fact, orders span two consecutive days, whereas the planning horizon considers the tasks of one day, or less, as we will see for re-optimization in dynamic settings. This generates pending yesterday and tomorrow tasks that have an impact on the planning horizon.
Further interesting aspects are related to the presence of new specific preference attributes (e.g. collecting order subsets on a same route, having a preferred number of vehicles at the depot at the end of the planning horizon), as well as the combination of several attributes issued by real world

contexts.

4.4 Exact Methods for MAVRPs

A large variety of solution methods for MAVRPs have been devised by researchers. In the following, we provide some of the works in the Operations Research literature devoted to exact algorithms for VRPs with different attributes.
A real-world MAVRP is solved in [20]. The problem involves, among others, heterogeneous fleet, time windows, multiple capacities, hours-of-service regulations, maximum route length, open routes, split-delivery, client-vehicle compatibility constraints. The authors adopt a Column Generation approach. The Master Problem relies on a set-covering formulation, whereas the Pricing Problem is a particular type of ESPPRC solved through a bounded bidirectional dynamic programming algorithm. The pricing procedure is split into three phases for the generation of different types of variables. At the end of the phases, a heuristic algorithm produces an integer solution starting from the solution of the continuous relaxation of the problem, then a MILP solver is run to reach optimality.
In [86], a VRP with Pickup and Delivery and Time Windows is considered. A branch-and-cut-and-price algorithm has been implemented. The solution method relies on a set partitioning model. Two pricing problems, an elementary and non-elementary Shortest Path Problems with Resource Constraints, are considered. For speeding up the pricing phase, construction heuristics, large neighborhood search and truncated label setting algorithms are also used. Valid inequalities are inserted in order to have a tighter relaxation of the original problem. Also, the authors prove how some previously used cuts for this type of VRP are implicit in the Set Partitioning formulation of the problem.
In [9] (then extended in [8]) the authors illustrate a unified method to tackle multiple MAVRP types. An exact framework is proposed based on a set partitioning formulation and three heuristic methods that find bounds to the dual solution of the LP relaxation. The bounds are used by a column-and-cut generation algorithm that finds the optimal integer solution adding valid inequalities to the model. The authors illustrate how the three bounding routines can be tailored to handle different constraints from other specific VRP variants.
In [14], a Pickup and Delivery VRP is characterized by the presence of multiple depots and a heterogeneous fleet. Moreover, customers place preferences on time of visit, so that their violation implies a cost. A branch-and-price algorithm has been devised to exactly solve this problem. A set covering formulation of the problem is used for the Master Problem. The Pricing Problem exploits exact bi-directional dynamic programming techniques and

heuristic routines. The implemented branching rules involve the number of vehicles and the arcs in the paths.

4.5 Basic Exact Framework for VRP

The branch-and-price framework described in the following is a very popular baseline for the optimal solution of many variants of the VRP ([14, 20, 86] among others). We will rely on the main scheme of this algorithm when devising a Column-Generation algorithm for EPDT, as explained in Chapter 6.

One of the main exact approaches for the solution of Vehicle Routing Problems and their extensions relies on the Set Partitioning formulation (4.13 - 4.16) reported in Section 4.1. The formulation considers an exponential number of variables, which naturally leads to the use of a CG algorithm. As explained in Chapter 2, we recall that the CG is based on decomposing the problem into a Master Problem (MP) and one or more Slave (or Pricing) Problems (SPs). MP is solved on a restricted set of variables, called Reduced Master Problem (RMP), while SPs generate new variables to add to the Reduced MP. In the VRP context, every variable of the MP corresponds to a feasible route of the VRP instance. Hence, starting from a set of feasible routes as the Reduced MP, one can generate new variables of negative reduced cost in order to try to improve the solution (see [33] among others).

Definition 4.1. *Given a complete graph $G = (N, A)$ with cost coefficients c_{ij} on each arc $(i, j) \in A$, we call* distance matrix *the matrix $D = (c_{ij})_{i,j \in N}$.*

Definition 4.2. *A distance matrix of a graph $G = (N, A)$ is* Euclidean *if for every $i, j, k \in N$ it holds:*

$$c_{ij} \leq c_{ik} + c_{kj}.$$

Let us define the *Set-Covering* formulation corresponding to the Set-Partitioning model in (4.13) - (4.16):

$$\min \sum_{r \in \Omega} c_r x_r \tag{4.17}$$

$$s.t. \sum_{r \in \Omega} a_{ir} x_r \geq 1 \qquad \forall i \in N \setminus \{0\} \tag{4.18}$$

$$\sum_{r \in \Omega} x_r \leq |K| \tag{4.19}$$

$$x_r \in \{0, 1\} \qquad \forall r \in \Omega, \tag{4.20}$$

where the only difference is constraint (4.18) that requires each node to be contained in at least one route. We recall that the aim is to cover the customer nodes, so the depot node 0 is excluded from constraints (4.18).

Remark 4.1. *Under the assumption of a Euclidean distance matrix, the optimal solution of the Set-Covering problem is also optimal for the Set-Partitioning model, since we can remove any multiple node visit from the routes till obtaining a partition which is still optimal.*

Based on Remark 4.1 and assuming a Euclidean distance matrix, it is equivalent to consider the Set-Partitioning or Set-Covering model, although the latter formulation owns a higher numerical stability (more details in [33]).
Now we consider the linear relaxation of the Set-Covering formulation:

$$\min \sum_{r \in \Omega} c_r x_r \tag{4.21}$$

$$s.t. \sum_{r \in \Omega} a_{ir} x_r \geq 1 \qquad \forall i \in N \setminus \{0\} \tag{4.22}$$

$$\sum_{r \in \Omega} x_r \leq |K| \tag{4.23}$$

$$x_r \geq 0 \qquad \forall r \in \Omega. \tag{4.24}$$

Notice that we do not need to consider the constraints $x_r \leq 1 \ \forall \ r \in \Omega$, since values higher than 1 would only deteriorate the objective by Remark 4.1. The Reduced Master Problem must be initialized by a feasible subset of routes. One way to do it is introducing a set of dummy routes $d_m, m = 1, \ldots, |N|$ with $c_{d_m} = \infty$ and $a_{id_m} = 1$ for $i = m$, 0 otherwise. The solution of the Reduced MP with an LP solver provides us with the primal solution of the problem and the dual solution $(\pi_0, \pi_1, \ldots, \pi_{|N|})$, where π_i $(i \in N \setminus \{0\})$ are the dual variables related to the covering constraints (4.22) and π_0 the dual variables corresponding to the maximum number of vehicles allowed by (4.23).

4.5.1 The Pricing Problem

With reference to [33], we describe how the Pricing Problem can be formulated as an Elementary Shortest Path Problem (ESPP), defined as the ESPPRC (see Section 2.3.2) without any resource constraints.
At each node of the tree structure in the Branch-and-Price procedure, the continuous relaxation of the Reduced MP is considered. As explained in Section 2.1.3, we know that the reduced cost $\bar{c}_r$ of a route r can be obtained by the following formula:

$$\bar{c}_r = c_r - (u^*)^T A_r,$$

where c_r is the cost of route r in the objective function, A_r is the column related to the route r in the matrix of constraints and u^* is the dual solution

corresponding to the incumbent primal solution of the RMP.
Hence, given a route r and the dual solution π it holds:

$$\bar{c}_r = c_r - \sum_{i \in N \setminus \{0\}} \pi_i a_{ir} - \pi_0.$$

Notice that, given a route $r = (n_0, n_1, \ldots, n_l, n_{l+1})$ with $n_0 = 0$, and the distance matrix $D = (c_{ij})_{i,j \in N}$, the cost of the route is computed by $c_r = \sum_{m=0}^{l} c_{n_m, n_{m+1}}$. Moreover the corresponding column A_r has 1 on each entry related to a node contained by r, 0 otherwise. Therefore, we have that

$$\bar{c}_r = \sum_{m=0}^{l} c_{n_m, n_{m+1}} - \sum_{m=1}^{l} \pi_{n_m} - \pi_0,$$

hence we obtain

$$\bar{c}_r = \sum_{m=0}^{l} (c_{n_m, n_{m+1}} - \pi_{n_m}).$$

In light of what observed above, for each arc (i, j) we can define the *reduced cost coefficient* $\bar{c}_{ij} := c_{ij} - \pi_j$, so that the reduced cost of a route is computed as

$$\bar{c}_r = \sum_{(i,j) \in r} \bar{c}_{ij}.$$

Let us observe that finding variables with negative reduced cost is equivalent to search for a negative cost path with respect to the reduced cost coefficients $\bar{c}_{ij}$. Therefore, a way to find such paths or ensure that no such paths exist is to solve a Shortest Path Problem. In particular, since the model we stated does not allow cycles in the routes (coefficients a_{ir} do not assume values greater than 1), the paths must be elementary (ESPP).
Notice that in the presence of constraints defined on some route attributes, this problem become the ESPPRC defined in Section 2.3.2 and can be solved through the label correcting algorithm described in the same section.

4.5.2 Branching strategies

A straightforward branching rule is to take any variable x_r with fractional value f in the current solution, and decompose the problem into two problems containing respectively the additional constraints $x_r \leq \lfloor f \rfloor$ and $x_r \geq \lceil f \rceil$. In the current formulation, this branching technique corresponds to fix some variable to be 0 and 1 respectively. In other words we force r to be or not to be part of the solution. This branching strategy has the drawback of decomposing the problem into two unbalanced sub-problems. Moreover, it has not a direct representation n SP.
As suggested in Section 2.2.3, an effective branching scheme is obtained by

decomposing the problem through the arc variables of the Arc-Flow formulation of the problem, since they are included in the Pricing Problem. The arc variable x_{ij} can be fixed to value 0 or 1. In the former case, the corresponding arc (i, j) is removed from the graph $\bar{G}$ and then the ESPPRC is solved. In the latter case we solve two ESPPRC, one from s to i and another from j to d.

4.6 Classical Heuristics for VRPs

In this section we explore the basic heuristic and meta-heuristic procedures presented in the scientific literature.

Solution representation

One of the fundamental characteristic in heuristic algorithms is the *solution representation*. The most direct way is to encode a solution as a set of routes where a route is encoded as a list of nodes. Another popular encoding is the giant tour representation, where a string containing all customers nodes is considered and, then, decoding procedures are responsible for splitting the tour into different routes [78]. In [3], a solution is encoded by the assignments to time periods and a decoding procedure builds a route by means of a quick constructive heuristic.

Constructive Heuristics

We give a brief description the most popular constructive heuristics designed for the VRP (further details in [104]):

- the *Clarke and Wright* algorithm. It starts from an initial solution that contains a set of routes each one serving a single customer, then pair of routes are iteratively merged in such a way that two routes nodes i and j where the merging takes place maximize $c_{i0} + c_{0j} - c_{ij}$, called *saving*;
- the *Sweep* algorithm. Starting from a empty solution, it spans circularly the customers space and adds to the same route every node encountered as long as the route is feasible. If not, a new route is initialized and the insertion is performed on it;
- the *Route-first and Cluster-second* algorithm creates a giant tour that visits every node of the graph, then the tour is split into different feasible routes;
- the *Cluster-first and Route-second* algorithm firstly partition the customers into clusters, and then it solves a Traveling Salesman Problem in each one of them.

Improving Heuristics

The main improvement heuristics for the VRP are based on Local Search where different types of neighborhoods are explored. Neighborhoods may have an impact on each route independently, or on more than one route simultaneously [111]:

- *k-opt*: it splits the route into sub-routes by removing k arcs and it inserts k arcs to reconnect the sub-routes in different ways. Most used neighborhoods are 2-opt and 3-opt;
- *Or-exchange*: given a route, it relocates sub-sequences of consecutive nodes of some predefined size at a different position within the same route;
- *relocation*: it moves one node from a route to a different one;
- *swap*: it exchanges two nodes between two routes;
- *2-Opt** or *crossover*: it removes and inserts arcs similarly to the 2-Opt algorithm, but the reconnection affects sub-routes from two different routes;
- *CROSS-exchange*: it swaps sub-sequences of consecutive nodes between two different routes.

Neighborhoods containing a very large number of solutions are also part of successful algorithms:

- *ejection chain*: find a cycle that alternates existing and non-existing arcs and replace the existing with the non-existing ones;
- *multiple relocation*: applies the relocation heuristic to b customers within m routes.

4.7 Heuristic Approaches for MAVRPs

Because of the computational complexity of the MAVRP, together with the scalability required by more and more attributes of the real-world applications, heuristics and meta-heuristics algorithms have been proposed that can attain near-optimal solutions for large instances of VRPs with difficult constraints (see for instance [58, 63, 111]). Meta-heuristics can reach a high quality solution in an acceptable amount of time, while being less sensitive to the introduction of new attributes in the model with respect to an exact approach, both for mathematical formulation issues and for the required computational effort. Below, we explore the main heuristic approaches for the solution of MAVRPs in literature.

Search-based, population-based and hybrid algorithms have shown satisfying performance (a bibliography can be found in [18]), as well as matheuristic techniques [6]. Detailed surveys on the most successful heuristic algorithm in literature to solve the VRP with different attributes can be found in [18, 63, 111].
In the following, we describe some successful heuristic algorithms proposed in the literature, providing more details as they represent a substantial reference for the techniques developed in this thesis. First, we will consider some works that propose general methods devised to be suitable for different MAVRPs in a unifying approach. We will review some more specific approach to MAVRPs with pickup and delivery, focusing on the ones that make use of Tabu Search and Variable Neighborhood Search, which represent the basis of the methods we propose for EPDT in Chapter 5.

4.7.1 General approaches to MAVRPs

In [23] a *Parallel Iterated Tabu Search* is implemented to solve VRP instances with different sets of attributes. The algorithm embeds features of the Tabu Search algorithm into a Iterated Local Search (ILS) framework and it is *configured depending on the type of MAVRP*. For instance, in the presence of time windows, insertion of a node n in a route r is performed through the least-cost insertion of n between two consecutive nodes, whereas the removal of n from r is performed by re-linking the predecessor and successor nodes of n. With no time windows, a low level heuristic is responsible for the rearrangement of a small segment of r containing n, during both insertion and removal phase. The construction of the initial solution is performed by a sweep heuristic.
The search-based method used in the improvement phase is a Tabu Search heuristic. In particular, *it allows to visit infeasible solutions* by relaxing some constraints (capacity, route duration, and time windows), penalizing them in the objective function through adaptive non-negative coefficients. In the Tabu Search scheme, a solution s is partially characterized by a set of triplets (i, k, l) stating that customer i is visited by the vehicle k in the day l (the value of l is determinant in case of Periodic VRPs). The *neighborhoods* considered are

1. replace a customer into a different route (change (i, k, l) with (i, k', l));
2. change the visit combination related to multiple days.

The tabu tenure is selected at the beginning of each improvement phase randomly on an interval depending on the instance size. Whenever a move is applied, any other move containing the set of triplets just involved is declared tabu. Moreover, an *aspiration criterion* is adopted so that tabu solutions is accepted whenever it is better than all other solutions for at least

one triplet. The Tabu Search undergoes a *diversification* phase in order to visit poorly explored regions of the solution space. In particular, when the search is in a local minimum, the objective function is penalized by a factor depending on a measure of the incumbent move frequency in past solutions. An *intra-route refinement* phase is executed at regular iterations intervals, by removing and reinserting each customer as described above. The perturbation of the ILS is characterized by selecting a random customer and detecting a cluster of close nodes that are removed and reinserted in a random order with the aim of reducing the insertion cost. Two *stopping criteria* act at two different levels of the algorithm:

- the Tabu Search is stopped when the number of non-improving iterations reaches the value M, where M is more and more lowered as the iterations of the ILS progress, in order to spend less time when it is repeatedly called on similar solutions;
- the ILS is terminated when a maximum number of total iterations is reached.

Finally, *parallelization* is used to reach higher efficiency. The algorithm is executed in a multi-start fashion, and each parallel thread selects the algorithm parameters according to a probability distribution, to diversify the searches. Moreover, synchronization is performed to share knowledge on the solutions among the different threads, then a cross-over phase is triggered to mix solutions features from different parallel searches.

A unified heuristic algorithm based on Large Neighborhood Search that can handle different versions of the VRP is described in [77]. The VRP considered attributes are capacity, time windows, pickup and delivery, multi-depot, vehicle compatibility and open routes. In the first stage of the solution method, the VRP instance is transformed into a *Rich Pickup and Delivery Problem with Time Windows (RPDPTW)* that can be solved by an *Adaptive Large Neighborhood Search* (ALNS). The destroy and repair phases of the ALNS scheme can be embedded into any search-based algorithm (e.g. Simulated Annealing, Tabu Search, etc.). At the beginning of a new iteration, a *roulette wheel selection* is triggered to choose one destroy and one repair neighborhood. We recall that a *roulette wheel selection* consists in choosing one solution from a pool where a discrete probability distribution is defined and the probability of each single solution is proportional to its objective value.
The probability corresponding to a neighborhood N_j is given by

$$p(N_j) = \frac{\pi_j}{\sum_{i=1}^{\omega} \pi_i},$$

where ω is the number of neighborhoods and π_i is the score of neighborhood N_i, that is a measure of past performance of N_i. In order for each single

destroy and repair heuristic to avoid stalling, noise is added to the objective function in the evaluation phase. The authors specify that destroy and repair neighborhoods are defined as request removal and re-insertion. Below, we describe some of the *destroy neighborhoods* used to remove q requests:

- *random removal*: remove q requests randomly;
- *worst removal*: remove for q times the requests with highest cost;
- *related removal*: remove a set of requests related by some measure (for instance the distance between the operations).

The used *repair neighborhoods* are:

- *basic greedy heuristic*: repeatedly make the cheapest assignment of a request to a route;
- *regret heuristic*: repeatedly make the assignment of a request to a route so that the difference between the best and second best assignment of such request is maximized (it is a sort of look-ahead policy to prevent difficult requests to be stuck at late stages of the constructive routine).

In the end, a two-stages approach is adopted with the aim of *minimizing the number of vehicles* used. Given a solution s with m vehicles, routes are removed from s and requests are added to a requests bank at a high cost. Then, the ALNS is run starting from s and, if a new feasible solution (i.e. with empty requests bank) is not found, the procedure steps back to the last found feasible solution as starting point for the execution of an ALNS.

In [112, 113] a unified method for the solution of MAVRPs is described. The *Unified Hybrid Genetic Search* presented in these works hybridizes genetic algorithms and local search, and proposes a unified framework for solving a wide range of large-scale vehicle routing problems with several types of attributes.
The authors remark how their method consists of a *component-based framework*, rather than the classic approach that rely on modeling a very large set of attributes, obtaining a more flexible and efficient design. The main mechanisms are abstracted and, depending on the type of MAVRP, they are specialized with no impact on the main routines. For this purpose, *attributes are grouped* by their impact on different aspects of the solution method:

1. the *ASSIGN* attributes affecting the assignments to general resources (e.g. depots, days, vehicle types, etc.);
2. the *SEQ* attributes change the nature of the graph and of the related node sequences;
3. the *EVAL* attributes are related to the solution evaluation.

This separation is exploited by the implementation of *independent components*, one for each group of attributes, without affecting the overall procedure. For instance, different EVAL components can be embedded in the method with no impact on the ASSIGN and SEQ modules.
In particular, as the local-search moves are represented by the recombination of sequence segments, the EVAL module requires functions to *extend resources* (e.g. load on board, cumulative distance and duration, etc.) and *evaluate the concatenation* of two or more segments.
The algorithm starts with an initial population of solutions with the giant-tour representation, then a *selection* phase couples pairs of feasible and infeasible individuals to obtain an offspring by a *crossover* operator. Then, a *split* procedure is used to convert the giant tour into a set of routes through the depot and an *education* step is executed by a local-search. Infeasible solutions are *repaired* with given probability. Finally, solutions are converted back to the giant-tour representation and re-inserted into the population. The procedure terminates after a maximum number of iterations or when a time limit is reached.
The evaluation of the solutions contain *penalties for infeasibility* and *diversification* components. Moreover, for sake of *intensification*, a decomposition is triggered after a predefined number of iterations so that a set of *elite* solutions is selected and the ASSIGN components is kept fixed. The ASSIGN, SEQ and EVAL modules are called at different stages of the overall procedure. For instance, the crossover operator needs the ASSIGN module to decide which genetic material is transmitted, or the EVAL component is called by the education phase.

4.7.2 Specific approaches for pickup and delivery problems

The scientific literature offers a remarkable variety of that solve Pickup and Delivery problems, also in dynamic settings, which is relevant for EPDT. Among others, in [92] a MAVRP from real-world application is tackled by an optimization algorithm that mixes features from the Tabu Search algorithm and the Variable Neighborhood Descent method. The MAVRP under study is characterized by time windows, capacity of vehicles, heterogeneous fleet, compatibility between orders and vehicles, maximum number of orders in a vehicle (for work load balance) and pickup and delivery orders. In particular, orders may appear as pickup-only orders, delivery-only orders or pickup-and-delivery orders, always with a single pickup and/or delivery task. The objective of the problem is the minimization of total distance and inactivity times, coming from early arrivals with respect to time windows. The solution method includes a route construction method that builds routes according to one among multiple dispatching rules and rely on randomized functionalities for sake of diversification. An improvement phase follows, based on neighborhood search. It alternates two phases: a Variable Neigh-

borhood Descent and, once a local optimum is reached, a Tabu Search, as a shaking phase that generates a new solution. Both phases involves classical intra-route and inter-route neighborhoods as the ones described in Section 4.6.
In [42], a Tabu Search algorithm is devised in order to solve a pickup and delivery problem with soft time windows in static and dynamic settings. The problem rises from courier service for same-day pickup-and-delivery real time requests. The objective function is a weighted combination of three criteria: total travel time, sum of lateness over all locations and sum of overtime over all vehicles. The main neighborhood structure involves ejection chains and it is part of a Tabu Search algorithm. In particular the Tabu Search makes use of adaptive memory and decomposition techniques to reduce the computational time, as well as a master-slave design that includes parallel explorations. Since the dynamic setting requires a solution to be found quickly when answering to new orders, a specific procedure for dynamic insertions is devised, based on a local descent applied to the best solution in memory. The descent then is stopped at the first local minimum using the ejection chain neighborhood.

4.8 Approaches to Dynamic and Stochastic VRPs

The *input data* of VRPs can have a different nature: they can be fully known before the execution of the solution method (static settings) or, as we will see in the following, the can come into play during the execution od operations (dynamic settings) or can be affected by uncertainty (stochastic settings).

4.8.1 Dynamic setting

A vehicle routing problem is said to be *static* whenever all the problem input data are known beforehand. In many applications, for instance express courier deliveries, the static setting is almost never the case. Data about order request often happen to be available during the execution of the planning, therefore re-optimization techniques can improve the incumbent solution subject to the new input data. This last setting, in contrast with the static case, is called *dynamic*. Dynamic routing problems can be subject to more or less frequent input updates with respect to the planning horizon duration, as well as more or less restrictive deadlines to complete new requests' operations.
A measure called *degree of dynamism (dod)* has been defined by [65] to represent this concept

$$dod = \frac{1}{|R|} \sum_{i \in R} \frac{T - (l_i - t_i)}{T}, \tag{4.25}$$

where R is the set of all the requests in the VRP instance, the planning horizon is represented by the interval $[0, T]$, l_i and t_i are respectively the latest time a request $i \in R$ can be accomplished and the time the request i is known.

Solution strategies for dynamic VRPs

A basic technique to *re-optimize* a dynamic VRP instance when new input information becomes available is to simply solve the static problem including the new input data. This strategy implies to modify some parameters in order to make the solution consistent with past decisions. Re-optimizing with the full procedure may be not efficient enough, so another option is to devise two routines: an algorithm specifically creates the solution of the routing problem in static setting to run at the beginning of the planning horizon, then another algorithmic technique based on faster insertion heuristic is triggered any time new requests are revealed [13], giving rise to the so called reaction policy.
A different approach to tackle dynamic VRPs alters the *objective function* to meet more suitable criteria for the dynamic setting. For example, focusing the optimization only on the distance traveled ignoring the long-term components (since the routes are likely to be modified) [13], or also emphasizing the short-term minimization of distance and maximizing the long-term slack-time, so that future demand can be more easily satisfied [69].

4.8.2 Stochastic setting

Also uncertainty can affect input data. In this case, some parameters are characterized by (known or unknown) probability distributions that can be exploited for better decision making. This is called the *stochastic* VRP. Following the discussion in [84], stochasticity is mainly related to:

- *service time*: the time that an operation of pickup or delivery takes to be accomplished can vary. For instance, a truck may wait at a delivery point in line with other trucks;

- *travel time*: the time between two points can increase or decrease depending on the time of the day it is traveled, car accidents, etc.;

- *demand*: the exact amount or size of items that must be loaded during an operation is subject to variation;

- *customers*: this is a stochastic demand scenario where also the locations where operations must be fulfilled are not deterministic.

Solution strategies for stochastic VRPs

One class of algorithmic techniques used to solve stochastic versions of VRPs is based on the *sample scenario approach*, where multiple future outcomes are generated and inserted into the optimization procedure (e.g. [43]). This way, the evaluation of the solutions includes the impact of (potential) future events on the objective values. Therefore, the sample scenario approach relies on two main modules: (a) an optimization algorithm to solve the routing problem in static settings and (b) a sample generators of the future requests. The latter component requires a probability distribution to be known, but this is not always the case. Thus, if historical data are available, inferential statistics can provide estimated distributions for the sampling phase thanks to data-driven techniques (for example, see [12]).
Sampling and evaluating against several scenarios are time-consuming operations to perform at run-time, so *non-sample scenario techniques* have been investigated by researchers. For instance, approximation of the future utilization of the fleet together with future requests information in space and time is part of a solution method described in [59]. Also, in [40] a spatial and temporal clustering approach of the future demand is used to guide vehicles in regions with higher probability of requests.

4.8.3 Dynamic and Stochastic VRP

The Dynamic and Stochastic VRP (DSVRP) combines the dynamic setting of the input with the stochastic characteristics of the problem [13, 79].
Main literature approaches can be classified into *reactive* policies, that take into account only known information on past pending requests and the new one as in the pure dynamic case, and *anticipatory* algorithms, exploiting stochastic information on past requests to better accommodate the future ones [43]. Many solving methods devised for DSVRP are based on a *rolling horizon* setting: whenever an optimization is run, only a predefined near-future sub-interval of the planning horizon is taken into account, reducing complexity and uncertainty effects [116].
Moreover, the DSVRP introduces new degrees of freedom in the model: a *waiting time* schedule can be decided for each route, so that a vehicle that has completed an operation can either leave as soon as it has finished, or wait some amount of time (that is a variable of the model), with the aim of spending more time in more strategic points (see e.g. [13] or Section 7.3).
In a similar way, a *buffering* system can be used so that a known request is kept free to be re-assigned to another vehicle for a certain amount of time [80].
The *relocation* concept consists of the possibility for idle vehicles to be reallocated at more suitable positions to satisfy future requests. For instance, they may be median locations calculated from the historical data on orders

[43].
The usual assumption in the mentioned strategies is that whenever a vehicle is on its trip to some customer, it cannot be deviated toward some other place during that trip. Nevertheless, using the *diversion*, that is allowing the deviation of a vehicle from the current destination, can provide very effective solutions for DSVRPs [13].

Chapter 5

Design of a heuristic algorithm

In this chapter we consider the EPDT problem described in Chapter 3, and we devise an algorithm able to efficiently provide a high quality solution in an acceptable amount of time. The design of an algorithm to solve EPDT has to fit both the problem statement and the operational context.
The problem statement includes several attributes and asks for a flexible approach able to handle them simultaneously. State-of-the-art methods, despite their capability to adapt to different multi-attribute scenarios, can hardly fit the features of EPDT and take advantage of its specific definition towards efficient and effective solution methods. For instance, to the best of our knowledge, sequenced multi-pickup and multi-delivery orders, or the partition of the operations into tasks related to different days, have not been studied in literature, and led us to devising specific neighborhoods that exploit the order structure. Furthermore, EPDT defines long- and medium-haul trips which, together with a relatively short planning horizon, entail feasible solution with a large dispatch of vehicles and routes, and a reduced number of operations. This context suggests a tailored design of intensification mechanisms in the route sequencing phases.
Concerning the operational context, we recall that EPDT should be solved in both static and dynamic settings. For the dynamic setting, we observe that the frequency of events is relatively small compared to travel times or task duration. This configures relatively-low dynamism scenarios, suitable for reactive policies based on re-optimization (see Section 4.8.3), which require an efficient method to solve EPDT under up-to-date information.
In this Chapter, we give the details of the algorithmic approach we propose for EPDT, that hybridizes a Tabu Search algorithm with a Variable Neighborhood Descent (as defined in Section 2.4.2). We will start by providing a general description of the proposed solution method, illustrating the components of the score function, the solution evaluation procedure and

the neighborhoods structure.
The approach presented in this chapter represents one of the main contribution of the thesis, since it leads to an algorithm that is able to fit all the EPDT requirements in terms of both statement and operational context. More specific contributions are:

1. a solution evaluation procedure that suitably handles tasks in a route complying with the multi-pickup and multi-delivery precedence constraint (H2) (defined in Section 3.4);
2. a granular search that filters out unpromising moves based on a graph constructed on multi-pickup and multi-delivery orders;
3. a fast route evaluation procedure designed to reduce the computational effort on tomorrow tasks;
4. different parallel implementations of neighborhoods exploration and evaluation.

Notice that 1-3 take advantage of the specific EPDT features. Moreover, 2-4 represent speed-up techniques applied to the basic algorithm, assessed by results in Chapter 9. Part of the content of this chapter is the object of the works published in [28, 29, 30].

5.1 A two-level heuristic

Due to the inner complexity of the problem and the operational scenarios it supports, a meta-heuristic approach is suitable: we devise a Tabu Search algorithm hybridized with a Variable Neighborhood Search.
The design of the algorithm complies with several core characteristics described in Section 2.4.4 and [111]. In particular, we suitably combine multiple neighborhoods, granular exploration, parallel sub-routines, diversification phases.
Solving EPDT involves, at least conceptually, three decision degrees:

(i) the assignment of orders to vehicles;

(ii) for each vehicle, sequencing the tasks of the assigned orders;

(iii) determining the best vehicle route through task locations.

The proposed approach relies on the decomposition of the problem into two hierarchical levels: the first deals with (i) and determines order-to-vehicle assignments, which provide the input for the second level, dealing with (ii) and (iii), that is, the optimization of vehicle routes. The first level implements a Tabu Variable Neighborhood Descent heuristic (general details on Tabu Search and Variable Neighborhood Descent can be found in Section

2.4). The proposed approach, that we will detail in the following, can be summarized as follows. It starts from an initial solution (current solution) provided by a heuristic algorithm which iteratively determine the order-to-route assignment with the least marginal cost. After the constructive phase, the exploration of different neighborhoods is performed according to the VND policy. The solutions are evaluated by a score function and the best neighbor is chosen as the new current solution. The incumbent solution is updated each time a better current solution is generated. In order to avoid cycling, a tabu list stores information on the last visited solutions, so that they are excluded from the eligible neighbors. The process iterates until at least one out of two termination conditions occurs, namely reaching (a) a maximum number of iterations or (b) a maximum number of iterations without an improvement.
In the decomposition schema, each neighbor solution represents a first-level decision and determines, for each vehicle, the set of orders assigned to it. In order to evaluate this solution, the second-level decisions are made, since the score function depends on the route run by each vehicle to service assigned orders. Towards vehicle routes optimization, a second-level heuristic based on local search is devised. In other words, the first-level tabu search explores the space of all the possible assignments of orders to vehicles, whereas, at the second level, the search space covers task sequences and related vehicle routes.
The schema sketched above is very general and we need to specify its components, in particular: the score function used for first-level solutions, the second-level local search that evaluates the score function, procedures to compute an initial order assignment, first-level neighborhoods and tabu list. In the following, we will first describe each component, then, we will outline the overall approach proposed to solve EPDT.

5.2 The score function

The objective of EPDT is maximizing the profit from daily orders. Given a solution s, let $O(s)$ be the set of orders assigned to vehicles, and $R(s)$ the set of vehicle routes. Each route $r \in R(s)$, run by a vehicle $v(r)$, defines the sequence of tasks and, as a consequence, their starting time, computed from travel times between positions and from task duration. With reference to the notation defined in Table 3.1, the net profit $G(s)$ is defined as

$$G(s) = \sum_{o \in O(s)} P_o - \sum_{r \in R(s)} (C(r) + A(r) + W(r)) - \sum_{o \in O^U \setminus O(s)} L_o \qquad (5.1)$$

where we recall that

- P_o is the profit of order $o \in O$;

- $C(r) = C_v^F + C_v^U \cdot l(r)$ is the cost of route r;
- $A(r)$ is the prospective cost of route r;
- $W(r)$ is the cost for soft time windows violation of route r;
- L_o is the cost of missed urgent orders $o \in O^U \subseteq O$.

Given a route, the model that we adopted for the prospective cost $A(r)$ is based on the assumption that, similarly to the regular route cost $C(r)$, it behaves proportionally to the length $l^{next}(r)$ of the shortest (according to distances d_{ij}) route starting from the last-served today task and visiting all next-day tasks assigned to $v(r)$. Since incomplete orders information, especially during the early stages of the planning phase, may affect the evaluation of the prospective cost, we modulate its value through a coefficient κ_A. In this way, an estimate of such cost is taken into account, but incomplete information has an impact that can be reduced through κ_A. Therefore, we model the prospective cost $A(r)$ as

$$A(r) = \kappa_A \, C_v^U \, l^{next}(r).$$

Given a route $r = (t_0, t_1, \ldots, t_L)$, we represent the cost function $W(r)$ for soft time windows violations as

$$W(r) = \sum_{i=1}^{L} L_i^w \max(0, \tilde{t}(i) - t_i^e),$$

where $\tilde{t}(i)$ is the arrival time at node i, t_i^e is the end of the soft time window of node i and L_i^w is the soft time window violation cost at task i (0 if no soft time window is associated with i).
We recall that, in order for s to be feasible for EPDT, each route in $R(s)$ must satisfy several constraints. In particular, with reference to the encoding in Section 3.4, we need $O^M \subseteq O(s)$, that is, all mandatory orders are assigned. For the sake of an effective exploration of the order assignment space (as stated in Section 2.4.4), we allow visiting solutions with missing mandatory orders, which is penalized in the score function through a component

$$B(s) = \sum_{o \in O^M \setminus O(s)} P_o.$$

In addition, EPDT defines several preferences, that we model as soft constraints by including them in the score function [28]. In particular, we define the following performance indicators associated with the whole solution s or to each route $r \in R(s)$:

- $I_D(s)$: this indicator is related to soft constraint (S2). Let $\mathcal{Q}$ be the set of subsets of orders to be preferably assigned to a same vehicle,

and, for each $Q \in \mathcal{Q}$, let $N_Q(s)$ be the number of extra vehicles used by solution s to serve the order in Q. N_Q is obtained by counting the number of vehicles that are assigned with at least one order of Q, minus one. Then $I_D(s) = \sum_{Q \in \mathcal{Q}} N_Q(s)$;

- $I_E(s)$: this indicator is related to soft constraint (S3). It counts number of missing vehicles at the depot with respect to the minimum required;
- $I_F(r)$: this indicator is related to soft constraint (S4). It represents the number of orders assigned to the preferred vehicle;
- $I_H(r)$: this indicator is related to soft constraint (S5). It is a boolean indicator taking value 1 if r has, in solution s, its preferred end-of-day position, 0 otherwise;
- $I_J(r)$: this indicator is related to soft constraint (S1). It measures the extra-time with respect to the preferred maximum duration of r;

Notice that the indicators above make explicit the penalty terms introduced conceptually in Section 3.4.
Summarizing, a first-level solution s is evaluated by the following score function:

$$\begin{aligned} Z_1(s) &= G(s) - w_D I_D(s) - w_E I_E(s) - M\, B(s) + \\ &+ \sum_{r \in R(s)} \big(w_F I_F(r) + w_H I_H(r) - w_J I_J(r) \big) \end{aligned} \tag{5.2}$$

where w_X, for each indicator I_X are parameters that represent the weight of each soft constraint in the evaluation of a solution. The value of such parameter depends on the context where EPDT is applied, so they need to be calibrated. The constant M takes a big value to suitably penalize unfeasible solutions, and w_X takes smaller non negative values to foster preference matching.
The parameter adjustment is a hard and critical task that should be carefully executed in order for the objective components to match the decision maker's criteria. One way to accomplish it is a parameter tuning on the field, where operations managers provide feedback on the algorithm behavior obtained by changing parameter values, till reaching a set of parameters that correspond to satisfactory solutions according to the assessment of the operations managers themselves.

5.3 Solution evaluation and second-level heuristic

Given the order-to-vehicle assignments, most of the indicators taking part in the score function (5.2) depend on vehicle route, which is in turn defined

by the sequence of tasks. This asks for solving, for each vehicle, the *Intra-route optimization* problem, defined as follows: given the orders assigned to a vehicle, determine a feasible route r that minimizes the following function:

$$\begin{aligned} Z_2(r) &= C(r) + A(r) + W(r) + \\ &\quad -w_F I_F(r) - w_H I_H(r) + w_J I_J(r) + w_E \frac{\bar{N}_0}{|V|} \bar{I}_E(r) \end{aligned} \tag{5.3}$$

where, looking at the last addend, $\bar{I}_E(r)$ is a Boolean indicator taking value 1 if r does not end at the depot, 0 otherwise; $\bar{N}_0 = \max\{0, N_0 - \tilde{N}_0\}$, N_0 being the preferred number of vehicles at the depot, and $\tilde{N}_0$ being the number of vehicles with depot as end-of-day position. Notice that all the components of (5.3) directly come from the score function (5.2) but the last one, which is related to $I_E(s)$: it is intended to penalize solutions with a small number of vehicles ending at the depot, taking into account the weight of $I_E(s)$ in $Z_1(s)$ (see attribute S3). Moreover, the factor $\frac{\bar{N}_0}{|V|}$ increases the relevance of this component of $Z_2(r)$ when the number of vehicles is relatively small with respect to $\bar{N}_0$ (the additional number of vehicles required at the depot besides the one guaranteed by routes' feasibility).
Given a route r, $Z_2(r)$ is evaluated by computing the starting time for each task according to travel times t_{ij}, task duration, earliest starting time from the time windows, and inserting short and long compulsory breaks when necessary.

Intra-route optimization is a hard problem and, moreover, it has to be performed for each assignment generated by the first-level decisions. We thus propose a second-level heuristic based on local search: it starts from an initial route and generates neighbor solutions, evaluated by (5.3); the best neighbor is chosen as the new current solution, and the process is iterated until no improving neighbor exists. We now describe its components, after stating the following assumptions.

Assumption 5.1. *The second-level local search is triggered by:*

- *the assignment of one new order o to a vehicle route r;*
- *the removal of one or more orders from r, with, in case, the simultaneous assignment of a new order;*
- *the assignment to r of a set of orders coming from another route.*

Assumption 5.2. *The route affected by the changes that trigger the second level local search (as fro Assumption 5.1) is feasible, that is, it satisfies all EPDT hard constraints (H1-H7), but, in case, H4 on mandatory orders (which in fact depends on the overall assignment).*

In the following, a route r is defined a sequence $t_0, t_1, \ldots, t_n, t_{n+1}$, where t_i is the i-th task, and t_0 and t_{n+1} are defined as initial and final dummy tasks.

5.3.1 Initial task sequence

The route used to initialize the local search, depends on the triggering modification. In case a set of orders is removed from r, the initial route is the same r without the tasks corresponding to removed orders: this always yields a feasible route, since r is feasible by Assumption 5.2.
If a new order o is assigned to r, we obtain an initial route including o as follows. We recall that o may include one or more pickup and delivery tasks. For each i and j such that $0 \leq i \leq n$, $0 \leq j \leq n$ and $i \leq j$, insert all pickup tasks of o after t_i and all delivery tasks after t_j and evaluate (5.3), keeping the sequence providing the best value as the initial route. In order to speed up the convergence of the following local search, tasks are inserted after i or j according to a nearest neighbor criterion. Notice that all insertions yielding an infeasible route are discarded so that the procedure ends up with either a feasible route including the new order o, or no feasible route. The latter case prevents applying further local search.
If more than one new order is assigned to r, coming from a different route $\bar{r}$, a procedure similar tho the previous one provides the initial solution. In this case, the pickup (resp. delivery) tasks of all orders are inserted after t_i (resp. t_j), following the sequence they had in $\bar{r}$.

5.3.2 Second-level neighborhoods

We define neighborhoods by the following moves:

- *task insertion*: for each i, j such that $i \neq j$, insert t_i after t_j;
- *task swap*: for each i and j such that $i < j$, swap t_i and t_j.

The moves applied to generate neighbor solutions are different depending on o being fixed or not: we say that o is a *fixed order* if the vehicle it is assigned to cannot be changed. This is the case of orders including *yesterday* tasks: their assignment comes from yesterday planning leaving some pending *today* tasks already on board. Other fixed tasks may derive from special operations manager requirements. If o is fixed, we obtain neighbor solutions from both task insertions and task swaps, otherwise, only task swaps are applied. In order to preserve route feasibility, moves are discarded if the resulting route is not feasible.

5.3.3 Overall Outline of the second level heuristic

In Algorithm 8, we provide a high-level outline to summarize the second-level procedure, that we call $L2$. For sake of simplicity, we illustrate the case of a single order insertion o in a route r (the pseudo-code is similar for multiple orders insertion or for single order removal). An optimized route containing all the tasks in o is returned. With P (resp. D) we denote the set

of all pickup (resp. delivery) tasks in o. The procedure that determines the initial task sequence is encoded by Steps 3 to 14: it sequentially inserts tasks in P or D after a selected task t_i or t_j already in r according to a nearest neighbor heuristic, and it uses a set R to collect all feasible routes generated, if any, among which the one with better score Z_2 is chosen. Starting from this route, a local search procedure improves the sequence of all the tasks in r, using the second level neighborhoods described above and Z_2 as score function. If R is empty (see Step 11), the procedure stops and returns no feasible route.

Algorithm 8: Second-level heuristic

```
1  def L2(r, o):
2      P = getPickups(o), D = getDeliveries(o), R = ∅;
3      foreach (t_i, t_j) ∈ r × r : t_i ≤ t_j do
4          nearest_neighbor(t_i, P);
5          nearest_neighbor(t_j, D);
6          r̄ =
            concatenate([t_0, ..., t_i], P, [t_{i+1}, ..., t_j], D, [t_{j+1}, ..., t_{|r|}]);
7          if r̄ is feasible then
8              R = R ∪ r̄;
9          end
10     end
11     if R is empty then
12         return Infeasibility;
13     end
14     best = arg max{Z_2(r) : r ∈ R};
15     if o is fixed then
16         best = local_search(best, [task_insertion, task_swap], Z_2);
17     end
18     else
19         best = local_search(best, task_swap, Z_2);
20     end
21     return best;
22 end
```

5.4 Construction of the first-level Initial Solution

At the first level, the Tabu VND procedure starts from one initial solution of EPDT that can be provided by two alternative heuristic algorithms.

The first algorithm, called *Best Insertion (BI)*, is based on the greedy heuristic described in Section 4.7 and [77]. First, fixed orders are assigned to their vehicle, together with initial and final dummy tasks. Moreover, in order to deal with the end-of-day position, a further dummy task is added with suitable time window at the end of the planning day: its position is the one prescribed or preferred for the vehicle, if any, or the depot otherwise; in case of prescribed route ending position, the dummy task is part of a mandatory order, normal otherwise. Let $\bar{O}$ be the set of not yet assigned orders, initially equal to O. For each $o \in \bar{O}$ and each vehicle $v \in V$, we tentatively assign o to v and compute the score function (5.2) of the partial solution obtained. We denote with $s(o, v)$ the partial solution obtained from a solution s assigning order o to vehicle v.
Notice that each tentative assignment triggers the execution of the second level heuristic to evaluate (5.3) for the route r associated with v. If intra-route optimization does not provide any feasible route, then the assignment is discarded, thus guaranteeing that following tentative assignments always start from a feasible route, according to Assumption 5.2. A *dummy vehicle* is added to the solution in order to collect all the orders left unassigned at the end of the initial heuristic. Notice that, since the objective is to maximize the net profit of a solution, an order may be left unassigned not only for lack of feasible insertions but also because no assignment of such order leads to a score improvement.
BI, at each iteration, selects the $(\bar{o}, \bar{v})$ pair corresponding to the best partial solution, that is

$$(\bar{o}, \bar{v}) = \arg\max_{o \in O, v \in V} Z_1(s(o, v)).$$

Then, BI updates the current partial solution with $\bar{s} := s(\bar{o}, \bar{v})$. This procedure is repeated starting from $\bar{s}$ until all the orders are assigned to a route (including the dummy vehicle route).

The second algorithm is called *Round-Robin insertion with Priority* (RP) heuristic and defines three classes of vehicles:

i) vehicles with orders on board (they must be deployed since they have to complete the order);

ii) empty vehicles with starting position different from the depot;

iii) empty vehicles at the depot.

Notice that, by definition of the three classes, no vehicle can belong to more than one class. Vehicles within the same class are sorted by ascending

operational costs. Orders are assigned to vehicles in the first class, one order at a time for each vehicle, in a round-robin fashion. To this end, orders are sorted by a proximity criterion: given a route $r = (i_1, \ldots, i_L)$, the score associated with an order o positioned in j is

$$p_r(j) = U(o) + d_{0j} - \min_{i=1,\ldots,L} \{d_{ij}\},$$

where 0 is the depot and $U(o)$ is a parameter depending on the order priority. The rationale is to prefer the early (and hence hopefully better) insertion of priority orders, and orders far away from the depot. Once the capacities of the vehicles in the first class are saturated, remaining orders are assigned to the second class according to the same procedure. The assignment of remaining orders occurs sequentially (a vehicle is saturated before considering the following one) for the third class. In such a way, we try to accommodate as much priority orders as possible in different routes, while saving the deployment of some vehicles.

5.5 First-level Tabu Search neighborhoods

A Tabu Search procedure explores the space of possible order-to-vehicle assignments. With the aim of perturbing order-to-vehicle assignments, the first-level tabu search defines moves involving non-fixed orders currently in charge of different routes, including the ones in the dummy vehicle. We recall that the assignment of fixed orders and dummy tasks to the prescribed vehicle cannot be modified. We consider four neighborhoods.

Single Order Relocation

A first type of moves is the *single order relocation* (1R), where an order is moved from its current route to a new route. Denoting by k the number of non-fixed orders and by v the number of vehicles, the size is $O(k \cdot v)$, since the neighborhood contains a solution for each of the k non-fixed orders and each of the $v - 1$ alternative vehicles;

Two-Orders Swap

The second neighborhoods originates from *two-orders swap* (2S), consisting in swapping the assignment of two orders in two different routes. The neighborhood contains a solution for each pair of distinct orders assigned to distinct vehicles, therefore its size is $O(k^2)$;

Multiple Relocation

A third neighborhood is generated by *multiple relocation* moves (mR): they generalize 1R moves as they simultaneously displace up to m orders in a

same route r_1 to a different route r_2. We have a neighbor solution for each subset of at most m orders currently on board of the same vehicle and for each of the v alternative vehicle, hence the size of the mR neighborhood is $O(k^m \cdot v)$.

Two-Orders Chain Shift

The fourth neighborhood is the *two-orders chain shift* (2C), where an order o_1 assigned to route r_1 is moved to a route r_2, and an order o_2 assigned to r_2 is moved to a third route r_3. The size is $O(n^2 \cdot v)$.

Notice that all the moves (and hence v) include the dummy vehicle as an alternative vehicle for order relocation.
We observe that mR is inspired by the *related removal* neighborhood illustrated in Section 4.7. Moreover, the 2C Neighborhood generalizes the *ejection chain* concept presented in Section 4.6, and again, it is adapted to multi-pickup and multi-delivery orders.
All the neighborhoods are explored according to a steepest descent strategy (2.4.2). The evaluation of the solutions generated by the above moves always involves two routes that are modified in compliance to Assumption 5.1. Hence, the evaluation of a neighbor solution can be performed by calling twice the second-level heuristic, which provides new feasible routes that satisfy Assumption 5.2.

5.6 Exploration strategy: Tabu Variable Neighborhood Search

In this section we show two different strategies we devised for the exploration of the neighborhoods considered in the Variable Neighborhood structure of the main algorithm. In particular, we describe a sequential change of neighborhood (corresponding to the Variable Neighborhood Descent technique) and a new cyclic change of neighborhood. Moreover, we also outline two different ways to update the current solution on no improving iteration, as well as the tabu list behavior and the overall stopping conditions.

5.6.1 Sequential neighborhood switch

In the sequential switch strategy, neighborhoods are sorted by increasing size (1R, 2S, mR and 2C) and explored according to a Variable Neighborhood Descent technique: at each iteration a neighborhood is explored if and only if the previous one does not provide an improving solution. After each full neighborhood exploration, if the best solution found improves the current solution, it is chosen as the next solution in the Tabu Search trajectory. The exploration strategy SF is included as a component of Algorithm 6.

5.6.2 Cyclic neighborhood switch

In the sequential neighborhood switch described above, the Tabu VND tries to escape from 1R local optima before allowing non-improving moves. This implies iterating the search of computationally expensive neighborhoods as 2S, mR and 2C whenever a local optimum is reached, until an improving solution is found. We consider a different exploration schema, such that we always allow a maximum number I_n of non improving 1R moves before switching to one of the neighborhoods 2S, mR or 2C. The exploration switches back to neighborhood 1R as soon as an improving solution is found or, in any case, after a maximum number I_b of iterations. We implemented two alternative switching methods: in the first (denoted by SF), at each switch, we choose, in the order, either neighborhood 2S, mR or 2C; in the second (SR), we choose the switching neighborhood at random, with probability proportional to a measure of its strength. For the sake of a fast overall procedure, the strength of each neighborhood is evaluated offline by running a simple Tabu Search using only that neighborhood, and measuring the average improvement with respect to the initial solution.

5.6.3 Deterministic and random exploration

If an improvement is not generated by any of the neighborhoods, the choice of the next solution is either deterministic or random. Deterministic exploration selects the best non-improving solutions. Random exploration chooses the new current solution at random among the best five generated by any explored neighborhood, with probability proportional to the score function (roulette wheel selection). In this case, the overall exploration can be repeated to obtain hopefully better solutions. In Figure 5.1, an example of the benefit coming from the random exploration is shown. Notice how, in this sample execution, the deterministic is stuck cycling, whereas the random searches, after a deteriorating phase, manage to attain better solutions.

5.6.4 Tabu list and termination criteria

The tabu moves are encoded as one or more triplets (r_i, o_k, r_j), representing the fact that order o_k has been removed from route r_i and inserted in route r_j. Any move relocating the order o_k into r_i is considered tabu as long as there exists at least one triple (r_i, o_k, r_j) in the tabu list, where r_j is any route other than r_i.
To prevent search space exploration from cycling through the same order-to-vehicle assignments, the tabu list stores the last T moves yielding selected current solutions: a neighbor is declared tabu and temporarily forbidden if it moves at least one order to a vehicle it was assigned in the last T iterations.

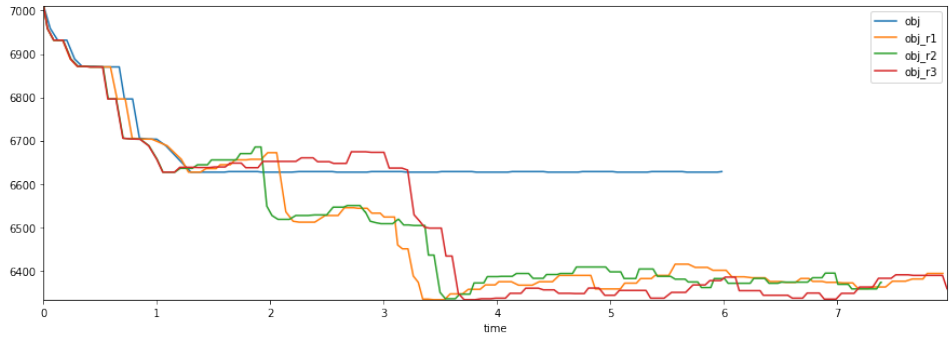

Figure 5.1: Objective values over time of the deterministic exploration (obj) and three random explorations (obj_r1, obj_r2, obj_r3) during a sample execution.

The first-level tabu search stops as soon as a maximum number M_1 of iterations without improvement or a maximum total number M_2 of iterations is reached.

5.7 Overall outline of the first level heuristic

In Algorithm 9 we provide a pseudo-code of the first-level Tabu Variable Neighborhood Descent, denoted as $L1$. For sake of simplicity, we consider only the initial solution provided by BI and we assume exploring the search space according to the sequential neighborhood strategy, corresponding to VND. Given an empty (or even partial) solution s, $L1$ calls procedure BI to obtain a complete initial solution (Step 2). Notice that BI relies on the second level heuristic $L2$ to evaluate the cost and the feasibility of the tentative routes it considers. Then, Steps 3 and 4 initialize the center and the best solution, together with the total iteration counter k, the non-improving iteration counter l and the tabu list T. Steps 5 to 35 are the main tabu search loop that explores the EPDT solution space. At each iteration, all the neighbors of the current center solution are visited by the inner loop (Steps 9 to 24), and a pool of the best 5 solutions, according to Z_1 score function, is maintained (Steps 7 and 22). The inner loop iterates on the neighborhood type N (set to 1R, 2S or mR in the order) and implements the variable neighborhood exploration strategy: each inner iteration generates all the neighbor solutions of type N, taking the tabu list T into account and calling the second level heuristic L_2 to evaluate the cost and check the feasibility of the order-to-vehicle assignments associated with each solution (Step 10).
After exploring a neighborhood (in a steepest descent fashion), if a neighbor solution is generated with better Z_1 score than the current center one,

the inner variable neighborhood loop is broken and the improving solution becomes the center for a new tabu search iteration (Steps 11 to 20), after updating the tabu list (Step 34) and, in case, the best solution found so far (Step 16). On the contrary, if the inner loop generates no improving solution (flag *improved* remains false), the center of the next tabu search iteration is chosen from the pool: the best one in case of deterministic exploration, or the one determined by roulette wheel selection in case od stochastic exploration (Steps 27 to 32). The main loop stops after matching the termination criteria based on the tabu search iteration counters l and k (Step 5).

5.8 Granular Tabu Search

The overall neighborhood evaluation may be computationally expensive, in particular for mR, due to its large size. Therefore, we propose a granular exploration of mR, by limiting moves to the ones involving most promising order subsets, selected on a distance criterion: the more orders are "close" to each other, the more they are likely to be conveniently accommodated in the new route.

Definition 5.1. *Let o_1 and o_2 be two orders, with sets of tasks $\mathcal{T}(o_1)$ and $\mathcal{T}(o_2)$ respectively. Given $\rho \in [0,1]$ and $\delta > 0$, orders o_1 and o_2 are* close *to each other if and only if*

$$|\{i \in \mathcal{T}(o_1) | \exists j \in \mathcal{T}(o_2) : d_{ij} \leq \delta\}| \geq \rho \cdot |\mathcal{T}(o_1)|$$

and

$$|\{i \in \mathcal{T}(o_2) | \exists j \in \mathcal{T}(o_1) : d_{ij} \leq \delta\}| \geq \rho \cdot |\mathcal{T}(o_2)|.$$

The previous definition states that two orders are close if at least a threshold fraction ρ of the tasks of o_1 are within a threshold distance δ from at least one of the tasks of o_2, and vice versa. Notice that the definition is symmetric with respect to any pair of orders.

Definition 5.2. *Given a set of orders $\Gamma = \{o_1, \ldots, o_n\}$, we define the* proximity graph of Γ *$G(\Gamma) = (\Gamma, A_\Gamma)$ as an undirected graph where A_Γ is the set of arcs and $(o_i, o_j) \in A_\Gamma$ if and only if o_i and o_j are close to each other.*

We associate to a route r a proximity graph defined on the set of non-fixed orders assigned to r. Based on this graph, we filter the mR neighborhood by applying moves only to subsets of orders corresponding to cliques of cardinality at most m in the proximity graph.

5.9 Destroy and Repair phase

As already observed, the second-level local search heuristic is an as critical as computationally expensive task. With the aim of reducing running

Algorithm 9: First-level heuristic

```
1  def L1(s, explorationStrategy):
2      s = BI(s, L2);
3      center = best = s;
4      k = l = 0, T = [center];
5      while l < M_1 and k < M_2 do
6          k = k + 1;
7          pool = [ ];
8          improved = false;
9          for N in [1R, 2S, mR] do
10             neighbors = generate_neighbors(center, N, T, L2);
11             n̄ = arg max{Z_1(n) : n ∈ neighbors};
12             if Z_1(n̄) > Z_1(center) then
13                 center = n̄;
14                 l = 0;
15                 improved = true;
16                 if Z_1(n̄) > Z_1(best) then
17                     best = n̄;
18                 end
19                 break
20             end
21             else
22                 pool = take_best_5 (pool ∪ neighbors, Z_1);
23             end
24         end
25         if not improved then
26             l = l + 1;
27             if explorationStrategy is deterministic then
28                 center = arg max{Z_1(n) : n ∈ pool};
29             end
30             else
31                 center = roulette_wheel(pool);
32             end
33         end
34         update_tabu_list(center, T);
35     end
36     return best;
37 end
```

times, we speed up the evaluation of the score function by separately sequencing today and tomorrow tasks. The sequence of tomorrow tasks is relevant in the score function for the evaluation of the prospective tomorrow cost, which, moreover, takes only distance into account. Observing that today tasks precede tomorrow tasks in any feasible sequence, we propose the following *fast route sequencing*: we decompose a route r in two subsequences including, respectively, today and tomorrow tasks. The second level heuristic limits the local search procedure to today tasks, and, then, a nearest neighbor heuristic sequences tomorrow tasks: starting from the last today task, the next task is the one compliant with pickup and delivery precedence, and minimizing the distance from the current task. This allows reducing the computational effort but has the drawback of yielding possible infeasibilities in the route, e.g. the violation of hard time windows or capacity constraints, although limited to tomorrow tasks. With the aim of taking the computational benefits, we devise the following phases:

1. *Optimize*: we run the first-level tabu search calling the second-level heuristic with fast route sequencing, obtaining a solution s.

2. *Check*: for each $r \in R(s)$, compute tomorrow tasks' starting times and capacity needs and check feasibility. If all routes are feasible, stop and return s. Otherwise, let t_i^r be the first task of r violating some constraints.

3. *Destroy*: for each unfeasible route r, remove all the orders containing task t_i^r or any of the following tasks, and assign them to the dummy vehicle.

4. *Repair*: run the first-level tabu search calling the second level heuristic with the usual route sequencing (local search and feasibility check extended to tomorrow tasks).

Notice that the Destroy phase brings feasibility back, which is preserved by the Repair phase. Therefore the overall procedure ends up with a solution satisfying all the constraints (but, in case, the one on mandatory orders, penalized by the score function Z_1). The proposed approach takes advantage from the relatively small number of tomorrow tasks with respect to the today ones in express delivery contexts, so that we expect that infeasibilities rarely occur.

5.10 Parallel Design

In this section we explore different techniques for low-level and high-level (see Section 2.4.4) parallel implementation on some specific components of the main algorithm. We take into account both CPU multi-threaded implementations and GPU (Graphics Processing Unit) meta-heuristic design.

In the former case, we develop two strategies, based on (high-level) inter- and (low-level) intra-neighborhood exploration respectively. In particular, the inter-neighborhood routine makes use of the main parallel strategy in [25]. For the latter, we analyze the main features of the GPU hardware that have an impact on the design of algorithms in order to attain a substantial efficiency. Meta-heuristic algorithms may contain several tasks that can be executed on a large set of parallel threads (an introduction can be found in [19]), and this is the case for local search algorithms [89] or large neighborhood search [88].
We want the parallel design to implement a steepest descent neighborhood exploration, as for the standard implementation.

5.10.1 Parallelization on CPU

To further reduce running times while preserving the quality of solution, we devise two parallel implementations of the first level tabu-search: parallel neighborhood, exploring neighborhoods 1R, 2S and mR in parallel, and the parallel evaluation, where the solutions of each neighborhood are evaluated in parallel.

The *parallel neighborhood* (PN) procedure considers one independent thread for each tabu search neighborhood. Each thread explores one neighborhood and stops as soon as (i) it evaluates all the neighbors or (ii) another thread evaluates all neighbors and finds an improving solution. In the deterministic exploration version, synchronization issues relate to solution improvement notification. In addition, in the stochastic version, we need to synchronize the pool of the best solutions when the last thread terminates.

The *parallel evaluation* (PE) procedure partitions the set of solutions of each neighborhood, and devotes an independent thread to each part. Stopping conditions and synchronization issues are similar to the ones for PN.

Notice that, in deterministic search, PE runs the same steps as the corresponding sequential implementation and, hence, provides the same (unless ties occur) final solution. On the contrary, even in deterministic search, PN may in principle select the best neighbor from a different neighborhood and end up with a different solution.

5.10.2 Parallelization on GPU

The GPU device was originally conceived to provide high performance computing for computer graphics purposes. The GPU hardware consists of several multi-processors that are optimized for massive parallelism, a requirement that may fit other scenarios than graphics. The straightforward approach is to generate and evaluate each neighbor on a single thread. Nevertheless, the GPU works in a completely different manner than the CPU,

so one must take care of peculiar aspects when designing optimization algorithms in order to have an actual advantage in terms of efficiency, namely:

- the memory transfer from the CPU to the GPU and vice versa is considerably slow. The transfer duration is called *latency*. For this reason, one must try to keep the data transfer as low as possible;
- the CPU and GPU can work asynchronously. This means that one can try to design its algorithm in such a way that computations are performed by at least one device while the data transfer is ongoing (*latency hiding*);
- the number of threads for parallel computations must be very large, in order to exploit the full potential of the hardware and justify the overhead of threads' scheduling. On the other hand, the threads should not be overloaded with memory usage, in order to prevent them from using slower memories when their own register memories are full. The *occupancy* is a measure of the ratio between the working threads over the total number of threads available;
- in the current architecture, the GPU executes the same instructions on 32 contiguous threads (*thread warp*) reading from 32 contiguous data allocated in the memory (this is called SIMD: Same Instruction Multiple Data). This means that, any time different instructions are required for a warp, the GPU must run sequentially different instructions on the 32 data. Therefore, different instructions on contiguous threads may deteriorate the efficiency up to a factor 32 (*divergence*);
- any time GPU memory is accessed, data are transferred by 128Byte buffers (*cache lines*), regardless of the amount of data requested.

We now briefly describe a new exploration design for the neighborhoods suitable for the GPU implementation. Let us consider the 1R neighborhood (the same concepts can be applied to any other neighborhood). We recall that the neighbors generation is performed in the following two steps:

1. for each pair of routes r_i and r_j in the center solution, 1R moves each order o from route r_i to route r_j;
2. at each assignment of o to r_j, the procedure L2 is triggered.

For simplicity, we assume that procedure L2 consists of only the *task_insertion* phase. Also, we restrict the problem to the single-pickup and single-delivery case. Under these hypothesis, at Step 2, procedure L2 generates the insertion of the pickup and delivery of o at position respectively l and m of r_j. In order to exploit the availability of a large number of threads, we combine

points 1 and 2 so that a single neighborhood 1R $\times$ L2 generates the assignments of order o to route r_j with pickup at position l and delivery at position m. On the one hand, the complexity of this neighborhood is suitable to cope with the occupancy issue on the GPU. On the other hand, the divergence of instruction is limited, since each thread runs only the evaluation of the move. Once all the neighbors are generated, we determine the best neighbor using a parallel *reduction* algorithm. The parallel reduction algorithm partitions the array of neighbors into pairs and applies the binary operator that compares the value of the objective function. The operator returns the neighbor with minimum objective function. The reduction executes this procedure until it finds the best neighbor over all the neighborhood in $O(\log_2 n)$ steps.

Chapter 6

Bounding through a Column Generation Algorithm

In order to assess the performance of the proposed algorithm for EPDT, we introduce an Integer Programming model based on set covering formulation. In this chapter we describe the Column Generation algorithm we used to solve the continuous relaxation of the model. With reference to the notation in Table 3.1, we will describe in details the Master and Pricing Problems and how they are tailored on specific features of EPDT: the Master Problem consists of a set covering model able to take into account open routes, while the Pricing Problem is an ESPPRC solved through a new label correcting algorithm based on the one described in Section 2.3.2 with relevant adaptations for label definition, resource extension and dominance rules to cope with the multi-pickup and multi-delivery attribute.
This method provides a bound to optimal solutions of EPDT that we can use to assess the effectiveness of the Tabu Variable Neighborhood Descent described in Chapter 5. The content of this chapter is summarized in [30].

6.1 An Integer Linear Programming formulation

As stated in the problem description in Section 3.4, we can model EPDT on a complete graph $G = (N, A)$ where nodes N represent tasks and each arc in A represents the link between an ordered pair of tasks. We now further detail that formulation in order to obtain an ILP model for EPDT. For each vehicle $v \in V$, N includes the following dummy nodes:

- α_v initial dummy tasks;
- ω_v final dummy tasks;
- ω_v^{eod}, the preferred or prescribed end-of-day position, in case v has one.

In particular, α_v and ω_v and, in case, ω_v^{eod} are associated with dummy mandatory orders, added to the set O^M.
Given a vehicle $v \in V$, any feasible route for v can be seen as a path in G that starts in α_v and ends in ω_v. Moreover, only v can visit α_v, ω_v and ω_v^{eod}. Let Ω^v be the set of paths in G corresponding to routes that are feasible for v. We remark that each path $r \in \Omega_v$ is elementary, i.e., r visits each node at most once.
For the sake of the Column Generation approach, we focus on the case where the two terms of the objective function that simultaneously take more than one route into account are negligible, namely the preferences on collecting orders in the same route $I_D(s)$ and the minimum number of vehicles with depot as end-of-day position $I_E(s)$. Hence, we do not take attributes S2 and S3 (see Section 3.4) into account.
For each path $r \in \Omega^v$, we denote its objective coefficient by c_r: it includes both the actual cost of a route and penalties for soft constraints violation related each route singularly: following the assumption above, we set $w_D = w_E = 0$ and, according to equation (5.3), we have that

$$c_r = Z_2(r), \tag{6.1}$$

where we recall that, for $w_D = w_E = 0$, $Z_2(r)$ is defined as

$$Z_2(r) \;=\; C(r) + A(r) + W(r) + w_F I_F(r) - w_H I_H(r) + w_J I_J(r).$$

As in Section 3.4, y_r are binary variables equal to 1 if path r is in the solution, 0 otherwise, and x_o are binary variables equal to 1 if and only if order o is assigned to no route in the solution. We recall that q_o is the coefficient of variable x_o in the objective function and it is defined as follows: $q_o = P_o$, if $o \in O^N$, and $q_o = P_o + L_o$, if $o \in O^U$, with P_o and L_o being the profit of order $o \in O$ and the cost for missed urgent order $o \in O^U$, respectively. EPDT can be thus formulated as:

$$\min \sum_{v \in V} \sum_{r \in \Omega^v} c_r y_r + \sum_{o \in O \setminus O^M} q_o x_o \tag{6.2}$$

$$\text{s.t.} \sum_{v \in V} \sum_{r \in \Omega^v} a_{ir} y_r \geq 1 \quad \forall\, i \in N : o(i) \in O^M \tag{6.3}$$

$$\sum_{v \in V} \sum_{r \in \Omega^v} a_{ir} y_r + x_{o(i)} \geq 1 \quad \forall\, i \in N : o(i) \in O \setminus O^M \tag{6.4}$$

$$\sum_{v \in V} \sum_{r \in \Omega^v} y_r \leq |V| \tag{6.5}$$

$$y_r \in \{0, 1\} \quad \forall\, r \in \Omega^v, v \in V \tag{6.6}$$

$$x_o \in \{0, 1\}, \quad \forall\, o \in O \setminus O^M \tag{6.7}$$

where, as in Section 3.4, a_{ir} is a constant equal to 1 if path r includes task node i, 0 otherwise, and $o(i)$ is the order i belongs to.

Objective function (6.2) is equivalent to maximizing the score (5.2), noticing that: (i) by definition $c_r = Z_2(r)$; (ii) by (6.3) no mandatory order is missing; (iii) the score can be written as

$$\sum_{o \in O} P_o - \sum_{o \in O \setminus O^M} q_o x_o - \sum_{v \in V} \sum_{r \in \Omega^v} c_r y_r; \tag{6.8}$$

and (iv) the first term in the previous equation is constant.
A solution satisfying (6.3) visits all tasks belonging to mandatory orders at least once, whereas, by (6.4), tasks in non-mandatory orders are not visited if and only if the corresponding order is not assigned. Finally, constraint (6.5) bounds the number of routes by the number of available vehicles. This constraint, together with (6.3), also guarantees that each vehicle v runs exactly one route, as shown by the following Lemma.

Lemma 6.1. *Given a feasible solution* $(\bar{y}, \bar{x})$ *to (6.3-6.7),* $\forall\ v \in V, \exists!\ r \in \Omega_v : \bar{y}_r = 1$.

Proof. For each vehicle v, there is at least one dummy task (e.g. α_v) belonging to a mandatory order that can only be included in Ω^v, hence, by (6.3), at least one variable $y_r, r \in \Omega^v$ has value 1. Moreover, if more than one variable take value one, constraint (6.5) would be violated. □

Model (6.4) is a set covering formulation, which is convenient from a computational point of view [10], but allows feasible solutions visiting tasks more than once. To this end, we notice that, in order to include preferences on order assignment and end-of-day position, route costs may include negative terms. Therefore, Remark 4.1 is not applicable to the formulation (6.2-6.7). We may adopt a set partitioning formulation with (6.3-6.4) turned into equalities, however the following theorem shows that solving the set covering formulation is equivalent to solving the set partitioning one.

Theorem 6.2. *Given an optimal solution to (6.2-6.7), it is always possible to derive an equivalent optimal solution with constraints (6.3-6.4) tight.*

Proof. Let (y^*, x^*) be an optimal solution and let $i \in N$ be a node satisfying either (6.3) or (6.4) strictly. If the solution is such that $y^*_r = 1$ only for paths r with non-negative cost components, we obtain another feasible solution that does not increase the cost (hence optimal) as follows: we remove i from all routes (but one if $o(i) \in O^M$ or $x^*_{o(i)} = 0$), and, if defined, we set $x^*_{o(i)} = 0$ (notice that $q_{o(i)} \geq 0$). Hence, recalling that feasible paths ar elementary, we can restrict to the case when visiting $o(i)$ strictly reduces the cost of two paths r or s such that $y^*_r = y^*_s = 1$ (if more than two variables are equal to one in y^*, we iterate the argument). This can happen if (Case 1) i has a preferred vehicle or (Case 2) i is the final dummy task of a vehicle with preferred end-of-day position. Let $v(r)$ (resp. $v(s)$) be the vehicle associated

with r (resp. s): by Lemma 6.1, $v(r) \neq v(s)$. In Case 1, by definition of EPDT, i has at most one preferred vehicle, say, without loss of generality, $v(r)$, so that removing i from s does not increase c_s and yields a solution with less paths covering i and non-increased cost. In Case 2, i appears only in either $\Omega_{v(r)}$ or $\Omega_{v(s)}$ which excludes that both r and s visit i. □

6.2 The Master Problem

An upper bound for EPDT can be obtained by solving the linear relaxation of (6.2-6.7), where the domain of y and x variables is extended to real non negative values. By non-negativity of q_o, upper bound constraints $x_o \leq 1$ can be neglected. This can also be done for y variables, by the same argument of Lemma 6.1. The size of sets Ω^v is very large and we adopt a Column Generation approach (see Section 2.1.3), where the linear relaxation of the model defined in (6.2-6.7) represents the Master Problem. We start from restricted sets $\bar{\Omega}^v$ and we dynamically add routes up to reaching an optimal solution. The y variables associated with initial sets $\bar{\Omega}^v$, together with variables x, should define an initial feasible solution. To this end: (i) we initialize each Ω^v with the route through tasks of fixed orders assigned to v (as an alternative, routes from the tabu search heuristic can be used), and (ii) we add a further dummy variable y_0, related to a dummy route collecting all remaining tasks but the ones of mandatory orders, with $c_0 = M$, M being a very large constant. The resulting model defines the Reduced Master Problem (RMP). After solving RMP, we exploit dual information to recover a column to be conveniently added, if any. To this end, we consider the dual variables $\pi_i \geq 0$ associated with constraints (6.3-6.4) for each node i, and $\mu \leq 0$ associated with (6.5), and we solve, for each vehicle $v \in V$, the following Pricing Problem that we can derive similarly to Section 4.5:

$$\text{PP}(v): \qquad \bar{c}_v = \min_{r \in \Omega^v} \left\{ c_r - \sum_{i \in N} a_{ir} \pi_i - \mu \right\}. \tag{6.9}$$

For each vehicle v such that $\bar{c}_v < 0$, the corresponding y_r can be conveniently added to the RMP, otherwise we have found the optimal solution to the linear relaxation of (6.2-6.7).

6.3 The Pricing Problem

PP(v) is an Elementary Shortest Path Problem with Resource Constraints (ESPPRC) see Section 2.3.2) on the subgraph $G(v) = (N(v), A(v))$ induced on G by the tasks that can be visited by v (corresponding to $N(v)$). ESPPRC can be solved by label correcting algorithms, as we have seen in Section 2.3.2: they build partial paths from α_v to $i \in N$ and make use of labels

to store information on partial cost, use of resources (e.g. time, capacity), order status etc. Partial paths and related labels are recursively extended to reach further nodes up to ω_v and, in principle, all paths are considered and the best one is chosen. In order to reduce the number of labels to be computed and stored, dominance rules are defined, allowing for deleting dominated paths, i.e., paths to a node i whose feasible extensions up to ω_v are also feasible for another cheaper path to i.
In the following we will describe how we adapted the general schema of Section 2.3.2 to the attributes of EPDT. In particular, classic dominance rules cannot be applied for the multi-pickup and multi-delivery precedence constraint, therefore we defined a new resource with a tailored dominance rule to fathom sub-optimal extensions with respect to this attribute.
We associate to each arc $(i, j) \in A(v)$ the time distance t_{ij} and a cost coefficient $\bar{d}_{ij}$ defined as

$$\bar{d}_{ij} = \tilde{w}_A(j)\, C_v^U\, d_{ij} - \tilde{w}_H(j, v) - \tilde{w}_F(j, v) - \tilde{\pi}_\alpha(i) - \pi_j \tag{6.10}$$

where:

- $\tilde{w}_A(j) = \kappa_A$ if j is a tomorrow task, 1 otherwise;
- $\tilde{w}_H(j, v) = w_H$ if j is the preferred end position for v, 0 otherwise;
- $\tilde{w}_F(j, v) = \dfrac{w_F}{|\mathcal{T}(o(j))|}$ if $o(j)$ has v as preferred vehicle assignment, 0 otherwise (we recall that, from Table 3.1, $\mathcal{T}(o(j))|$ is the set of tasks of the order $o(j)$);
- $\tilde{\pi}_\alpha(i) = \pi_i$ if $i = \alpha_v$, 0 otherwise.

Notice that for the preferred vehicle assignment we divide the weight w_F by the number of tasks of the order $o(j)$, since any feasible route will eventually complete that order, thus including in the cost the whole value of w_F.

6.3.1 Labels

A label represents a path from α_v by means of a tuple $L = (R, C)$ with $R = (t^W, t^R, l^1, l^2, \mathcal{O}, \mathcal{U})$, where:

1. i is the last node of the path;
2. t^W is the cumulative working time;
3. t^R is the cumulative driving time;
4. l^1 is the volume on board after visiting i;
5. l^2 is the weight on board after visiting i;

6. $\mathcal{O}$ is the set of open orders, i.e., orders with at least one visited pickup node and at least one delivery node not yet executed (orders started but not completed);

7. $\mathcal{U}$ is the set of unreachable nodes (delivery tasks with not-yet-executed pickups or already visited nodes)

Notice that the definition of L has been adapted to EPDT and, in particular, $\mathcal{O}$ and $\mathcal{U}$ account for open multi-pickup and multi-delivery orders. We recall (from Section 2.3.2) that, in pickup and delivery contexts, $\mathcal{U}$ usually reports the set of unreachable *orders* or equivalent information (see, e.g., [14, 54]). In fact, if orders include one pickup and one delivery task, the information on unreachable orders, combined with the status reported by $\mathcal{O}$, allows univocally identifying the task (pickup or delivery) that can be visited preserving elementary paths and precedence constraints as we have seen in 2.3.2. This is not the case in EPDT: e.g., once an open order is identified, information on unreachable order is not enough to identify which further pickup or delivery tasks must still be visited. As a consequence, we define $\mathcal{U}$ such that information on unreachable specific tasks is available and can be exploited to, e.g., avoid visiting the same node or, as we will see, define sufficient dominance conditions. We remark that resource $\mathcal{O}$ is redundant (the state of an order can be retrieved from $\mathcal{U}$), and used to speed up dominance checking as we will see in the following.

6.3.2 Label extension

A label to node i is extended to a label to node j by adding an arc (i, j) and updating the resources accordingly (see, e.g., [14]):

- t^R is obtained from travel times t_{ij};
- t^W is obtained from travel times t_{ij}, duration and time window of j and, when needed, short and night breaks;
- we add or subtract the weight and volume of j to l^1 and l^2 respectively, depending on j is a pickup or delivery task.

Concerning specialized resources for EPDT:

- $\mathcal{O}$ is initialized to the empty set and: we add $o(j)$ if j is its first pickup task to be visited; we remove $o(j)$ if j is its last visited delivery task;
- $\mathcal{U}$ is initialized with all delivery nodes and we extend it as follows: we add node j (elementary path); if j is the first pickup of $o(j)$ we add ω_v (the path cannot terminate with a new undelivered order on board) and (when defined) ω_v^{eod}, if $o(j)$ has further today tasks (we have today tasks to visit before the end of the day); if j is the last pickup of $o(j)$

we remove all deliveries of $o(j)$ from $\mathcal{U}$ (deliveries become reachable as soon as all pickups are visited); if j is the last delivery node on board, we remove ω_v (all orders have been completely delivered and we can reach the dummy final node); if j is the last today task on board, we remove ω_v^{eod}, if defined (the vehicle can visit the end-of-day dummy task).

Notice that only feasible extensions are considered, so that the ones violating hard time windows or capacity constraints are discarded.
Finally, given a label $L_i = (R_i, C_i)$ on a node i, and considering its extension to the node j with time window identified by $[t_j^b, t_j^e]$, the computation of the new cost C_j is obtained by

$$C_j = C_i + \bar{d}_{ij} + \tilde{w}_D(t_j^W, j)$$

where

$$\tilde{w}_D(t_j^W, j) = \begin{cases} w_J \cdot \max(0, t_j^W - D) & \text{if } j = \omega_v \\ L_j^w \cdot \max(0, t_j^W - t_j^e) & \text{otherwise,} \end{cases}$$

with D is the preferred maximum route duration (see Table 3.1).

6.3.3 Dominance

Given a label L, we denote by $i(L)$ the last node of the path represented by L; $t^W(L)$ the working time in L; $t^R(L)$ the driving time; $C(L)$ the partial path cost; $l^1(L)$ the vehicle load volume; $l^2(L)$ the vehicle load weight; $\mathcal{O}(L)$ the set of open orders in L; $\mathcal{U}(L, o)$ the set of unreachable nodes in L related to task of an order $o \in O$; $\mathcal{U}(L, -)$ the set of unreachable nodes in L with no associated order (e.g. initial and final dummy tasks). Given two labels L_1 and L_2, L_1 dominates L_2 if all the following conditions hold:

$$C(L_1) \leq C(L_2) \tag{6.11}$$
$$i(L_1) = i(L_2) \tag{6.12}$$
$$t^W(L_1) \leq t^W(L_2) \tag{6.13}$$
$$t^R(L_1) \leq t^R(L_2) \tag{6.14}$$
$$l^1(L_1) \leq l^1(L_2) \tag{6.15}$$
$$l^2(L_1) \leq l^2(L_2) \tag{6.16}$$
$$\mathcal{U}(L_1, -) \subseteq \mathcal{U}(L_2, -) \tag{6.17}$$
$$\mathcal{U}(L_1, o) \subseteq \mathcal{U}(L_2, o) \quad o \notin \mathcal{O}(L_1) \cup \mathcal{O}(L_2) \tag{6.18}$$
$$\mathcal{U}(L_1, o) = \mathcal{U}(L_2, o) \quad o \in \mathcal{O}(L_1) \cup \mathcal{O}(L_2). \tag{6.19}$$

In fact, dominance states that L_1 has a smaller cost, has consumed less resources and, moreover, L_1 has less unreachable nodes and, hence, L_1 has

at least the same feasible extensions as L_2. Concerning condition (6.19), we notice that it states equality for orders that are open in at least one label. In fact inclusion, sufficient for the classic pickup and delivery constraint, is not enough in a multi-pickup and multi-delivery scenario, as shown by the following example. Assume order o includes three pickup and one delivery tasks and let L_1 represent a path visiting one pickup task, and L_2 include one more pickup; clearly $\mathcal{U}(L_1, o) \subset \mathcal{U}(L_2, o)$, but the feasible extensions of L_2 do not include the task in $\mathcal{U}(L_2, o) \setminus \mathcal{U}(L_1, o)$, whereas feasible extensions of L_1 have to. Therefore, extensions of L_2 are not feasible for L_1, meaning that L_1 does not dominate L_2. If an order is not open in both L_1 and L_2, then it is either completed (all pickups and deliveries are unreachable) or not in charge (all deliveries and no pickups are unreachable), so that in this case inclusion is sufficient, as requested by (6.18).

Chapter 7

Toward new Data-Driven approaches for Dynamic and Stochastic VRPs

The EPDT problem stated at Chapter 3 is inspired by a real-world context, where dynamic requests are issued during the planning horizon. Moreover, stochastic components affect several aspects of the problem. Methods that can take into account the dynamism of the demand and the stochasticity in play are needed in order to guarantee stability during the execution of the routes. An interesting situation that is often encountered in real-world problems similar to EPDT, is the case of *dynamic and stochastic customers* where, as described in Section 4.8, customer demand and locations are revealed during the execution of the planning and, in addition, stochastic information is available offline.
In this chapter, we make preliminary studies and, to this end, we consider simplified versions of EPDT, based on the scenario adopted in [43]. In that work, authors solve a Dynamic and Stochastic Pickup and Delivery VRP with Time Windows (DS-PDPTW) define on a grid network, by means of a simulation-based anticipatory method that assumes known probability distribution of the customers demand. In particular, the authors show how in their experiments the anticipatory policy outperforms the reactive counterpart in terms of solution quality.
Our aim is to analyze the possibility of exploiting a priori statistical information on future demand by embedding it into methods that are devised for deterministic versions of the same routing problem, thus obtaining an anticipatory procedure (see Section 4.8.3) for the dynamic and stochastic case with small or no impact on the method itself. Moreover, we suppose that no probability distributions are known about the requests issued by the customers, whereas we assume the availability of historical data, which is nowadays often the case in transportation companies databases. In par-

ticular, we make use of Machine Learning methods to extract statistical information that, in turn, is preprocessed and can be exploited by the optimization procedure.
The first method is based on the introduction of representative orders that are added to the problem instance and aggregate anticipated space and time information on the demand, thus driving the optimization algorithm toward more stable solutions with respect to future requests. The approach we propose is similar to the one proposed in [40], where the definition of representative orders is based on a cluster analysis of data on delivery orders. Our contribution is mainly twofold. From the one side, we extend the cluster analysis to take pickup-and-delivery orders into account, so that a more complex feature space has to be considered. From the other side, we devise a more general approach, since it makes no assumption on distribution of customer requests over time, and it is more suitable for real cases where just historical data on orders is available, as for the application context that inspired this thesis. To this end, since the effects of introducing artificial orders depends on the characteristics of the available data, we also propose an a-posteriori measure of the quality of the obtained clusters intended for driving their actual utilization in anticipatory algorithms, also according to the analytical determination on a further simplified problem. The second method relies on the discretization of the space and time in order to compute an accessibility measure (like e.g. the one defined in [51]) of the discrete points, with the aim of introducing a special component in the evaluation of a solution that takes into consideration the opportunity for a route of interacting with future requests. The method, to the best of our knowledge, is new to the literature, and aims at hybridizing methods for the Dynamic and Stochastic VRP with concepts, like the accessibility, borrowed from spatial interaction analysis.
In this chapter we will focus on the dynamic settings typical from EPDT context, which is characterized by a low degree of dynamism and no diversion of vehicle from their current destination. We remark that, however, as shown by the cited literature, anticipatory policies provide in general a valid approach and, in fact, they may take advantage, when the degree of dynamism increases. Hence, the same observations offer a valid theoretical approach even in this case.

7.1 Dynamic and Stochastic VRP context

In this section we define the operational context of our study, inspired by the DS-PDPTW problem studied in [43].
Let $G = (N, A)$ be a graph, where N is the set of nodes (where each node represents a pickup or delivery operation) and A is the set of arcs between pair of nodes. We consider a fleet V of vehicles and a set of n_O pickup and

delivery orders O such that

$$O = \{(p_k, d_k, T_k) : k = 0, 1, \ldots, n_O\},$$

where p_k, d_k and T_k are, respectively, the pickup node, the delivery node and the time of request of the order.
In order to represent the dynamic scenario, we need the definition of the *subset* $O(\tau) \subseteq O$ *of known requests at time* τ:

$$O(\tau) = \{(p_k, d_k, T_k) \in O : T_k \leq \tau\},$$

that represents the requests received within time τ. In addition, given a time τ, we define the following sets of orders:

- the set $O_P(\tau)$ of *pending orders* as the set of open orders in $O(\tau)$ at a time less than or equal to τ;
- the set $O_F(\tau)$ of *free orders* as the set $O(\tau) \setminus O_P(\tau)$.

The dynamic version includes new decision variables, that represent the *waiting times* for vehicles at each node of their routes. A route r is then defined as

$$r = \{(i_0, \ldots, i_L), (w_0, \ldots, w_L)\},$$

where $i_l \in N$ for each $l = 0, \ldots, L$ are the nodes visited and each w_l is the waiting time corresponding to node i_l.
In this model, routes must start and finish at the depot. If a vehicle is already in the trip toward some location, it cannot be redirected to a different destination, i.e., *diversion* is not allowed (see Section 4.8.3). Furthermore, idle vehicles can also be *relocated* to a location $l \in \bar{N}$, where $\bar{N} \subset N$ is the set of home locations.
We remark that, because diversion is not allowed, idle vehicles, as well as the ones with assigned orders, may start a trip toward any destination and choosing appropriate waiting times may be beneficial.
Let $\tilde{t}(i)$ be the time of arrival at the node i. We define a penalty function that expresses the delay with respect to the request time and a soft deadline D for a delivery d_k:

$$W(\tilde{t}(d_k), T_k) = \max(0, \tilde{t}(d_k) - T_k - D).$$

The objective of the problem is to minimize the total penalty associated with the customer service

$$z = \sum_{(p_k, d_k, T_k) \in O} W(\tilde{t}(d_k), T_k).$$

Notice that, as stated in [43], soft deadlines are introduced to model, e.g., same-day courier companies in urban areas: their clients often require an

express service for deliveries of letters and small items. The inconvenience of late service can be represented by a soft deadline after which the customer satisfaction decreases as the amount of delay grows.
We underline that this scenario includes two new type of decisions: the waiting time at each node of a route and the relocation of idle vehicles. Moreover, the operational context assumes triggering an optimization phase each time a new request is received. This means that at each optimization phase, all the free orders, i.e. the last arrived, as well as the ones assigned but not yet executed, are dynamically (re-)assigned.
In [43], the DS-PDPTW problem is solved by an anticipatory technique based on simulation. In the following we will explore different anticipatory approaches that aggregate the statistical information through data-driven methods in a preprocessing phase and provide algorithms with components that guide them toward solutions that are in principle able to better accommodate the future demand.

7.2 Data

A main characteristic of the scenario under study is the presence of historical data of the customers' requests. As we have seen in Chapter 1, nowadays transportation companies make use of different tools that store a large amount of data, like e.g. vehicle tracking tools, real-time fleet management platforms, order booking portals, etc. The availability of such data fosters the research toward data-driven approaches that can be an asset for decisions in different aspects of the transportation companies management, as for example planning or marketing.
Given M as a non negative integer number, a record of a *historical data set* is defined as a M-tuple $(\delta_1, \ldots, \delta_M)$ of features. In this general description, δ_i can belong to any type of domain for each $i = 1, \ldots, M$. Since data are supposed to be collected during different working days, we say that each single order belongs to an *instance* of the problem, where an instance corresponds to the set of input data of the problem in a single day. Given a set of E instances, we associate a unique index $\eta_k \in \{1, \ldots, E\}$ with each record of data in the historical data set. Hereinafter, we assume that a historical data set H contains records of past customers' requests

$$H = \{(\delta_{k1}, \ldots, \delta_{kM}, \eta_k) : k = 1, 2, \ldots, n_H\},$$

where n_H is the amount of records in the historical data set and δ_{ki} is the i-th feature of the k-th record.
Moreover, given an instance $\tilde{\eta}$, we define as the *historical data set restricted to instance* $\tilde{\eta}$ the set

$$H(\tilde{\eta}) = \{(\delta_{k1}, \ldots, \delta_{kM}, \eta_k) \in H : \eta_k = \tilde{\eta}\}$$

7.3 Data-driven instance augmentation

The approach described in this section is based on the introduction of artificial orders, called *representative orders*, that are added to the instance before the optimization phase. The representative orders consist of strategic space-time positions that aim to favor solutions that are more stable with respect to future demand. In our approach, the representative orders are determined by means of machine learning methods run on the historical data. In the following we outline the method with reference to the DS-PDPTW defined in Section 7.1. In order to obtain a set of representative orders, we propose a technique based on clustering, inspired by the strategy used in [40].
Moreover, taking advantage from the aggregated information in the representative orders, we will suggest relocation and waiting time strategies.

7.3.1 Overall Procedure

The representative orders technique is based on using an optimization algorithm for DS-PDPTW designed for the deterministic (static or dynamic) scenario on an augmented instance, which include a set of n_R representative orders O_R defined as

$$O_R = \{(p_m, d_m, T_m) \in m = 1, \ldots, n_R\}.$$

In our case, such an algorithm may be the Tabu VND outlined in Chapter 5 where, in order to overcome the loss of efficiency due to the extra orders in input, only the 1R neighborhood is considered. We define $O_R(\tau)$ for a time τ as the set of representative orders with time of request less than or equal to τ:

$$O_R(\tau) = \{(p_m, d_m, T_m) \in O_R : T_m \leq \tau\}.$$

The steps that outline the algorithm able to cope with the DS-PDPTW are summarized in the following list:

- at each optimization phase at time τ, (triggered as explained at 7.1), the optimization algorithm inserts in the solution the set of orders $O_P(\tau) \cup (O_R \setminus O(\tau))$;
- representative orders can be either removed from the solution once it has been computed, or kept to perform the *relocation* of idle vehicles: a vehicle at a representative order's task is in principle in a more convenient position to reach future requests issued in the space-time region spanned by the corresponding cluster;
- given a route $r = \{(i_0, \ldots, i_L), (w_0, \ldots, w_L)\}$, we set to 0 every waiting time w_l such that $o(i_{l+1}) \notin O_R$. For all the other waiting time variables, given a pair of consecutive nodes i_l and i_{l+1} of a route, where

$o(i_{l+1})$ is the m-th order $(p_m, d_m, T_m) \in O_R$, the waiting time w_l is set in such a way that the arrival time $\tilde{t}(i_{l+1})$ of the vehicle at i_{l+1} is as close as possible to, but not smaller than the time of request T_m of $o(i_{l+1})$:

$$w_l = \max\left(0, T_m - \tilde{t}(i_l) - t_{i_l, i_{l+1}}\right),$$

where $t_{i_l, i_{l+1}}$, according to the notation in Chapter 3, is the time distance between nodes i_l and i_{l+1}. We underline that a vehicle must respect the associated waiting times along its route as long as the next task corresponds to a representative order. As soon as a task of a dynamic order becomes the next operation to perform, the waiting time schedule is overridden and the vehicle starts its next trip immediately.

7.3.2 Representative Orders

In order to create a set of representative orders, we rely on a machine learning technique that builds clusters from the historical data set and provides a set of representative orders O_R. These orders correspond to the centroids of the clusters and, by the centroid definition itself, each of them represents a point in space-time that aim to minimize the (squared) space-time distance to reach the orders in the same cluster. We suppose that the nodes lie on $\mathbb{R}^2$, and we consider the following historical data set H_1 with n_H data:

$$H_1 = \{(p_k^x, p_k^y, d_k^x, d_k^y, T_k, \eta_k) : k = 1, 2, \ldots, n_H\},$$

where features consist of, in the order, pickup x and y coordinates, delivery x and y coordinates, time of request and instance.
In order to balance the features processed by the clustering algorithm, we standardize the data set H_1 by rescaling them so that the distribution of each single feature has zero mean and unit variance. We consider the matrix H_{std} that is built by appending all the elements in H_1 on multiple rows and dropping the feature η_k for all $k = 1, \ldots, n_H$.
Given a matrix A, we denote by A^j its j-th column. We devised three different clustering techniques for H_{std}, all based on the K-Means algorithm combined with a *hierarchical* setting:

PDT this clustering considers all the features (Pickup coordinates, Delivery coordinates, Time of request) of H_{std} with no hierarchical setting;

PDt this clustering first clusters the features in $(H_{std}^1, \cdots, H_{std}^4)$ (Pickup coordinates, Delivery coordinates), that we call *first level clustering*. After that, it executes a new clustering, that we call *second level clustering*, restricted to H_{std}^5 (time of request) on data in each partition of H_{std} that has been previously generated;

Tpd this clustering first clusters the feature in H^5_{std} (Time of request), that we call *first level clustering*. After that, it executes a new clustering, that we call *second level clustering*, restricted to $(H^1_{std}, \cdots, H^4_{std})$ (pickup coordinates, delivery coordinates) on data in each previously generated partition of H_{std}.

In order to have a control on the number of clusters to generate, we adopt a K-Means algorithm. A reasonable choice is to approximately fix this parameter to the average number of orders during the planning horizon, in order for the representative orders to be "close" to the unknown orders not only in space and time, but also in cardinality. Given a clustering procedure P among PDT, PDt and Tpd, a hierarchical level (if defined) i and a number of clusters n_C, we denote the K-Means algorithm at the i-th level in procedure P with n_C clusters by

$$K^i_{\mathrm{P}}(n_C).$$

We now suppose that the historical data set H_1 contains data relative to J instances. For each clustering approach, we compute the value n_C as follows:

1. for the $K_{\mathrm{PDT}}(n_C)$, we set n_C at the mean number of orders per instance
$$n_C = \left\lfloor \frac{1}{J} \sum_{j=1}^{J} |H_1(j)| \right\rfloor,$$
2. for the $K^1_{\mathrm{PDt}}(n^1_C)$ and $K^2_{\mathrm{PDt}}(n^2_C)$ method, we choose the values of n^1_C and n^2_C so that their product is close to the value n_C as computed at the previous point. For this reason, we introduce an extra parameter d corresponding to the desired ratio between n^1_C and n^2_C and solve the following system :
$$\begin{cases} n^1_C = d\, n^2_C \\ n^2_C = \left\lfloor \sqrt{\dfrac{n_C}{d}} \right\rfloor \end{cases}$$
3. for the $K^1_{\mathrm{Tpd}}(n^1_C)$ and $K^2_{\mathrm{Tpd}}(n^2_C)$, we determine n^1_C and n^2_C as for the previous point.

In Figure 7.1, we see an example of a representative order together with the projection of the corresponding cluster on the pickup and delivery subspace in a grid graph. The red dots corresponds to the projections of the centroid in the pickup and delivery sub-space, whereas the blue dots are the projections on the same spaces of pickups and deliveries from historical data, the thickness of the dots is proportional to the frequency they appear in such cluster.

Therefore, given a clustering procedure P, the set of representative orders

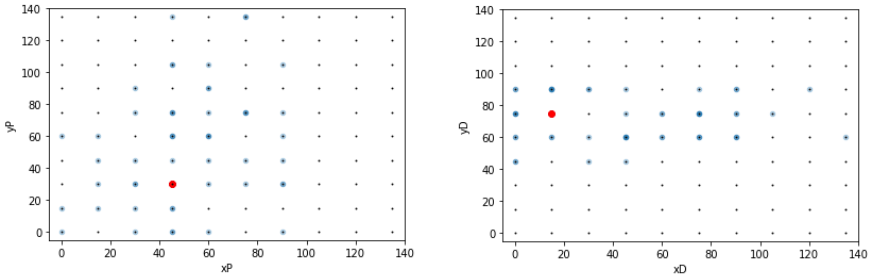

Figure 7.1: Projection of a representative order on the pickup axis (left) and delivery axis (right).

is

$$O_R = \{\mu_{C_m} : m = 1, \ldots, \tilde{n}(P)\},$$

with μ_{C_m} the centroid of the m-th cluster obtained with P, and $\tilde{n}(P)$ is n_C if $P = \text{PDT}$, $n_C^1 \cdot n_C^2$ otherwise.

7.3.3 Choosing relocation positions and waiting times

The clusters provided by the first phase of our approach have different features depending on the distribution of orders in the historical data set H_1.

Definition 7.1. *Given a cluster C, we define the* sparsity *of C $S(C)$ as*

$$S(C) = \frac{1}{|C|} \sum_{l \in C} d(l, \mu_C),$$

where $d(\cdot, \cdot)$ is the distance induced by the 2-norm in the feature space $\mathbb{R}^5$, and μ_C is the centroid of the cluster C.

Definition 7.2. *Given a cluster C in the historical data set H_1 with J instances, we define the* frequency *of C $F(C)$ as*

$$F(C) = \frac{1}{J} \sum_{j=1}^{J} Bool \left(\sum_{r_k \in H_1(j)} \chi_C(r_k) \right),$$

where $Bool(x)$ takes value 0 if $x = 0$ and 1 otherwise, and $\chi_C(y)$ is the characteristic function of the cluster C.

We make the following observations:

1. the *larger* the sparsity $S(C)$ of a cluster C, the *worse* the quality of its centroid as strategic space-time point to satisfy dynamic requests. Indeed, a sparse cluster in H_1 aggregates a set of locations and times that are not concentrated, so that moving from the μ_C will probably result in a high cost;

2. the *larger* the frequency $F(C)$ of a cluster C, the *higher* the chance that a dynamic request will fall in the region covered by C. This means that μ_C will likely be representative for at least one future request.

We thus introduce the concept of *reliability* associated with a representative order.

Definition 7.3. *Given a cluster $C \in \mathcal{C}$ in the historical data set H_1 and two parameters $\alpha, \beta \in \mathbb{R}_{\geq 0}$ we call* reliability factor *the value defined as*

$$R(C) = \alpha \cdot F(C) - \beta \cdot S(C).$$

The reliability factor of a cluster consists of a linear combination of sparsity and frequency and represents the *quality* of a cluster. The higher the value of the reliability, the most fruitful is expected to be the usage of such cluster in the optimization phase. Parameters α and β must be tuned in order to assign suitable weights to each of the two components of the reliability factor.
The reliability factor can be exploited mainly in the two following ways:

- after the optimization run, the representative orders can be removed or kept in the solution, representing a relocation of idle vehicles. The decision whether relocating or not a vehicle can be performed by comparing the reliability factor of the incumbent representative order with a given threshold value: ideally, relocation should be made only with high reliability representative orders;

- in the optimization method, the departure time of a vehicle toward the pickup or delivery location of a representative order is modulated by the reliability factor: the lower the quality, the more the vehicle waits before leaving. This is justified by the fact that, without diversion, a moving vehicle cannot be re-routed, and therefore a trip toward areas with poor statistical benefit should be avoided.

7.3.4 Analytical insight of waiting times

As we noticed above, the impossibility to divert the current trip of a vehicle when a re-optimization is triggered affects the benefit of the representative orders: on the one hand, a vehicle in a strategic location is likely to reach the new requests at a small cost; on the other hand, if a new request is close to a traveling vehicle, we have to wait until it makes the next stop and, then,

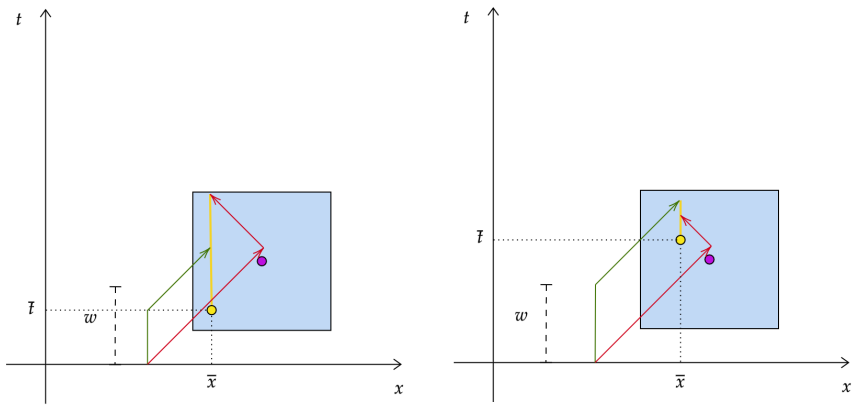

Figure 7.2: An example of how different waiting times (w or 0 in this example) outperform each-other in two different scenarios.

we will be able to send it to fulfill the new order's operations. In fact, it is not trivial to determine the waiting times for idle vehicles, since it depends on multiple factors, even related to the distribution of the requests over the space-time domain.

Figure 7.2 illustrates a 1-dimensional case (space axis x) where the same waiting time can guarantee an earlier or a later arrival with respect to no waiting time, depending on the outcome of the space-time (x,t) position of a sampled order to reach in two different realizations, that are shown at the left and at the right of the same figure. In both cases, we consider two vehicles with same speed, one (in green) waits the waiting time w before departing, the other (in red) leaves as soon as possible toward the centroid (purple). The yellow point and the corresponding line represent the presence at time $\bar{t}$ of the task of the sampled order at position $\bar{x}$, assuming the support of the probability distribution is within the blue square. We can see that different waiting times (w or 0 in this example) outperform each-other in two different scenarios. In fact, in the first case (graph on the left of Figure 7.2) the green vehicle that has to wait (at most) w can go straight to the incumbent order location, as the order is issued while the vehicle is idle, whereas the red vehicle is already directed toward the center and cannot be diverted to the new order location, so that it arrives later than the first vehicle. In the second case (graph on the right of Figure 7.2), the request arrives late: the red vehicle, that leaves immediately, arrives earlier than the green one, that waits w, as the second leg of its trip starts from a more convenient position.

In Section 7.3.3 we have introduced the reliability factor $R(C)$ that may be used to heuristically solve the problem of assignment of the decision variables $w_{1j}, \ldots, w_{nj}$, trusting a representative order o_R, corresponding to

a cluster C_{o_R}, according to $R(C_{o_R})$: the higher the reliability, the shorter the waiting time. In order to preliminary assess this policy, we make an analytical inspection of a simplified version of this problem, thus providing an insight of the difficulty in the determination of the relocation policy and of the waiting time variables, even in a restricted context.
In particular, we make the following assumptions:

Assumption 7.1. *Orders consist of a single task (pickup or delivery, indifferently).*

Assumption 7.2. *We consider* 1 *vehicle,* 1 *random order and* 1 *representative order.*

Assumption 7.3. *We assume to have a known probability distribution of the random order on a subset of the space-time domain of requests.*

Our goal is to answer the question: *"which is the waiting time for the vehicle before leaving toward the operation of the representative order that minimizes the average arrival time?"*.
We show in the following an analytical approach for the solution of this problem.

Notation

- $\hat{x}$ and $\hat{t}$ are the starting points in respectively space and time of the vehicle;
- v is the speed of the vehicle;
- $d(x, y)$ is the distance between x and y;
- c is the operation of the representative order;
- w is the waiting time variable;
- t_{max} is the maximum request time;
- Θ is the set of outcomes for the random order;
- X and T are random variables representing the task location and request time of the random order. Thus, a random order is defined by the pair $(X, T) \in \Theta$;
- $p(x, t)$ is the probability density function of the random order (X, T);
- $p_X(x)$ and $p_T(t)$ are the marginal probability density functions of the random variables X and T respectively;
- $\tilde{T}(X, T)$ is the arrival time at X for the random order defined by X and T;

Determining the optimal Waiting Time

As stated in the notation, the random variable $\tilde{T}$ is a random variable depending on the random order (X, T). We can write $\tilde{T}$ as

$$\tilde{T} = \min(\hat{t} + w, T) + \tag{7.1}$$
$$+ v^{-1} d(\hat{x}, X) H(\hat{t} + w - T) + \tag{7.2}$$
$$+ (\max(v^{-1} d(\hat{x}, c), T - \hat{t} - w) + v^{-1} d(c, X)) H(T - \hat{t} - w), \tag{7.3}$$

where $H(t)$ is the *unit step function* defined as follows

$$H(t) = \begin{cases} 1 & \text{if } t \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The term 7.1 in the sum models the fact that if the random order is requested at a time T, the vehicle leaves at the same time only if it was in the waiting status, that means $T \leq \hat{t} + w$. On the contrary, if T falls later than $\hat{t} + w$ the departure of the vehicle is triggered at the $\hat{t} + w$.
The second term 7.2 is not null only when $T \leq \hat{t} + w$ and computes the travel time from $\hat{x}$ to X.
The third term 7.3 assumes a positive value only when $T > \hat{t} + w$. In this case, since the departure takes place earlier than the time of request T, the time is computed in two pieces, called legs: the first leg refers to the trip from $\hat{x}$ to the centroid c, the second leg is from c to X. Notice that the computation of the first leg travel time includes a maximum function that handles the case when the random order is requested during the first leg trip or later. In the latter case, the vehicle will be idle at c until the time of request T.
We now make explicit the dependence of the expected value of $\tilde{T}$ on the variable w. We want to examine the sign of the first derivative of $\mathbb{E}[\tilde{T}](w)$ in order to find the minimum as a function of w, that is the value w^* such that

$$w^* = \arg\min_w \left\{ \mathbb{E}[\tilde{T}](w) \right\}.$$

Notice that the value of w^* represents the best waiting time in order to minimize the expected value of the arrival time $\tilde{T}$.
In order to find the value of w^* that minimizes the average arrival time, we introduce the following theorem:

Theorem 7.1. *Let (X, T) be a random order with X and T independent random variables such that $T \sim \mathcal{U}([\hat{t}, t_{max}])$. Given $\Theta = [\hat{t}, t_{max}] \times R$ with $R \subset \mathbb{R}^n$, let c be a point in R and $d(\cdot, \cdot)$ a distance function defined on R. Then, the average waiting time is a monotonic function of w. In particular:*

$$\frac{d}{dw} \mathbb{E}[\tilde{T}](w) \geq 0 \Leftrightarrow \mathbb{E}_X[d(\hat{x}, X)] \geq \mathbb{E}_X[d(c, X)].$$

Proof. We want to compute the derivative with respect to w of the function

$$\mathbb{E}[\tilde{T}](w) = \int_{\Theta} \tilde{T}(x,t,w)p(x,t)dxdt.$$

In order to better handle the step function H, we introduce a partition of the outcomes space dependent on the variable w:

$$\Theta_1(w) = \{(x,t) \in \Theta : t \leq \hat{t} + w\} \tag{7.4}$$
$$\Theta_2(w) = \{(x,t) \in \Theta : \hat{t} + w < t \leq \hat{t} + w + v^{-1}d(\hat{x},c)\} \tag{7.5}$$
$$\Theta_3(w) = \{(x,t) \in \Theta : t > \hat{t} + w + v^{-1}d(\hat{x},c)\}. \tag{7.6}$$

This allows us to define for each $i \in \{1,2,3\}$ the functions $E_i(w)$ as

$$E_i(w) = \int_{\Theta_i(w)} \tilde{T}(x,t,w)p(x,t)dxdt,$$

which in turn let us ignore the functions H in any partition, obtaining the following formula for the computation of the expected value of $\tilde{T}$

$$\begin{aligned}
\mathbb{E}[\tilde{T}] = \sum_{i=1}^{3} E_i(w) = \\
&= \int_{\Theta_1(w)} (t + v^{-1}d(\hat{x},x))p(x,t)dxdt + \\
&+ \int_{\Theta_2(w)} (\hat{t} + w + v^{-1}d(\hat{x},c) + v^{-1}d(c,x))p(x,t)dxdt + \\
&+ \int_{\Theta_3(w)} (t + v^{-1}d(c,x))p(x,t)dxdt.
\end{aligned}$$

Since X and T are independent, it holds

$$p(x,t) = p_X(x) \cdot p_T(t) \quad \forall\ (x,t) \in \Theta,$$

then, we can rewrite each function $E_i(w)$ as

$$E_1(w) = \int_{\hat{t}}^{\hat{t}+w} t\ p_T(t)dt + v^{-1}\mathbb{E}_X[d(\hat{x},X)] \cdot \mathbb{P}_T[\hat{t} < T \leq \hat{t} + w] \tag{7.7}$$
$$E_2(w) = (\hat{t} + w + \tau + v^{-1}\mathbb{E}_X[d(c,X)]) \cdot \mathbb{P}_T[\hat{t} + w < T \leq \hat{t} + w + \tau] \tag{7.8}$$
$$\begin{aligned}
E_3(w) = &\int_{\hat{t}+w+\tau}^{t_{max}} t\ p_T(t)dt + \\
&+ v^{-1}\mathbb{E}_X[d(c,X)] \cdot \mathbb{P}_T[\hat{t} + w + \tau < T \leq t_{max}],
\end{aligned} \tag{7.9}$$

where $\tau = v^{-1}d(\hat{x},c)$ is the time distance from $\hat{x}$ to c.
Thanks to the Leibniz integral rule, we now calculate from (7.7) - (7.9) the

first derivative of each of the functions $E_i(w)$, $i = 1, 2, 3$.

$$\frac{d}{dw}E_1(w) = p_T(\hat{t} + w) \cdot (\hat{t} + w + v^{-1}\mathbb{E}_X[d(\hat{x}, X)]) \tag{7.10}$$

$$\frac{d}{dw}E_2(w) = \mathbb{P}(\Theta_2) + (\hat{t} + w + \tau + v^{-1}E_X[d(c, X)])\cdot \cdot(p_T(\hat{t} + w + \tau) - p_T(\hat{t} + w)) \tag{7.11}$$

$$\frac{d}{dw}E_3(w) = -\, p_T(\hat{t} + w + \tau) \cdot (\hat{t} + w + \tau + v^{-1}\mathbb{E}_X[d(c, X)]). \tag{7.12}$$

Expressions (7.10) - (7.12) lead us to the derivative of the expected value of $\tilde{T}$:

$$\frac{d}{dw}\mathbb{E}[\tilde{T}](w) = \sum_{i=1}^{3} \frac{d}{dw}E_i(w) = \mathbb{P}_T([\hat{t} + w, \hat{t} + w + \tau]) + +p_T(\hat{t} + w) \cdot (v^{-1}\mathbb{E}_X[d(\hat{x}, X) - d(\hat{x}, c) - d(c, X)]). \tag{7.13}$$

By the type of distribution of the random variable T, together with (7.13), we have that

$$\frac{d}{dw}\mathbb{E}[\tilde{T}](w) = \frac{\tau}{t_{max} - \hat{t}} + \frac{v^{-1}\mathbb{E}_X[d(\hat{x}, X)] - \tau - v^{-1}\mathbb{E}_X[d(c, X)]}{t_{max} - \hat{t}} = = \frac{v^{-1}}{t_{max} - \hat{t}} \cdot (\mathbb{E}_X[d(\hat{x}, X)] - \mathbb{E}_X[d(c, X)]), \tag{7.14}$$

hence we obtain

$$\frac{d}{dw}\mathbb{E}[\tilde{T}](w) \geq 0 \Leftrightarrow \mathbb{E}_X[d(\hat{x}, X)] \geq \mathbb{E}_X[d(c, X)]. \tag{7.15}$$

□

Let us notice that, in the last member of (7.15), the choice of the value c can be determinant in the minimization of w. In fact, we have the following corollaries of Theorem 7.1:

Corollary 7.1. *Let (X, T) be a random order with X and T independent random variables such that $T \sim \mathcal{U}([\hat{t}, t_{max}])$. Given $\Theta = [\hat{t}, t_{max}] \times R$ with $R \subset \mathbb{R}^n$, let c be such that*

$$c = \arg\min_y \int_R d(x, y)p_X(x)dx.$$

Then $\mathbb{E}[\tilde{T}](w)$ is a monotonic increasing function, and

$$w^* = 0.$$

Proof. Trivial by the definition of c applied to the left-hand side of (7.15). □

Corollary 7.2. *Let (X, T) be a random order with X and T independent random variables such that $X \sim \mathcal{U}(R)$ with $R \subset \mathbb{R}^n$ and $T \sim \mathcal{U}([\hat{t}, t_{max}])$. Given $\Theta = [\hat{t}, t_{max}] \times R$, let c be such that*

$$c = \arg\min_y \int_R d(y, x)dx.$$

Then $\mathbb{E}[\tilde{T}](w)$ is a monotonic increasing function, and

$$w^* = 0.$$

Proof. From (7.15) we obtain that

$$\frac{d}{dw}\mathbb{E}_{XT}[\tilde{T}](w) \geq 0 \Leftrightarrow \fint_R d(\hat{x}, x)dx \geq \fint_R d(c, x)dx.$$

$$\Leftrightarrow \int_R d(\hat{x}, x)dx \geq \int_R d(c, x)dx.$$

Then the proof is trivial by the definition of c. □

By Corollary 7.1, we have that the best choice for the vehicle, in the problem under study, is to leave immediately toward the representative order's task. We observe that the value assumed by c in Corollary 7.1 is known in literature as the solution of the *continuous 1-Median (or Fermat-Weber) Problem*. We remark that the value of c in Corollary 7.2 represents the geometric median of the set R (see, e.g., [39]).
Notice also that, with no knowledge on T distribution, Equation 7.13 shows that $\mathbb{E}[\tilde{T}](w)$ may have a change of sign with respect to w, since the term $\mathbb{E}_X[d(\hat{x}, X) - d(\hat{x}, c) - d(c, X)]$ is non-positive by triangular inequality. This suggests that, in the general case, the probability on T plays a crucial role in the determination of w^*.
Although applied on a simplified problem, Corollary 7.1 seems to suggest that, given a cluster of orders, it is reasonable to start from the solution of a 1-Median problem in order to obtain the representative order, where the probability is approximated by the frequencies of orders within the cluster.

7.3.5 Towards an application to EPDT

The observations above are based on a problem representing a simplified version of EPDT. In particular, EPDT orders have a more complex structure, as they may consist of more than a single pickup and a single delivery operation. Therefore, in order to extend this method to EPDT, we need to adapt the procedure that creates the representative orders. One of the possibilities we suggest, is to approximate each historical multi-pickup and multi-delivery order with a single-pickup and single-delivery order where tasks are represented by the barycenter of, respectively, the pickups and the

deliveries of that order. Another technique may rely on the decomposition of multi-pickups and multi-deliveries into a set of approximating pairs of single-pickup and single-delivery orders, taking the peculiarities described in Section 4.3 into account. A hybrid method between the two just described may also be considered, where we substitute groups of pickups or deliveries with their barycenter only if they are concentrated in a small area, and then we apply the decomposition of the latter approach. Let us observe that, regardless of the method chosen to build the representative orders, the main procedure described in this section is not affected by an application to EPDT, as representative orders are added to the input data of any instance of the problem.

7.4 Accessibility Approach

In this section we describe another anticipatory technique that aims to cope with dynamic and stochastic routing problems that can be integrated in solution methods implemented for the deterministic case. This approach is based on the classic concept of *accessibility* [51]. The accessibility measure is mainly applied in the urban logistic theory and represents, in a network of services (e.g. schools, ATMs, hospitals, etc.), the potential of interaction at each location, i.e., how easy it is for each location to reach and use services through the network. Our aim is to apply the accessibility concept to the dynamic orders' distribution, so that we obtain an accessibility function in space and time standing for how easy it is for a vehicle in a space-time point to reach a future order from that specific location. In this way, the evaluation of a route can take into account, besides the delay on soft time windows, also the values of the accessibility in each visited nodes, in an attempt of anticipating near-future demand.

7.4.1 Overall Procedure

According to the main goal of this chapter, the accessibility approach, as the one presented in Section 7.3, relies on an optimization algorithm designed for the deterministic scenario. In particular, we may make use of the Tabu VND from Chapter 5 modified in such a way that we add to the objective function a new component $\varphi(r)$ for each route r that takes into account information on the future demand: this component is added to the second-level score function $Z_2(r)$ that evaluates a route r. Therefore the new second-level score function is

$$\tilde{Z}_2(r) = Z_2(r) + \varphi(r). \tag{7.16}$$

We propose a way to determine $\varphi(r)$ with the concept of *accessibility*. We consider a set of decision makers I and a set of services J. In order to define

the accessibility, we introduce the *utility* u_{ij} for a decision maker i of interacting with a service j: the choice of a decision maker $i \in I$ among different services $j \in J$ depends on the utility value u_{ij}. The mathematical framework that is classically used to model the decision process among multiple alternatives is the *logit model* (see [98], among others).
We define p_{ij} as the probability that the decision maker i chooses the service j: the logit model for a decision process in function of the utilities is defined by

$$p_{ij} = \frac{e^{u_{ij}}}{\sum_{j \in J} e^{u_{ij}}}, \quad \forall\, i \in I. \tag{7.17}$$

The *accessibility* for a decision maker $i \in I$ is defined as

$$\Phi_i = \sum_{j \in J} e^{u_{ij}},$$

and corresponds to the denominator of (7.17).
The connection between the accessibility theory and the routing scenario under study corresponds to:

- identify I with the set of nodes N of the graph;
- identify J with the set of requests (j, t) from each node $j \in N$ issued at time $t \in \mathcal{T}$, where $\mathcal{T}$ is the planning horizon;
- consider an accessibility measure $\Phi_i(t)$ in the space-time $N \times \mathcal{T}$ domain;

Therefore, given a route $r = (i_1, \ldots, i_L)$, we model the term $\varphi(r)$ with the cumulative accessibility collected at each node of the route:

$$\varphi(r) = \sum_{l=1}^{L-1} \Phi_{i_l}(\tilde{t}(i_l)), \tag{7.18}$$

where we recall that $\tilde{t}(i)$ is the arrival time at node i.
Our goal is to foster routes that collect more accessibility at each visited node, representing a more robust setting when the re-optimization is triggered as dynamic orders rise.

7.4.2 Data-driven accessibility measure

In this preliminary, study we drop the Pickup and Delivery attribute of the problem defined in Section 7.1, considering only the Soft Time Windows attribute associated with the deadline D. Therefore, each order defines a single pickup operation, hence a suitable historical data set H_2 with n_H data is considered:

$$H_2 = \{(p_k^x, p_k^y, T_k, \eta_k) : k = 1, 2, \ldots, n_H\},$$

where features consist of, in the order, pickup x and y coordinates, time of request and instance. We discretize the planning horizon as $\mathcal{T}^d = \{t_1, t_2, \ldots, t_d\}$, where d is the cardinality of the discretized set. Therefore, we obtain an accessibility value $\Phi_i(t)$ for each $i \in N$ and $t \in \mathcal{T}^d$. Notice that, given a route $r = (i_1, \ldots, i_L)$, the arrival time at the l-th index $\tilde{t}(i_l)$ may not coincide with any $t_m \in \mathcal{T}^d$, then we approximate it with the closest element in $\mathcal{T}^d$.

Classically, the utility u_{ij} is supposed to depend on the travel cost through the network from i to j and on the attractiveness of service W_j, measured, e.g., by the "quantity" of available service. In order to avoid an unconditional growth, the logarithm of W_j is taken [98]. Under these hypothesis, given a node $i \in N$, a time $t \in \mathcal{T}^d$ and the order j of H_2, we compute the utility $u_{ij}(t)$ by

$$u_{ij}(t) = \alpha \ln(W_j) - \beta c(i, t, j),$$

where α, β and W_j are parameters to be tuned, while the cost $c(i, t, j)$ is the delay of the order j with respect to the time window $[t_j, t_j + D]$, t_j being the time associated with j, starting the trip from i at time t.

We want to calculate an estimate of the accessibility values making use of the historical data in H_2. We assume that H_2 consists of data corresponding to Q instances, and we denote by j_d the node related to the data d in the historical data set. We define the accessibility value $\Phi_{i,\eta}(t)$ of the η-th instance as

$$\Phi_{i,\eta}(t) = \sum_{d \in H_2(\eta)} e^{u_{ij_d}(t)},$$

In order to obtain the estimate of the overall accessibility $\Phi_i(t)$, we calculate the sample mean of the single-instance accessibility values:

$$\Phi_i(t) = \frac{1}{Q} \sum_{\eta=1}^{Q} \Phi_{i,\eta}(t).$$

In Figure 7.3, we report an example of the accessibility function. The problem is modeled on a grid graph with a distance of 20 units between adjacent nodes and a planning horizon defined by the interval $[0, 480]$. The orders have been randomly sampled according to a Normal distribution in time (with mean 240 and standard deviation 100) on the two nodes with coordinates (40, 40) and (20, 120). For each node i, we show a graph of the accessibility value $\Phi_i(t)$ in function of t, computed on a set of 4000 sampled orders. Notice that, as expected, the accessibility values are very high in correspondence of the nodes where orders are requested, and gradually decay as we increase the distance from them in space. Moreover, $\Phi_i(t)$ in each i shows that the computed accessibility over time is consistent with the Gaussian distribution of the demand in time.

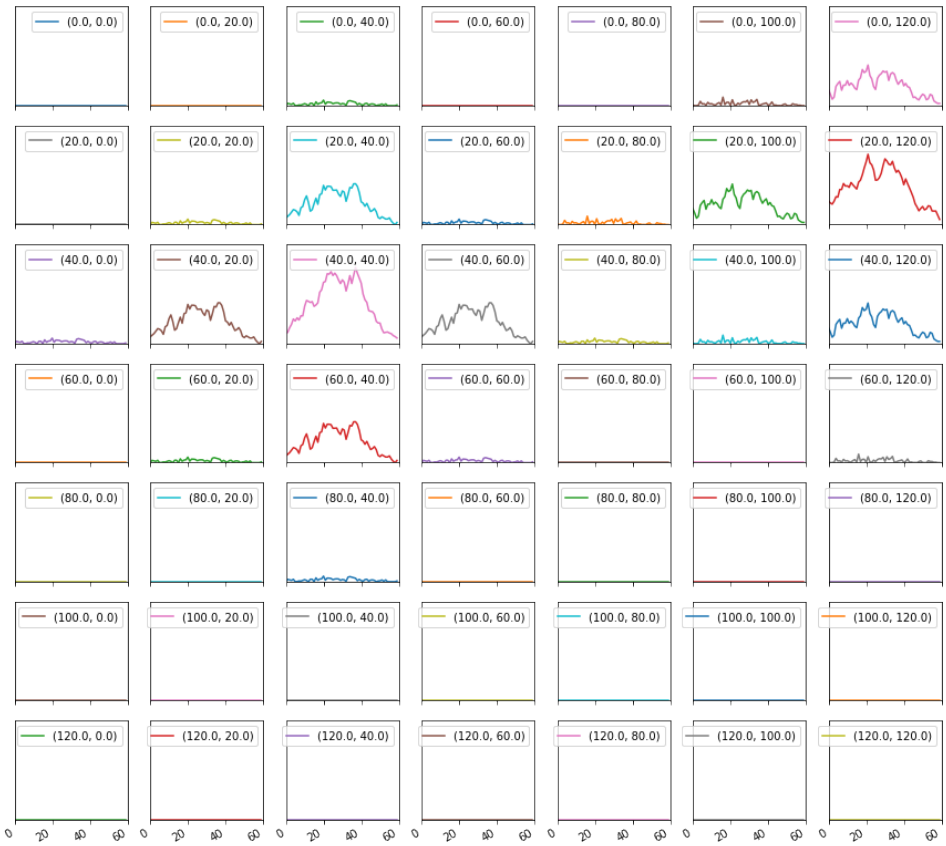

Figure 7.3: Example of an accessibility function on a 7x7 grid graph

7.4.3 Towards an application to EPDT

The accessibility approach we devised in this section is developed on a problem with a reduced set of attributes with respect to the ones in EPDT. In particular, the considered problem is characterized by orders with only a single (pickup or delivery, indifferently) task, preventing a direct application of this approach to EPDT because of the way the utility is defined. Hence, in order to overcome this issue, we can extend the utility concept in such a way that the j index in $u_{ij}(t)$ is related to a multi-pickup and multi-delivery order and W_j corresponds to the order revenue. The cost $c(i, t, j)$ can be intended as the cost for reaching each task location of order j starting from i, then we can estimate it by means of a fast TSP heuristic, as a Nearest Neighbor algorithm, with fixed initial position i. Independently from the adaptation of the accessibility computation, the Tabu VND in Chapter 5 can include the objective component corresponding to the cumulative accessibility with no further effort, obtaining then a solution method able to solve the dynamic and stochastic version of EPDT.

Chapter 8

Integration in a decision support system

Trucking companies have made a considerable effort to provide their operational managers with decision support systems to suggest optimized vehicle routing and scheduling strategies, as well as actions to dynamically react to many operational issues like vehicles or transportation network failures, real-time issued demands etc. Off-the-shelf available software include systems to track and manage the fleet in real time (e.g. [102]), to optimize truck routes (e.g. [115]), to interconnect stakeholders and to synchronize their time windows (e.g. [107]). Nonetheless, to the best of our knowledge, useful supporting functionalities are spread in different existing tools, or not present at all. In this chapter we introduce a new tool to support routing operations in small trucking companies that integrates such functionalities in a single system, together with operations research and artificial intelligence algorithms to aid the operations manager making difficult decisions. The development of such system has been carried out at the Research and Development division of Trans-Cel, a small freight trucking company located in Padova (Italy) which inspired the definition of EPDT as detailed in Chapter 3. A prototype implementation of this platform, called *Chainment*, is installed at Trans-Cel and supports the decision makers in the operations office. In particular, Chainment includes an algorithmic engine which implements the method described at Chapter 5 with specific adaptation for the platform requirements and interface toward the specific problem at Trans-Cel. A summary of this chapter content is published in [29].

8.1 Integrated support tool

The operations management office of a small trucking company has to face several decisional tasks during the working day, as shown by the following illustrative scenario.

Example 8.1.1. *Clients issue their transportation requests (in terms of e.g. freight features, pickup/delivery positions, time window preferences and dock slot availability) that are stored in an order repository. Based on this information, at the beginning of the working day,* the office assigns requests to trucks and determines their routes*. During the operations, further requests may be issued having pickup and/or delivery time windows falling in the current day. According to the real-time status of the fleet, the company has to decide if the order can be profitably taken in charge and, in case,* modify vehicle routes accordingly*. After deliveries, by exploiting the knowledge on customers' behavior, the trucking company may* solicit some orders in convenient position*, as to avoid driving empty trucks. By monitoring trucks' tank status,* fuel requests *are appropriately consolidated and negotiated with trusted fuel companies. Similarly, standard and extraordinary trucks* maintenance operations are planned *and interleaved with usual pickup and delivery operations.*

Chainment, the support tool developed at Trans-Cel, is meant to aid the decision makers in such kind of situations by means of several communicating modules, integrated through a web platform. We focus on the modules that support or interact with the vehicle routing operations, based on the current vehicles' status and planned activities, as well as on information about pending and forecast transportation requests. The platform is designed as a single-page web application according to the Software-as-a-Service (SaaS) model, that is a software available through a subscription and hosted in a cloud system. Moreover, a social network structure is applied to interconnect users (trucking companies, customers, logistics operators, fuel companies etc.), by means of user profiles, chats and real-time interaction for, e.g., orders' negotiations or direct communications between operations managers and drivers. As a part of Chainment, the modules interact by means of real-time data sharing, and take advantage from the available real-time and historical information (e.g., pending orders, vehicles, orders and road network real-time status database of past orders). In Figure 8.1, we summarize the module communication scheme. We remark that several modules use an optimization engine, that relies on the optimization algorithm solving EPDT (see Chapter 5) and on predictive models based on Machine Learning. In the following, we describe the modules included in the operations management support tool and the interface toward the optimization engine.

8.2 The Orders Portal

The Orders Portal module (Figure 8.2) supports inserting and managing orders. It collects data on orders and provides different views to both the carriers and their customers. Customers can make their own transportation

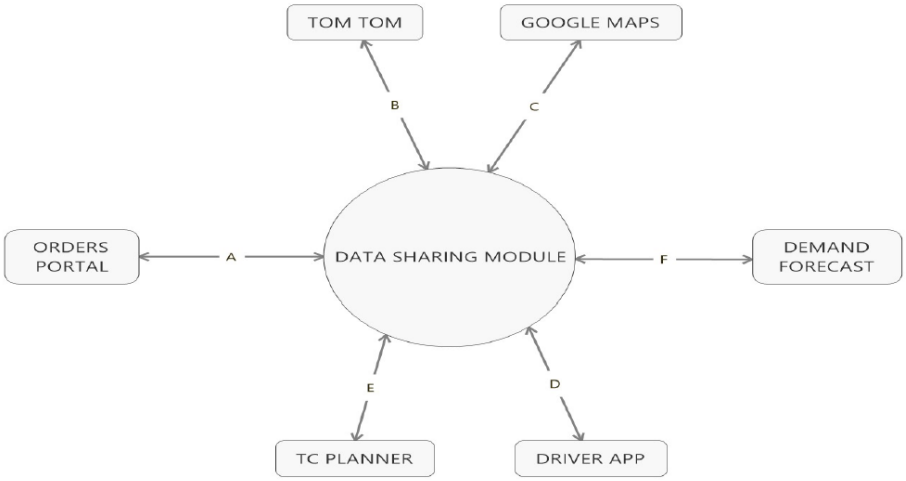

Figure 8.1: Chainment modules interaction.

requests by filling up an online form connected to the Data-Sharing module. Any change to the order data is tracked and notified to both parties in real-time. In particular, an interactive negotiation system that establishes the price of the incumbent order is a main component of such module. The carrier's operations manager has access to a profit estimator tool based on the evaluation of possible prices and marginal costs for the incumbent order, as suggested by historical and real-time data. As we will detail in Section 8.6.2, a machine learning algorithm analyses historical data to fix a suitable price interval for the order to take in charge, based on past decisions concerning similar orders, as defined by some selected order features. Moreover, as we will detail in Section 8.6.1, the Order Portal interacts with the algorithmic engine (adapted from the approach described in Chapter 5) to estimate the minimum and the maximum marginal cost of the order. This is done by dynamically inserting the incumbent order into the current route planning, taking into account the real-time status of the fleet and of the remaining currently assigned orders. The price and the cost intervals give an estimation of the possible profit related to a new order, guiding the operations manager in the negotiation process. After confirmation from both parties, orders are not changeable and appear in the data repository as orders in charge for the carrier, ready to be inserted into the daily planning (manually or by running the routing algorithm), and available for providing statistical information to, e.g., data-driven approaches.

Figure 8.2: The Orders Portal of Chainment.

8.3 The Driver App

The Driver App (Figure 8.3) handles the tracking of vehicles' operations and movements: a mobile application has been developed in order to connect the drivers to the data-sharing module. By means of this module, drivers obtain their daily routes, available either as a sequence on a timeline or in navigation mode. As they progress with the trip, the drivers send feedbacks to the data-sharing module. This allows updating the order and vehicle status, as well as collecting useful historical information, such as departure and arrival time at each position, starting and ending time of each pickup or delivery operation or other activities (e.g. breaks). Moreover, the status of the main vehicle components can be updated, as for instance the pressure value of tires, the fuel status and overall kilometers cumulated by the vehicle, for the sake of maintenance. The update can be performed either manually by drivers who can regularly insert data through the Driver App, or automatically by means of sensors, more relevant in the Industry 4.0 and Internet of Things paradigm [53]. Drivers can also make use of a photo sharing system, in order to report miscellaneous events, such as damages to the vehicle or to the packaging, or documents received by customers after completing an operation. Moreover, the Tom Tom Telematics API [102] provides further data that are automatically sent to the data-sharing module, thanks to the Tom Tom device integrated in each vehicle (e.g. driving or idle status of the vehicle, the actually traveled routes, etc.).

Figure 8.3: The Driver App of Chainment.

8.4 The Planning Module

The Planning module (Figure 8.4) determines the assignment of daily operations to vehicles and the related routes, after collecting the information on orders, fleet and road network status from the other modules, including the ones that interconnect with Google Maps and Tom Tom Telematics to obtain real-time traffic information. To this end, the module includes a routing algorithm to solve EPDT and provide a set of feasible routes. The module offers both a map view of the suggested routes and a drag-and-drop interface, which allows the operations manager to interact with the routing algorithm. The former view, thanks to the interaction with Google Maps and Tom Tom Telematics, shows both the actual route rode by each vehicle, and the routes as arranged in the current planning. The latter view allows the user creating an initial (even partial) solution and launching the routing algorithm to complete and optimize it, exploiting the fact that the method proposed in Chapter 5 for EPDT can start from any provided initial solution (see Section 5.4). The same interface gives the opportunity of overruling the routes suggested by the algorithm, by, e.g., fixing the final position of a vehicle, or the assignment of some orders to specific trucks, or a sub-route. All planning information is made available to the operations manager, such as the overall profit, the total costs of the planning, the total spatial and time distance. Moreover, any change in the planning registered through the interface can be evaluated by activating the routing algorithm to determine the impact of the modification.
At the beginning of the working day, the Planning module suggests an initial plan for daily operations after solving the static version of EPDT. In a dynamic context, during the day, the fleet, order or road network status often diverge from the initially planned ones (e.g. road network failures or con-

gestion, pickup or delivery operations taking longer than planned etc.), so that routes can be conveniently re-optimized starting from the actual status gathered from the real-time data sharing module. The Planning module is thus able to suggest possible strategies to modify the current vehicle routes in response to several dynamic events, for example:

- in case of real-time issued orders, the module runs a dynamic version of the routing algorithm in order to include the new operations into the most convenient route. To this end, the interaction with the Fleet module through real-time data sharing allows collecting the information necessary to deploy a solution that is consistent with the actual current status;

- in case of vehicle failures, the broken vehicle is excluded from the fleet, and related orders status is changed in order to let them be carried out by other vehicles. If load or unload operations are possible in the location where the vehicle broke down, a new pickup operation is associated with all orders on board, and the routing algorithm can assign the updated order to the remaining vehicles.

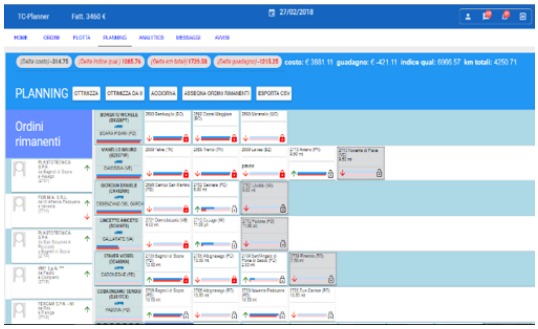

Figure 8.4: The Planning module of Chainment.

8.5 The Demand Forecast Tool

The Demand Forecast module (Figure 8.5) is designed to analyze data on past demand and predict possible future orders (e.g., at least in principle, with an algorithmic engine that solves dynamic and stochastic version of EPDT based on the framework that is outlined in Chapter 7). At the moment, the module exploits the historical information on orders to detect orders that are likely to be issued in the near future, so that they can be taken into account by the planning module. Moreover, profitable forecast orders may be detected by interaction with the Order module and proactively solicited by telemarketing, in order to increase the probability to show

up. A use case example is when a vehicle is or will be in a region with type of orders compatible with its available capacity. A marketing strategy can thus be dispatched. In order to identify fruitful regions, the module plots a heat map that shows the most visited areas: this, together with the Fleet and the Order modules, suggests convenient positions and order features that may be compatible with some running trucks. Notice that heat maps may depend on the time of the day or the day of the week, according to historical information on order features (position, date, operations time windows etc.).

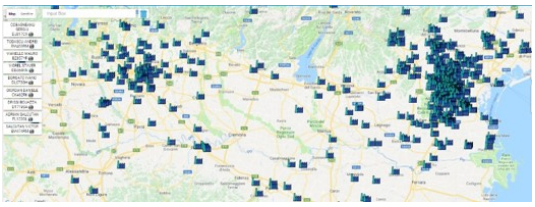

Figure 8.5: The Demand Forecast module of Chainment.

8.6 The Algorithmic Engine

The modules of Chainment base different functionalities on a common algorithmic engine, that implements a routing optimizer and predictive models.

8.6.1 The routing optimizer

In order to achieve their goals, most of the modules integrated in the support platform need to solve the EPDT problem either in static or settings. The algorithm described in Chapter 5 satisfies the requirements of the platform up to some adaptations, due to the interaction between the module and the operations manager:

1. the decision maker can suggest an initial partial or complete plan: the routing optimizer starts from that initial plan;

2. the decision maker can suggest an infeasible plan: the routing optimizer must repair the solution and then optimize it;

3. the decision maker can modify or overrule the solution: the routing optimizer cannot alter such changes;

4. the driver app provides real-time data on, e.g., actual arrival and departure time: the routing algorithm has to take the new estimates of such values available through the data sharing system.

Among others, we observe that the requirement 2 can be attained by the algorithm of Chapter 5 thanks to penalties for violation of soft constraints (S1) - (S5). In addition, we relax the problem hard constraints (H1) - (H7), penalizing their violations via extra (high) penalties to add in the objective function with the aim of guiding the search toward a feasible solution that is provided as initial solution.
Other modules solve a dynamic version of EPDT, for instance:

1. the Planning module (Section 8.4) must react to dynamic events during operations, like vehicle breakdowns, just-in-time orders, etc.;

2. the Order Portal (Section 8.2) must provide a real-time estimate of the best- and worst-case marginal costs of an order to assist in the price negotiation, as we will detail in Section 9.6.2.

Because of the low demand dynamism in the EPDT operational context, we apply a reactive strategy based on re-optimization, as stated in Chapter 5. In particular, we run the same algorithm to re-optimize the routes, using the input from different modules to collect the current vehicle, order and network status, and determine a synchronized initial solution. Moreover, for the sake of new order pricing, the order management module runs the EPDT algorithm in dynamic settings to estimate the marginal cost.

8.6.2 Predictive models

The algorithmic engine includes predictive models that are called by different modules. We devised models for the estimate of the time needed for pickup or delivery operations (service time), and a price range for an incumbent order.

Service time prediction

We implement a prediction system that provides the operational time occurring for load and unload tasks. The model is based on a Support Vector Regression that takes as input the weight and volume of the incumbent load. The tool returns the sum of the estimated operational time and the expected delay corresponding to the facility where the task is executed (e.g. waiting time in queue, slow operations setup, etc.). Notice that the latter component is easily estimated by the historical data collected by the Driver's App module. The prediction of such model is used both in the Order Portal, for providing the operations manager with a service time forecast, and by the routing optimizer, to use estimates of the service time when it is not input by the user.

Price range prediction

In the Order's Portal module, a machine-learning algorithm is run to exploit historical data in order to analyze past decisions and predict a suitable price interval. The interval is compared to the marginal cost interval estimation obtained by the dynamic insertion of the same order evaluated by the routing optimizer. The algorithm uses a decision tree to implement an ordinal classifier [41], based on orders features, namely:

1. average distance between the depot and the pickups;
2. average distance between the pickups and the deliveries;
3. total weight;
4. total volume;
5. urgency level;
6. requested truck facilities.

Given the features, the classifier applies a parametrized function that selects the interval to associate to a given order among a set of predefined intervals, sorted by average price. As usual, the parameters are obtained by training the classifier on a suitable dataset of orders. For this task, we have used a k-fold cross validation scheme, with a 4:1 ratio between training and validation sets [76].

8.7 Implementation technologies

This section provides information about the technologies used for the implementation of the algorithmic engine and its integration in the support platform developed by the Chainment team at Trans-Cel.

Project Management

The development of Chainment follows an *agile* methodology [2]. This project management system is based on the following principles:

1. Individuals and Interactions over processes and tools;
2. Working Software over comprehensive documentation;
3. Customer Collaboration over contract negotiation;
4. Responding to Change over following a plan.

These values focus on creating a development environment more suitable to comply with the evolution of software requirements and relying on a collaborative system with customers. This is in contrast with *waterfall* methodology, which starts from a detailed and inflexible project definition, taking to the finished product with poor interaction with the customer. We adopted the *Kanban* agile solution for the Chainment project. Kanban is based on the continuous improvement of the software and keeps track of the different tasks of the team by a board where columns are provided that represent the task status (i.e. to do, in progress, ready to test, done). This allows the project manager to have a insight on the team effort and bottle-necks in the project development. Among the many available, we used Trello [108] and Taiga [99] as Kanban online platforms.
UML (Unified Modeling Language [87]) has been used to describe subsystems of the overall project, whereas the documentation has been written by inline comments compliant with the Doxygen format. Doxygen [35] is a free software that generates automatic HTML and LateX documentation parsing comments which correspond to some specific format, and also automatically creates class diagrams. Some of the main tools for project development are shown in Figure 8.6.

Figure 8.6: Project Management Tecnologies

Version Control System

The version control system for repositories is Git [44]. Git (see Figure 8.7) is a free and open source software, and it is very efficient for projects of any dimension. it offers a command line interface but also a large variety of client applications (e.g. Git Bash UI, SourceTree [96], GitKraken [45], etc.) are available. Git offers the possibility of creating different work-flows at different levels of the development phase, detaching from a particular work-flow. This action is called *branching*. Multiple developers can work in a local branch, and they can synchronize such branch with a remote source git work-flows (typically labeled with the prefix *origin*), which is the central reference for all the team.

Figure 8.7: Git

Database

A SQL (Structured Query Language) database is used in the Chainment environment, with MySQL (see Figure 8.8) as database management system. MySQL [72] handles relational databases and come with a command line client where the user can interact with a server instance of the SQL server for different purposes, even though many open source client UIs are available (e.g. MySQLWorkbench [73], DBeaver [27], etc.). The two main types of database interaction are the following:

- DDL (Data Definition Language) is used to define tables and relation between them;
- DML (Data Manipulation Language) is used to perform the CRUD operations, that are the basic functionality for persistent storage of data: Create, Read, Update, Delete.
- DQL (Data Query Language) is used to perform relational algebra operation of tables, that are, e.g., joins, unions, differences, grouping of tables.

Figure 8.8: MySQL

Programming Languages

The web application is based on a server-client paradigm and the main functionalities have been implemented in PhP for the back-end (Laravel framework) and JavaScript for the front-end (Angular and Node JS frameworks). Concerning the optimization engine, which is the topic of the thesis, the routing algorithm, which asks for high levels of efficiency, has been implemented in C++ 14 (see Figure 8.9). In particular, we made use of the *template method* design pattern (see [4] for details on C++ design patterns). This pattern relies on writing the procedure of an algorithm as a skeleton whose internal routines that require specific implementation are inherited by the client. High level functionalities, such as input parsing and pre-processing from different sources, output writing, have been implemented in Python 2.7.15 (see Figure 8.9). Python [81] is a general purpose high level interpreted language with a very large availability of packages. Moreover, an interactive Python module, called IPython, has been used for debugging and analysis purposes, especially through the interface JuPyter [57]. We used the Boost C++ libraries for specific purposes, such as threading pools for multi-thread implementations and exposition of C++ classes to Python. Decoupling high and low level components through a binding between C++ and Python led to high development speed and maintainability of the code. Prediction tools have been implemented in Python, because of the large support and functionalities availability. The same language has been adopted for a fine integration of the prediction tools with the rest of the optimization engine, which already adopts Python in many routines. In particular, we used the following libraries: Sci-Kit Learn [94] is a package containing all the most popular of machine learning algorithms, Pandas [68] includes many classes suitable for data modeling, Statsmodel [90] provides several statistical analysis tools compatible with the Pandas package.

Figure 8.9: Programming Languages

Development Environments

For the development of the C++ software we made use of Visual Studio Code [114] (see Figure 8.10). This is a code editor developed by Microsoft, it has been chosen for being light-weight and being supported with a large number of packages. For more fine-grained tasks, has profiling the memory during debugging (especially for what GPU is concerned), we used Microsoft Visual Studio 15 [71], the popular Microsoft Integrated Development Environment. For Python debugging and data analysis, we made use of JuPyter (Figure 8.10), an interface for IPython which allows the execution of blocks of code and inline outputs and plots.

Figure 8.10: Development Environments

Build and deploy system

The Chainment web application is cross-platform. The deployment system is handled by Vagrant [109], a software that creates a virtual machine hosting a Chainment server instance by parsing a configuration file containing all the technological requirements and repository links. The build system for C++ code is handled by CMake [21]. CMake (Figure 8.11) is a software to configure and launch the build system (e.g. Visual Studio projects in Windows, makefiles in Unix) and let the user set all the desired build options for compiling and linking on any platform.

Figure 8.11: CMake

Third-party APIs

TomTom Telematics developers API [103] (see Figure 8.12) have been used for data like real-time coordinates related to vehicles in fleet and the reachable range of a vehicle given different residual resources (e.g. fuel in the tank or work hours available). Also, we used Google Maps developers API [49] (in Figure 8.12) for several tasks: coordinates of customer locations in the Demand Stats and route visualizations in the Planning module, as well as the distance matrix as input for the optimization algorithm. In order to reduce the number of requests to APIs, we stored in the Chainment database all the API data. Connection to APIs has been performed through the Python interface provided both by TomTom and Google Maps.

Figure 8.12: Third-party APIs

Chapter 9

Computational Results

In this chapter, we report the computational results collected during several tests over different sets of instances, both from literature and real world benchmarks. With the aim of focusing on the assessment of the algorithmic approach to EPDT as proposed in Chapter 5, we consider our experiments on real-world instances. Moreover, we show that the proposed approach can be competitive with state-of-art algorithms on problems of broader interest, by comparison on PDPTW literature benchmarks.
Tests have been run with focus on different aspects of the algorithm implementation: parameters calibration, effectiveness and efficiency assessment, as well as validation on the field in static and dynamic settings. We implemented different variants of the solution method proposed in Chapter 5, by suitably combining alternative implementations of its components: deterministic and stochastic exploration (Section 5.6.3), complete and granular exploration of the multiple relocation neighborhood mR (Section 5.8), single- and multi-thread implementations (Section 5.10), sequential and cyclic neighborhood switch (Section 5.6).

9.1 General settings

In this section we provide the general settings that we adopted during the computational campaign. We have implemented several versions of the Tabu VND heuristic for EPDT described in Chapter 5, obtained by combining the different algorithmic components that are summarized in Table 9.2. For example, we have basic components for constructive (BI, RP) and improving (DET, RND) routines or further components for neighborhood switching strategies (SF, SR), and we can add filtering (+F), neighborhoods or different parallel implementations. In general, we will describe the different versions according to the following notation: given two algorithms A and B we denote by $A \rightarrow B$ the *pipeline* execution of A and B: first algorithm A is executed, then B is run starting from the output of A. Moreover, by

Table 9.1: Parameters

Parameter	Description
	Tabu VND
M_1	the maximum number of iterations with no improvement
M_2	the maximum number of iterations
T	T is the tabu list length
	mR filter
ρ	fraction of tasks in the definition of close orders
δ	distance threshold in the definition of close orders
	Cyclic neighborhood switch
I_n	maximum number of non improving 1R moves
I_b	maximum number of iterations before restarting 1R exploration

best$(A_1, \ldots, A_n)$ we mean that algorithms $A_1, \ldots A_n$ are executed starting from the same initial solution (empty for constructive heuristics) and then the solution with best objective value is returned.
The overall procedure considered in the following are BestI, BestI1 and BestI2. In particular, BestI makes use of both BI and RP for the constructive phase, whereas BestI1 uses only BI. Moreover, BestI2 corresponds to the procedure BestI1 from which we remove the exploration of neighborhood 2C. All the algorithm versions are considered with fast second level heuristic and infeasible route fixing since, during our test, the destroy-and-repair phase has been necessary just in three cases.
All the heuristic algorithm versions, as well as the label correcting algorithm in Chapter 6 have been implemented in C++, compiled with the MSVC compiler from Visual Studio 2015. We used SCIP as a framework for the Column Generation, and SOPLEX as linear programming solver.
As reported in Chapter 5 and summarized in Table 9.1, the heuristic algorithm defines some parameters that have to be calibrated. The parameter calibrations has been performed by manual tuning, even supervised by operation managers, sometimes with tests on the field (the procedures proposed by this thesis run in the operations room in parallel with the non-supported procedure at Trans-Cel during route planning and execution). For the sake of calibration, we split the available instances into a calibration set (where manual adjusting is performed) and a test set to validate the parameters choice. Calibration and test sets are selected so that the former corresponds to about the 70% of the entire data set. The results and the benchmark features provided in the following are related to the test sets.
In the following notation, a *gap* between a solution method A with respect to a method B is defined as the percentage difference $(f(s_A) - f(s_B))/f(s_B)$ of the objective values (given f the objective function) for solutions s_A and s_B obtained by A and B respectively. Moreover, several columns in the next tables will show values as average and, in parenthesis, minimum and maximum value, namely *avg (min; max)*.

Table 9.2: Algorithmic components and related notation

name	algorithm
BI	Best Insertion
RP	Round robin insertion with Priority
DET	Tabu VND with deterministic exploration
RND	Tabu VND with 3 stochastic explorations
SF	cyclic neighborhood switch with fixed order
SR	cyclic neighborhood switch with random order
+F	granular exploration on mR neighborhood
+nPN	parallel exploration inter neighborhood
+nPE	parallel exploration intra neighborhood
± [N]	add/remove the neighborhood N
BestC	best(BI, RP)
BestI	best(BI→DET, BI→RND, RP→DET, RP→RND)
BestI1	best(BI→DET, BI→RND)
BestI2	BestI1 - [2C]

9.2 Real-world Benchmarks definitions

We gathered two distinct real-world benchmarks, *TC1* and *TC2*, by collecting instances from Trans-Cel database. The feature of TC1 and TC2 are summarized by Table 9.3 and 9.4. TC1 contains 30 instances, where each instance correspond to a single working day between March and April 2016. TC2 consists of 43 real instances corresponding to working days from March to December 2017 at Trans-Cel. We consider also a partition of TC2 into five groups TC2.i, with $i = 1, 2, 3, 4, 5$, depending on the tasks number range. The first and second column shows, respectively, the name of the groups and subgroups of TC2, together with the corresponding tasks number range. The third column gives the set size. Following columns show average and, in parenthesis, minimum and maximum number of tasks, orders and vehicles, respectively.
During the multiple tests phases, we devoted a substantial amount of time to working in parallel with Trans-Cel operations managers and analyze their planning methodology. The operations managers make use of Excel spreadsheets to create the daily routes planning: they assign orders to vehicle routes and obtain, through Excel functionalities, some basic analysis (e.g. total revenue of orders, total weight transported per route, etc.) that supports next assignments or route modifications. We exploited the availability of solutions on spreadsheets, in order to obtain objective gaps between the human and algorithmic solutions.

Table 9.3: TC1 and TC2 instances (tasks)

Group	Subgroup	Count	Pickups	Deliveries
TC1		30	21.2 (6.0 ; 39.0)	38.8 (11.0 ; 63.0)
TC2		43	27.9 (4.0 ; 44.0)	44.2 (8.0 ; 73.0)
	TC2.1 (0-40)	9	8.1 (4.0 ; 16.0)	14.9 (8.0 ; 25.0)
	TC2.2 (41-80)	9	28.6 (24 ; 36.0)	33.2 (27.0 ; 44.0)
	TC2.3 (81-90)	10	32.6 (32.0 ; 35.0)	51.8 (49.0 ; 55.0)
	TC2.4 (91-100)	9	36.1 (35.0 ; 37.0)	57.7 (56.0 ; 62.0)
	TC2.5 (101-116)	6	41.9 (38.0 ; 44.0)	66.3 (65.0 ; 73.0)

Table 9.4: TC1 and TC2 instances (orders and fleet)

Group	Subgroup	Orders	Fleet
TC1		25.6 (8.0 ; 46.0)	8.3 (3.0 ; 14.0)
TC2		27.8 (3.0 ; 55.0)	11.2 (3.0 ; 14.0)
	TC2.1 (0-40)	7.7 (3.0 ; 12.0)	7.1 (3.0 ; 14.0)
	TC2.2 (41-80)	20.7 (14.0 ; 29.0)	12.3 (10.0 ; 14.0)
	TC2.3 (81-90)	33.3 (30.0 ; 40.0)	12.0 (10.0 ; 13.0)
	TC2.4 (91-100)	38.3 (34.0 ; 44.0)	12.3 (11.0 ; 14.0)
	TC2.5 (101-116)	43.7 (36.0 ; 55.0)	12.7 (12.0 ; 13.0)

9.3 Results of basic algorithm on real-world benchmarks

In this section, we report an analysis of the basic versions of the Tabu VND heuristic for EPDT, that is, versions that do not include speed-up, filtering nor cyclic neighborhood exploration components.

9.3.1 Selecting the initial solution procedure

The first test aims at comparing the potential of the two procedures RP and BI defined at Section 5.4 to provide the initial solution to the Tabu Search heuristic. The considered algorithm versions are RP and BI combined with both deterministic and stochastic explorations, and BestI. With reference to Table 9.1, by preliminary calibration, executed as described in Section 9.1, we set $M_1 = 50$ and $M_2 = 500$, and values of T depending on the number of non-fixed orders N_{nf} in the incumbent instance. In particular, $T = 4$ if $N_{nf} < 10$, $T = 8$ if $N_{nf} > 20$ and $T = \lfloor (2/5)N_{nf} \rfloor$ in all the other cases. Notice that the considered algorithm versions do not require further parameters. In Table 9.5, we report tests on TC1 instances run on a i5-5200 2.20 GHz machine with 8 GB RAM. The results are devoted to the analysis of the performance for the different initial solution heuristics. Columns report: the configuration name; the percent improvement over the operations manager's solution; the running time in seconds; the frequency the configuration finds a solution within 1% of the best solution (provided by BestI), the frequency the configuration dominates all the others. BestI

Table 9.5: Results on real-world instances

Algorithm	Gap(%)	Time (sec.)	Win(%)	Dom.(%)
RP	-10.5 (-42.3, 5.9)	1.2 (0.0; 10.9)	–	–
BI	7.3 (-10.2; 15.3)	0.1 (0.0; 0.3)	–	–
BestC	7.3 (-10.2;15.3)	1.3 (0.0; 11.1)	–	–
RP→DET	8.5 (-3.9; 15.3)	9.6 (0.0; 53.5)	53.3	16.7
RP→RND	8.4 (-2.7; 15.3)	19.8 (0.0; 146.8)	50.0	10.0
BI→DET	9.3 (2.2; 15.3)	33.2 (0.0; 295.0)	76.7	23.3
BI→RND	9.1 (-1.3; 15.3)	72.6 (0.0; 528.7)	63.3	6.7
BestI	9.6 (2.2; 15.3)	172.3 (0.1; 551.0)	100.0	100.0

is able to provide an average 9.6% improvement over the operations manager's solution within about 3 minutes (10 in the worst case). On average, all the tested Tabu Search configurations improve over the baseline by 8.4% to 9.3%, even if, for some configurations, the column devoted to Gap shows that the operations manager still finds better solutions for some instances. The Tabu Search starting from BI heuristic and deterministically exploring the variable neighborhood seems to better trade-off efficiency, effectiveness and reliability, always providing solutions better than the baseline, with an improvement between 2.2% and 15.3% (9.3% on average) obtained within 33.2 seconds on average (5 minutes in the worst case). The last column shows that all configurations reach a strictly better solution in some instances. Notice that all the algorithms but RP give the same maximum improvement on a same medium-size instance (15.3%) where BI is able to find a good solution that cannot be further improved by tabu search. We also observe that the initial heuristic, in particular RP, may perform very poorly in the worst case, showing they are very sensitive to the input instance. If we consider the initial heuristics combined with the basic versions of the Tabu VND, we observe that starting from BI we obtain a winning solution quite frequently (more than 63% of times), while RP, although providing a good dominance rate, finds a solution within 1% from BestI only half of the times. Given the results, in the following tests we remove the RP algorithm from BestI in the constructive phase, with no sensible loss in effectiveness. For analogous reasons, we also drop the neighborhood 2C from the VND algorithm, since it rarely provides significant improvements to the solution, resulting in the version BestI2 of the algorithm.

9.3.2 Improvements from basic algorithmic components

The following tests are devoted to examine the quality improvements due to the different components of BestI2. Tests are executed on benchmark TC2, using a i5-7400 3.00 GHz machine with 8GB RAM. By the calibration of algorithm BestI2 parameters, we set $M_1 = 30$ and $M_2 = 100$, while T is defined as in Section 9.3.1.

Table 9.6: Basic algorithms vs BestI2, objective gaps

Group	BI vs BestI2 (%)	BI→DET vs BestI2 (%)
TC2.1	2.8 (0.0 ; 13.3)	0.1 (0.0 ; 0.8)
TC2.2	5.6 (0.0 ; 14.6)	0.0 (0.0 ; 0.4)
TC2.3	6.2 (0.2 ; 10.8)	0.7 (0.0 ; 3.3)
TC2.4	8.8 (3.0 ; 14.1)	0.3 (0.0 ; 1.7)
TC2.5	8.2 (3.5 ; 14.8)	0.2 (0.0 ; 1.0)
TC2	*6.2 (0.0 ; 14.8)*	*0.3 (0.0 ; 3.3)*

Table 9.7: Basic algorithms vs BestI2, running time (sec.)

Group	BI (s)	DET (s)	BestI2 (s)
TC2.1	0.0 (0.0 ; 0.0)	0.3 (0.0 ; 2.4)	1.4 (0.0 ; 9.7)
TC2.2	0.1 (0.0 ; 0.3)	3.2 (0.0 ; 8.5)	12.9 (0.0 ; 32.3)
TC2.3	0.2 (0.1 ; 0.3)	5.4 (2.4 ; 12.0)	23.7 (14.1 ; 40.1)
TC2.4	0.4 (0.2 ; 0.8)	9.0 (3.8 ; 20.1)	40.3 (18.5 ; 79.9)
TC2.5	1.1 (0.3 ; 4.3)	19.1 (4.4 ; 56.7)	77.7 (22.9 ; 196.8)
TC2	*0.3 (0.0 ; 4.3)*	*6.5 (0.0 ; 56.7)*	*27.8 (0.0 ; 196.8)*

In Table 9.6 and 9.7, we show, respectively, objective gaps and running times for each group of instances, defined in the first column of both tables. The second and third columns of Table 9.6 report the objective gap between BI and BestI2, and between DET and BestI2, respectively. Starting from the second column, Table 9.7 contains running times of BI, DET and BestI2.
Concerning the quality improvement in Table 9.6 due to different components of the algorithm, we notice that the tabu search, either with or without randomized exploration, significantly improves over the initial solution provided by BI, which is about 6% on average far from BestI2 (about 9% and up to 15% for larger instances). Adding randomization provides appreciable benefits in terms of reliability: DET solutions are only 0.3% worse than BestI2 on average, but the gap is 3.3% in the worst case. As we may expect in Table 9.7 we observe that DET and, in particular, BestI2 pays the better performance with larger running times. On average, DET is 20 times slower than BI, and BestI2 is more than four times slower than DET (from few seconds to half a minute). In the worst case, DET remains under one minute computation, whereas BestI2 may take up to about 200 seconds for larger instances.

9.4 Results of advanced algorithm settings on real world instance

In this section we analyze the effects of granular and parallel exploration, as well as of different algorithmic strategies (e.g. SF and SR). Along with the computational times, we also report the impact of these modifications on the solution quality. All the experiments have been run on the set of

Table 9.8: Effect of +F, +4PE, +4PN: objective gaps vs BestI2

Group	DET+F+4PE	BestI2+F	BestI2+F+4PE	BestI2+F+3PN
TC2.1	0.6 (0.0 ; 2.4)	0.0 (0.0 ; 0.0)	0.0 (0.0 ; 0.0)	0.0 (0.0 ; 0.0)
TC2.2	0.1 (0.0 ; 0.6)	0.1 (0.0 ; 0.6)	0.1 (-0.2 ; 0.6)	0.0 (-0.3 ; 0.6)
TC2.3	0.7 (0.0 ; 3.3)	0.1 (-0.4 ; 0.8)	-0.2 (-1.2 ; 0.5)	-0.1 (-1.0 ; 0.8)
TC2.4	0.4 (0.0 ; 1.7)	0.1 (0.0 ; 0.3)	0.1 (-0.6 ; 0.7)	0.2 (-0.8 ; 1.3)
TC2.5	0.3 (-0.4 ; 1.0)	0.2 (-0.4 ; 1.0)	0.2 (-0.4 ; 1.0)	0.1 (-0.1 ; 0.5)
TC2	*0.4 (-0.4 ; 3.3)*	*0.1 (-0.4 ; 1.0)*	*0.0 (-1.2 ; 1.0)*	*0.0 (-1.0 ; 1.3)*

instances TC2.

9.4.1 Effect of filtering and parallel explorations

The results in Table 9.8, 9.9 and 9.10 refer to tests executed on a i5-7400 3.00 GHz machine with 8GB RAM with the parameters $M_1 = 30$ and $M_2 = 100$, T defined as in Section 9.3.1. Moreover, concerning filtering calibration, we set parameters $\rho = 0.5$ and $\delta = 20$.

Table 9.8 compares different applications of filtering and parallel exploration to DET and BestI2 algorithms. For each group of instances (first column), the table shows objective gaps toward BestI2; in the second column results concern the DET configuration with 2R filtering (+F) and intra-neighborhood parallelism on four threads (+4PE); in the remaining columns algorithm BestI2 is considered with 2R filtering (+F), and results on this configuration are reported in the third column; the fourth and fifth column show results, respectively, on the application of intra- (+4PE) and inter-neighborhood (+3PN) parallelization, with, respectively, four and three threads to BestI2+F.

For each filtered or parallelized version, the solution quality is almost the same as the corresponding basic version. Filtering the deterministic version gives a slightly worse average quality: the gap just increases to 0.4% from 0.3% (reported in Table 9.6).

Figure 9.1 supports such observation by means of a box-and-whisker diagram. Notice that for BestI2+F+4PE the gaps are more concentrated toward 0 than DET+F+4PE, meaning that the degrade of the solution quality due to the filter is higher and more frequent for the DET algorithm, whereas BestI2 has a much smaller sensitivity to filtering.

The randomized exploration in BestI2 seems not to suffer from filtered 2R moves: the last row shows that the overall average solution quality remains the same and, in five instances, the different implied exploration of the solution space leads to better solutions than BestI2. In particular, BestI2+F+4PE or BestI2+F+3PN have the same average performance, and BestI2+F+4PE seems slightly better in terms of reliability (quality loss up to 1.0% instead of 1.3%).

The additional components +F, +4PE and +4PN, intended for algorithm

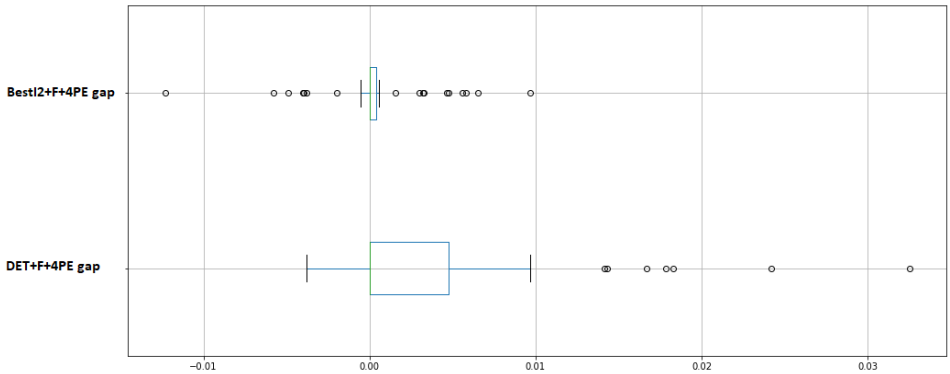

Figure 9.1: Boxplot of the objective gaps of BestI2+F+4PE and DET+F+4PE against BestI2.

Table 9.9: Effect of +F and +4PE: DET running times (sec.)

Group	DET+F	DET+F+4PE
TC2.1	0.3 (0.0 ; 1.9)	0.1 (0.0 ; 0.7)
TC2.2	1.9 (0.0 ; 5.3)	0.7 (0.0 ; 1.8)
TC2.3	4.0 (1.3 ; 8.2)	1.4 (0.5 ; 2.6)
TC2.4	4.8 (2.3 ; 9.4)	1.7 (0.8 ; 3.2)
TC2.5	12.7 (3.3 ; 36.0)	4.0 (1.2 ; 10.4)
TC2	*4.2 (0.0 ; 36.0)*	*1.4 (0.0 ; 10.4)*

speed-up, allow improving efficiency, as shown in Table 9.9 and 9.10: they report the running times, in seconds, for the different algorithm versions. The second and third columns in Table 9.9 report results on the filtered deterministic version DET+F with and without the parallel intra-neighborhood exploration (+4PE), respectively. Similarly, Table 9.10 shows the same application of filtering (+F) and intra-neighborhood parallelism (+4PE) to BestI2 algorithm in the second and third column, whereas the fourth column is devoted to the impact of inter-neighborhood exploration (+4PN).

Neighborhood filtering is quite effective, since it reduces running times by about 37% on average for both DET and BestI2. With this improvement, DET+F running times are always below 10 seconds, but for larger instances

Table 9.10: Effect of +F, +4PE, +4PN: BestI2 running times (sec.)

Group	BestI2+F	BestI2+F+4PE	BestI2+F+3PN
TC2.1	1.2 (0.0 ; 9.2)	0.5 (0.0 ; 2.8)	0.8 (0.0 ; 5.2)
TC2.2	9.2 (0.0 ; 32.2)	3.0 (0.0 ; 8.5)	4.5 (0.0 ; 15.6)
TC2.3	17.7 (6.1 ; 33.0)	5.7 (2.1 ; 9.6)	8.1 (3.8 ; 16.0)
TC2.4	20.4 (10.3 ; 34.4)	7.6 (3.6 ; 13.2)	10.8 (5.6 ; 16.3)
TC2.5	48.9 (15.3 ; 113.6)	17.8 (5.8 ; 44.5)	31.8 (6.1 ; 110.3)
TC2	*17.4 (0.0 ; 113.6)*	*6.1 (0.0 ; 44.5)*	*9.7 (0.0 ; 110.3)*

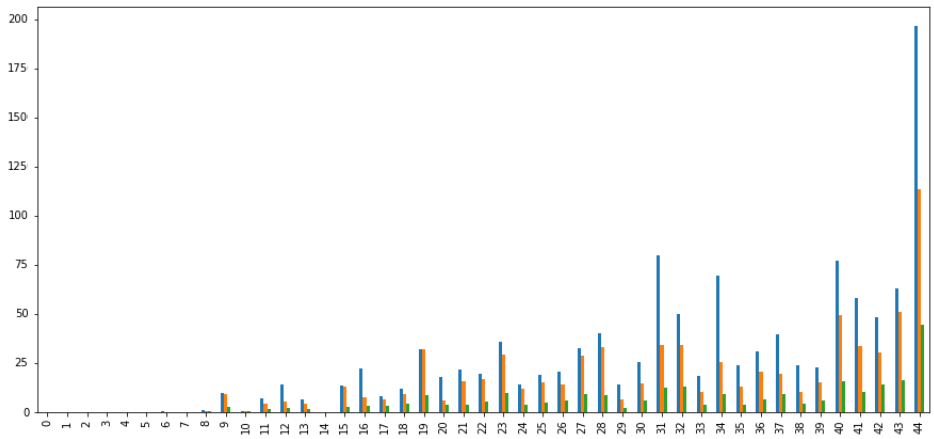

Figure 9.2: Running times (sec.) of BestI2 (blue), BestI2+F (orange) and BestI2+F+4PE (green) on TC2.

that may take up to 36 seconds to be solved, as we can see in the second column of Table 9.9. We get the most significant speedup from parallel evaluation +4PE: comparing the average running times in the last row of Table 9.10, we obtain a further average 65% time reduction by exploiting the four threads currently available on most computers. DET+F+4PE runs always in less than about 10 seconds (just 1.4 on average), as shown in the last column of Table 9.9 at the last row. Similarly, Table 9.10 shows that BestI2+F+4PE takes 6 seconds on average, 17.8 on average for large instances and always less than 45 seconds. The reduction is consistent, but less relevant, for BestI2+F+3PN, allowing a speedup of 44.5% on average upon BestI2+F. Moreover, the running time is, in the worst case, close to two minutes. In fact, PN exploits three threads and, moreover, more neighbor solutions are likely to be visited, since parallel neighborhood exploration loses some of the advantages of variable neighborhoods. In Figure 9.2, we show the run times in seconds for different combination of the algorithms executed on each instance of TC2, sorted by group size. The figure shows the trend of the running times for different configurations of BestI2 as the size of the instance increases. In particular, we see how BestI2+F+4PE has a limited growth with respect to BestI2+F and BestI2.

Summarizing, BestI2 provides the best average results in terms of solution quality, and the speeding up components (filtering and parallel implementation) preserve the quality of results: in fact, from detailed results in Table 9.10, none of these three versions dominates the others. From an efficiency perspective, BestI2+F+4PE performs better and, more interestingly, is able to limit running time to a few seconds on average and, in any case, to less then 45 seconds. This performance makes BestI2+F+4PE suitable for both daily route planning in static settings and reaction policies based on re-

optimization, due to the relatively low demand dynamism of EPDT. In fact the algorithm, integrated in the operations management office support tool, is triggered by different modules (see Chapter 8) to, e.g., accommodate on-line orders, react to unavailable infrastructures of vehicle failures, estimate costs to assist order negotiation etc. and the response times are compatible with the business model and operational settings of small trucking companies like Trans-Cel.

9.4.2 Impact of alternative exploration strategies

The impact of the new exploration schemes SF and SR is shown in Table 9.14, reporting the results of tests executed on a i5-5200 2.20 GHz machine with 8GB RAM. We used the algorithm BestI1. Since we use SF and SR strategies, we increased the number of non-improving iterations $M_1 = 60$ and we set the parameters $I_n = 15$ and $I_b = 5$ (see Table 9.1). Tests takes randomization, neighborhood filtering (F) and parallel implementation (4PE) into account. For each of the proposed switching methods SF and SR, the table reports the per-cent gap with respect to BestI1 and the running time in seconds, showing, for each instance group, the average, minimum and maximum values. Concerning SR, the probability of choosing switching neighborhood 2S, 2R or 2C is set to, respectively 55.8%, 31.1% and 13.1%. These probability values have been computed by running off-line on TC2 a Tabu Search algorithm with just one of the neighborhoods 2S, 2R, 2C and computing, for each neighborhood, the percentage of the times the related Tabu Search finds the best solution.

9.4.3 Statistical significance in algorithm selection

The comparison among different versions of the Tabu VND algorithm has been carried out along with the execution of statistical significance tests. In particular, since we cannot make any assumption on the normality of data distributions, we make use of the Wilcoxon test [83]. Given two groups of the sampled data, the Wilcoxon test provides p-values in order to measure the statistical significance of the obtained results. For sake of completeness, below we report the p-values computed for each pair of algorithm versions. Tables 9.11 and 9.12 show the p-values of different algorithm versions in terms of the obtained objective values, whereas Table 9.13 shows the same results for different exploration strategies (SF and SR) and BestI1 with parallelization and filtering. We set our significance level at 5%, remarking that we did not apply any correction for multiple comparisons, so results may be characterized by a high false positive rate. Notice that from Tables 9.11 the results on the objective gaps are significant when comparing any DET version with any BestI2 version, except for BestI2+F+3PN with DET, whose p-value is equal to 17%. Comparison between BestI2 versions and

Table 9.11: P-values for different algorithm versions vs BestI2

	BestI2	BestI2+F	BestI2+F+3PN	BestI2+F+4PE
BestI2	–	0.02	0.36	0.43
BestI2+F	0.02	–	0.73	0.35
BestI2+F+3PN	0.36	0.73	–	0.99
BestI2+F+4PE	0.43	0.35	0.99	–
DET	0.00	0.02	0.17	0.04
DET+F	0.00	0.00	0.02	0.00
DET+F+4PE	0.00	0.00	0.02	0.00

Table 9.12: P-values for different algorithm versions vs DET

	DET	DET+F	DET+F+4PE
BestI2	0.00	0.00	0.00
BestI2+F	0.02	0.00	0.00
BestI2+F+3PN	0.17	0.02	0.02
BestI2+F+4PE	0.04	0.00	0.00
DET	–	0.09	0.09
DET+F	0.09	–	–
DET+F+4PE	0.09	–	–

between DET versions are not supported by significant p-values, with the exception of BestI2 and BestI2+F that has p-value equal to 2%. From Table 9.13, we can observe that the objective gaps are significant when comparing BestI1+F+4PE to any of the other two algorithm versions with sequential and random neighborhood switch strategy, whereas results do not provide evidence for statistical significance in the comparison between the sequential and random switch strategies.

9.5 Assessment through optimality bounds

In this chapter we run tests to assess the quality of the solution provided by the proposed heuristic according to optimality gaps provided by the Column Generation algorithm described in Chapter 6.
In the following tests, we took into consideration the set of instances TC2 with algorithm BestI2 run on a i5-7400 3.00 GHz machine with 8GB RAM.

Table 9.13: P-values for different exploration strategies

	BestI1+F+4PE	SF+F+4PE	SR+F+4PE
BestI1+F+4PE	–	0.00	0.00
SF+F+4PE	0.00	–	0.81
SR+F+4PE	0.00	0.81	–

Table 9.14: Results of SF and SR vs BestI1

Group	SF+F+4PE (%)	SF+F+4PE (s)	SR+F+4PE (%)	SR+F+4PE (s)
TC2.1	0.0 (-0.4 ; 0.0)	0.7 (0.3 ; 3.8)	0.0 (-0.4 ; 0.0)	0.7 (0.2 ; 3.1)
TC2.2	0.2 (-0.2 ; 1.0)	4.9 (0.3 ; 13.1)	0.4 (-0.2 ; 1.3)	4.3 (0.2 ; 11.1)
TC2.3	-0.3 (-4.9 ; 0.8)	8.8 (5.8 ; 15.0)	0.1 (-5.1 ; 2.5)	8.3 (4.7 ; 12.7)
TC2.4	-0.6 (-3.9 ; 0.9)	11.8 (7.4 ; 16.6)	-0.1 (-3.5 ; 1.8)	11.2 (6.7 ; 15.6)
TC2.5	0.3 (-0.3 ; 1.8)	22.3 (8.6 ; 43.6)	0.6 (-0.1 ; 3.2)	18.5 (6.8 ; 32.4)
TC2	*-0.1 (-4.9 ; 1.8)*	*8.5 (0.3 ; 43.6)*	*0.2 (-5.1 ; 3.2)*	*7.6 (0.2 ; 32.4)*

Table 9.15: Results of CG vs BestI2

Group	CG bound (%)	CG opt (%)	BestI2 opt (%)	CG vs BestI2 (%)
TC2.1	100.0	77.8	66.7	0.8 (0.4 ; 1.1)
TC2.2	77.8	66.7	22.2	1.4 (0.4 ; 3.1)
TC2.3	90.0	60.0	0.0	0.8 (0.1 ; 1.4)
TC2.4	100.0	77.8	11.1	0.8 (0.2 ; 2.4)
TC2.5	66.7	33.3	0.0	0.6 (0.0 ; 0.9)
TC2	*88.4*	*65.1*	*20.9*	*0.9 (0.0 ; 3.1)*

The parameters are the same as the one calibrated for tests in Section 9.3.2, namely $M_1 = 30$, $M_2 = 100$ and T as in Section 9.3.1. We set $M = 10 \cdot \sum_{o \in O} P_o$, where M is the penalty for missed mandatory orders in the set covering model defined in Chapter 6. We recall that, according to the assumptions in Section 6.1, the results in this table do not take preferences on collecting orders in the same route ($I_D(s)$) and minimum number of vehicles at the depot at the end of day ($I_E(s)$) into account, then we deactivated those features in the instances.

Table 9.15 compares the quality of the solutions provided by BestI2 to the bound obtained by the column generation procedure (CG) described in Chapter 6. Results are aggregated per group (first column) and we report per-cent values on: number of instances where the CG converges within one hour and, hence, a bound is available; number of instances where the linear programming relaxation from CG has no fractional variables; number of instances BestI2 solves to proven optimality (when the bound is available); optimality gap between the upper bound from CG and the BestI2 solution in the remaining cases (when the bound is available). Notice that the gap measures the profit loss with respect to a baseline (CG in this case). Figure 9.3 shows the gaps of the CG bounds found against BestI2 on TC2 instances. Each point correspond to an instance of TC2 with an optimality bound obtained through CG. On the ordinates we have the gap between the BestI2 solution of that instance and the related bound, while the green dashed line represents the average gap. We observe that the proposed CG converges in most of the real instances (88.4% on average). Moreover, the solution of the LP relaxation is often integer, thus providing the optimal solution to EPDT (under the assumption above) in the 65.1% of cases (almost 78% of smaller and 33% of larger instances are solved). Based on CG

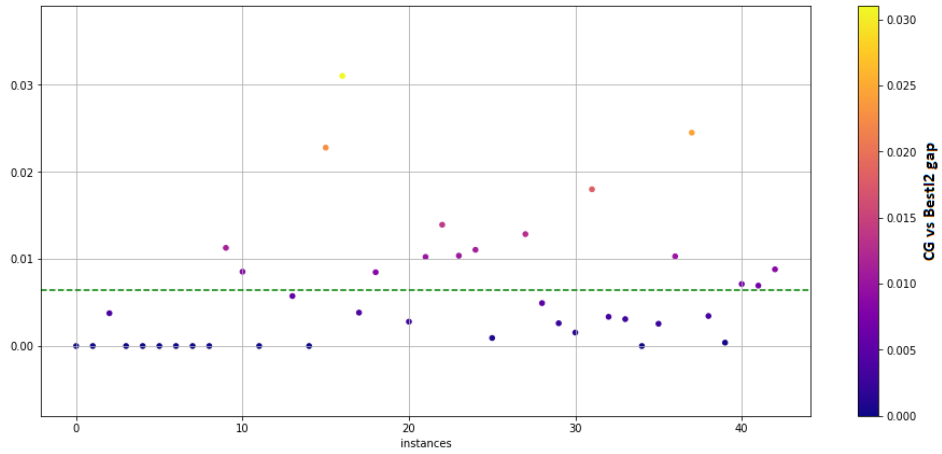

Figure 9.3: Gaps of the CG bounds found against BestI2 on TC2 instances.

bound, we prove the optimality of nine solutions over 43 (21%), in particular for small instances. In the remaining cases the gap, when available, is fairly small: less than 1% on average and at most 3.1% in the worst case. This attests to the ability of the proposed two-level tabu search of finding optimal or nearly optimal solutions for EPDT in real scenarios.

9.6 Tests on dynamic settings

In the dynamic scenario considered in this section, new orders show up during the workday operations. For each new order, a price is negotiated between the customer and the carrier. As we have seen in Section 8.6, the algorithmic engine is used by the Order Portal to supports the carrier in this task with two intervals: the first is a suggested order price range, based on an ordinal classifier trained on past decisions. The second is the interval characterized by the best and worst-case marginal cost as from the insertion of the order itself in the current solution by means of the routing optimizer. With this information, the decision maker can decide either to accept or discard the incumbent order, assisted by the two intervals obtained in real-time. Experiments of this section have been executed on a i5-5200 2.20 GHz machine with 8GB RAM.

9.6.1 Price estimation

We have estimated the ordinal classifier parameters on a dataset of 1604 orders. A score is associated with each prediction for the validation in the learning phase. The score of a prediction is computed as follows: we consider an evaluation dataset and, for each entry, we measure the classifier

Table 9.16: Results on dynamic scenario: Marginal costs, BestI1 vs SF (s)

Group	Marginal costs (s)	BestI1+F+4PE re-opt.(s)	SF+F+4PE re-opt.(s)
TC2.1	0.3 (0.0 ; 0.5)	0.5 (0.2 ; 0.8)	0.3 (0.1 ; 0.5)
TC2.2	5.4 (0.7 ; 8.4)	8.0 (4.4 ; 13.6)	5.0 (2.7 ; 7.7)
TC2.3	7.3 (3.2 ; 12.0)	16.7 (13.9 ; 21.1)	9.1 (7.8 ; 12.8)
TC2.4	11.4 (6.9 ; 15.8)	23.8 (20.8 ; 26.6)	13.9 (13.5 ; 17.1)
TC2.5	11.6 (8.8 ; 19.4)	24.4 (22.5 ; 27.7)	15.2 (14.1 ; 18.1)
TC2	*9.1 (0.0 ; 19.4)*	*20.4 (0.2 ; 27.7)*	*12.4 (0.1 ; 18.1)*

error as the distance (in number of intervals) between the selected interval and the correct one; we compute the maximum error, i.e., the distance of the correct interval from the farthest one; the score is obtained by comparing the norms of the vectors associated with the classifier and to the maximum errors, such that 100% corresponds to no errors and 0% to all maximum errors. The classifier takes negligible time to suggest a price interval and we estimate a score of 75.3%.

9.6.2 Marginal cost estimation

Table 9.16 contains results on dynamic re-optimization for marginal cost computation and insertion of dynamic orders.
Concerning the marginal cost interval, the lower bound is obtained solving EPDT in dynamic setting to insert the new order: this is done by applying the routing algorithm (we considered version BestI1+F+4PE) to re-optimize the routes starting from the current fleet and order status. We have conducted some experiments on TC2 with parameters calibrated as for test in Section 9.4.1, that is, $M_1 = 30$ and $M_2 = 100$, T defined as in Section 9.3.1, $\rho = 0.5$ and $\delta = 20$. For each instance in the benchmark, we consider five cases, each including a new order randomly selected from all the orders in different instances. For the same cases, we estimated the worst case marginal insertion cost by inserting the new order into each of the current vehicle routes, running an intra-route optimization and taking the worst assignment, defined by the route that provides the largest cost increase. The time needed to obtain the lower and upper bounds (average, minimum and maximum) is shown in the second column of Table 9.16. The algorithm takes a few seconds to estimates the best and the worst marginal costs (about 20 seconds in the worst case), which is fairly compatible with negotiation activities required from the Orders Portal module described in Section 8.2.

9.6.3 Marginal cost estimation with 3 orders

A further experiment, whose results appear in the last two columns of Table 9.16, measures the time needed to solve the dynamic EPDT to suggest the

assignment of more than one new simultaneous orders. The columns show, in the order, the running times of algorithm BestI1+F+4PE and SF+F+4PE, run with the same parameters as calibrated above: the former algorithm has been executed with parameters $M_1 = 30$ and $M_2 = 100$, T defined as in Section 9.3.1, $\rho = 0.5$ and $\delta = 20$; the latter algorithm uses $M_1 = 60$, $I_n = 15$ and $I_b = 5$. For each instance in the benchmark, we have chosen at random three new orders to be inserted and repeated the experiment five times: even in this case, and in particular for the SF exploration strategy, running time seems to be largely compatible with operational settings.

9.7 Comparison with Literature Benchmarks

In order to verify how the proposed algorithm solves problems of broad interest, in this section we execute some tests on literature benchmarks.
We consider the Li-and-Lim's benchmark for PDPTW [64], that consists of single pickup and single delivery orders, hard time windows and vehicle capacity, and the objective function considers the number of vehicles and the total distance in lexicographic order.
The attributes of the PDPTW benchmarks above represent a subset of attributes in EPDT. In order to represent the objective function of Li-and-Lim's benchmarks in the EPDTmodel, we set vehicle deployment fixed costs to a big-M constant. Moreover, we set revenues to 0, so that, in fact, we minimize the total travel distance. PDPTW routes are closed and we model them with constraints on the route ending position. Moreover, since we have only today tasks, we simplified route evaluation. We have run the BestI2+F+4PE version on a i5-7400 3.00 GHz machine with 8GB RAM with the usual parameters $M_1 = 30$, $M_2 = 100$, T defined as in Section 9.3.1, $\rho = 0.5$ and $\delta = 20$.
The Li-and-Lim's benchmark includes 56 instances adapted from the Solomon's benchmark [95]. They define 100 orders each, and are classified in six classes, as listed in the first columun of Table 9.17, depending on: spatial distribution (orders are clustered in LC, uniformly distributed in LR, mixed in LRC); scheduling horizon (shorter in '1' and longer in '2' instances, with suitable smaller and larger capacities). Table 9.17 compares the results obtained by our heuristic to the best solution, available from [93] and provided by different works [11, 52, 64, 82, 85]. For each class, we report average and maximum number of additional vehicles needed by our procedure, the related per-cent value, the per-cent additional distance, and the running time in seconds (average, minimum and maximum). The number of additional vehicles is between zero and two (0.5 on average) and the routes are longer by 4.1% on average. Notice that if we restrict to '1'-instances, the average increase in the route length is only 1.9%. In fact, instances of type '1' are more compliant with usual EPDTsettings, where the time horizon is short

Table 9.17: Results on PDPTW instances (BestI2).

Group	Δvehicle	Δvehicle%	Δdistance%	time (s)
LC1	0.2 (1.0)	2.2 (10.0)	-0.3 (-17.3 ; 19.2)	1000 (652; 1786)
LC2	0.1 (1.0)	3.1 (25.0)	4.8 (0.0 ; 27.3)	4332 (4262; 4441)
LR1	0.7 (2.0)	5.7 (18.2)	3.0 (**-0.3** ; 7.5)	1255 (274; 1983)
LR2	0.6 (1.0)	18.2 (33.3)	7.6 (0.7 ; 17.6)	4269 (4237; 4341)
LCR1	0.6 (2.0)	4.7 (13.3)	2.9 (0.9 ; 4.9)	4425 (4248; 4919)
LCR2	0.6 (1.0)	15.6 (25.0)	6.7 (3.0 ; 11.7)	558 (326; 801)
All	*0.5 (2.0)*	*8.5 (33.3)*	*4.1 (-17.3 ; 27.3)*	2607 (274; 4919)

with respect to the distance between tasks. Running times show how the efficiency of our heuristic is strongly related to the number of orders expected in a route, due to the computational complexity of the second-level heuristic for solution evaluation, which suffers from routes of consistently large sizes, as is the case for '2'-instances. Therefore, the time of execution of our algorithm differs on average of 1 order of magnitude with respect to other ad-hoc approaches in literature for PDPTW (see, e.g., [64]). Nonetheless, we obtain one new best solution for instance LR106, which slightly improves the one of [64]: we use the same number of vehicles and shorten the total distance by 0.3%.

Chapter 10

Conclusions

The Vehicle Routing Problem is one of the most studied problems in Operations Research. Several extensions of the classical VRP definition have been treated in literature, mostly rising from real-world applications. In fact, decision makers from transportation company have to deal with VRPs where a large number of attributes are considered simultaneously. These types of problems are known as Multi-Attribute Vehicle Routing Problems (MAVRP).
We focused our study on a particular MAVRP inspired by a small freight transportation company named Trans-Cel S.n.C. (Padova, Italy). This problem is new to the Operations Research literature and we call it Express Pickup and Delivery in freight Trucking problem (EPDT). It includes some peculiar characteristics, as multi-pickup and multi-delivery requests, daily planning horizon where orders may contain tasks taking place either in the same day or in two consecutive days, and other specific constraints and quality criteria, which make the problem worth of research.
The operational context where the problem must be solved asks for a fast optimization tool in order to obtain a high quality initial route plan in a static context, as well as rapidly answer to dynamic events, as for instance a just-in-time transportation request, vehicle breakdowns, or road network congestions.
Due to the high complexity of the problem and operational scenario requirements, we propose a heuristic algorithm to assist the decision maker in the planning phase. The devised heuristic is able to support the planning process both in static scenario and, through re-optimization, in dynamic scenario. The algorithm combines a Tabu Search algorithm with a Variable Neighborhood Descent (Tabu VND) and is structured on two levels: at the first level, the Tabu VND explores the order-to-vehicle assignment space. At the second level, triggered at any order assignment or removal, a local search algorithm is run on the task sequence of the route. Among our main contributions, we introduced specific characteristics in this scheme

to enhance its efficiency: we designed a granular exploration based on the definition of a proximity graph that includes a concept of distance between pairs of orders; we set up different types of parallel implementations for the neighborhood exploration; we devised a fast evaluation of the solutions with destroy-and-repair fixing phase; we implemented two exploration strategies based on sequential and cyclic neighborhood switch.

Given the availability of historical data on customer orders, that are more and more present in transportation companies databases, we have examined two new approaches to the dynamic and stochastic VRP, both based on data-driven techniques. Our main goal is the definition of components computed a-priori that aggregate preprocessed statistical information from the historical data. The components are then embedded in a solution method for the deterministic version of the problem, providing the method itself with the ability of solving the dynamic and stochastic variant, at small or no impact on the algorithm design. The first procedure is based on determining clusters of orders whose centroids provide strategical space-time positions that guide the algorithm toward order assignments and route sequencing more suitable to accommodate near-future requests. The second approach introduces a concept of accessibility on each node of the network, representing a measure of the opportunity to satisfy future orders across the graph. We then add a term in the objective function representing the accessibility, in space and time, that is collected by each route of the solution, in order to obtain more stable routes with respect to future demand.

Trans-Cel has carried out the development of a software platform, named Chainment, to support the operational manager in small freight transportation context. Chainment consists of multiple modules that communicate through a data-sharing system. The main provided functionalities support the operation manager of Trans-Cel by means of optimization and artificial intelligence tools. In particular, Chainment relies on an algorithmic engine that includes prediction models and a routing optimizer based on the heuristic algorithm we devised for EPDT. The former are triggered during the revenue estimation of just-in-time requested orders and service time estimation, whereas the latter comes into play for static routing, dynamic re-optimization and marginal cost evaluation of incoming orders.

Computational results show that the routing algorithm reaches effectiveness and efficiency that are compatible with the operations managers' requirements. In fact, the solution of real instances is obtained in 6.1 seconds on average, about 45 seconds in the worst case. In particular, tests on real instances assess that the solution quality is preserved by filtering and parallel explorations, as the average objective value loss is 0.0%, 1.0% in the worst case. The dynamic setting is characterized by demand coming at a relatively low dynamism. Experiments on dynamic scenarios show that re-optimization is suitable to handle just-in-time requests: in particular we tested the dynamic insertion of three orders, that was performed by the al-

gorithm within an average time of 20.4 seconds, 27.7 in the worst case. The quality of the solution method has been assessed through the comparison with an optimality bound provided by a Column Generation algorithm that we implemented. The results show an average gap between the bound and the algorithm of 0.9%, 3.1% in the worst case. In particular, we obtained a bound for 88.4% of the instances, and in 20.9% of the tests the heuristic algorithm was able to find the optimal solution. The Pricing problem is solved through a label correcting algorithm for an Elementary Shortest Path Problem with Resource Constraints (ESPPRC), where label definition, extension function and dominance rule have been suitably adapted to the EPDT context, which represent a further contribution of the thesis. In particular, we introduced specific resources to handle multi-pickup and multi-delivery orders. We have also run tests on a PDPTW set of benchmark instances as particular cases of EPDT, proving the algorithm to be competitive in terms of effectiveness with respect to the state-of-the-art, providing also a new best solution.

Future works may be devoted to the realization of a Branch-and-Price algorithm, possibly starting from the proposed Column Generation algorithm. Moreover, new GPU designs may be studied for the parallel generation of neighbor solutions during the exploration phase. Furthermore, the proposed approach to dynamic and stochastic version of EPDT by data-driven techniques is at a preliminary stage, so more efforts may be carried out in order to better analyze and possibly enhance the proposed methods performances.

Acknowledgments

I will start with a huge thanks to my supervisor, prof. Luigi De Giovanni, whose advise through all these years have been precious, making me constantly grow my passion for Operations Research. He has been an example in both technical matters and attitude toward science and research. Also, I want to thank all the PhD students and researchers that I met during these three years, in the Mathematics department of Padova, at SINTEF, where I spent a part of my PhD, and at OR conferences and workshops, in particular to the OR group AIRO and AIROYoung. Moreover, I warmly thank all the people from Trans-Cel, the drivers, the office operators and the development team. I learned a lot from all of them, especially how to build the connection between science and industry, from both sides. In particular I thank my tutor Filippo Sottovia, for his determination, energy and for believing in an outstanding project, that I am proud to be part of. Finally, I want to thank my family and friends, for the good times spent together and the support when things go wrong, especially my mother and my father, who has been looking after me from above in years.

Thanks to every single one of you, enjoy science, enjoy Maths, enjoy Operations Research!

References

[1] T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[2] Agile. https://agilemanifesto.org/.

[3] J. Alegre, M. Laguna, and J. Pacheco. Optimizing the periodic pick-up of raw materials for a manufacturer of auto parts. *European Journal of Operational Research*, 179(3):736 – 746, 2007.

[4] A. Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[5] C. Archetti, A. Hertz, and M.G. Speranza. Metaheuristics for the team orienteering problem. *J. Heuristics*, 13:49–76, 02 2007.

[6] C. Archetti and M.G. Speranza. A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, 2(4), 2016.

[7] P. Arias, J. Caceres-Cruz, D. Guimarans, and A. A. Juan. Rich vehicle routing problem: Survey. *ACM Computing Survey*, (2):229–268, 2014.

[8] R. Baldacci, E. Bartolini, A. Mingozzi, and R. Roberti. An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*, 7(3):229–268, 2010.

[9] R. Baldacci and A. Mingozzi. A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming*, 120(2):347–380, 2008.

[10] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, 46(3):316–329, 1998.

[11] R. Bent and P. Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33:875–893, 2006.

[12] R. Bent and P. Van Hentenryck. Online Stochastic Optimization Without Distributions. pages 171–180, 01 2005.

[13] G. Berbeglia, J.F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8 – 15, 2010.

[14] A. Bettinelli, A. Ceselli, and G. Righini. A branch-and-price algorithm for the multi-depot heterogeneous-fleet pickup and delivery problem with soft time windows. *Mathematical Programming Computation*, 6:172–197, 2014.

[15] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[16] M. A. Boschetti, V. Maniezzo, M. Roffilli, and A. Bolufé Röhler. Matheuristics: Optimization, simulation and control. In M. J. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Sampels, and A. Schaerf, editors, *Hybrid Metaheuristics*, pages 171–177, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[17] S. Bradley, A. Hax, and T. Magnanti. *Applied mathematical programming*. 1977.

[18] O. Bräysy, M. Gendreau, G. Hasle, A. Lokketangen, and J.Y. Potvin. Metaheuristics for the vehicle routing problem and its extensions: a categorized bibliography. In B. Golden, S. Ranghavan, and E. Wasil, editors, *The vehicle routing problem: latest and new challanges*, pages 143–169. Springer, New York, USA, 2008.

[19] A. Brodtkorb, T. Hagen, C. Schulz, and G. Hasle. GPU computing in discrete optimization. Part I: Introduction to the GPU. *EURO Journal on Transportation and Logistics*, 2, 2013.

[20] A. Ceselli, G. Righini, and M. Salani. A Column Generation Algorithm for a Rich Vehicle-Routing Problem. *Transportation Science*, 43(1):56–69, 2009.

[21] CMake. https://cmake.org/.

[22] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer Programming*. 2014.

[23] J.F. Cordeau and M. Maischberger. A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033 – 2050, 2012.

[24] C. Cortes and V. Vapnik. Support-Vector Networks. In *Machine Learning*, pages 273–297, 1995.

[25] T. G. Crainic, M. Gendreau, P. Hansen, and N. Mladenović. Cooperative Parallel Variable Neighborhood Search for the p-Median. *Journal of Heuristics*, 10(3):293–314, May 2004.

[26] T.G. Crainic and M. Gendreau. Cooperative Parallel Tabu Search for Capacitated Network Design. *J. Heuristics*, 8:601–627, 11 2002.

[27] DBeaver. https://dbeaver.io/.

[28] L. De Giovanni, N. Gastaldon, I. Lauriola, and F. Sottovia. A Heuristic for Multi-attribute Vehicle Routing Problems in Express Freight Transportation. In A. Sforza and C. Sterle, editors, *Optimization and Decision Science: Methodologies and Applications ODS 2017. Springer Proceedings in Mathematics & Statistics*, volume 217, pages 161–169, Cham, 2017. Springer International Publishing.

[29] L. De Giovanni, N. Gastaldon, M. Losego, and F. Sottovia. Algorithms for a Vehicle Routing Tool Supporting Express Freight Delivery in Small Trucking Companies. *Transportation Research Procedia*, (30):197–206, 2018.

[30] L. De Giovanni, N. Gastaldon, and F. Sottovia. A two-level local search heuristic for pickup and delivery problems in express freight trucking. *Networks*, 2019. 1–18. https://doi.org/10.1002/net.21917.

[31] P. Dejax, D. Feillet, M. Gendrau, and C. Gueguen. An exact algorithm for the Elementary Shortest Path Problem with Resource Constraints: application to some vehicle routing problems. *Networks*, (44):216–229, 2004.

[32] G. Desaulniers, J. Desrosiers, I. loachim, M. M. Solomon, F. Soumis, and D. Villeneuve. *A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems*, pages 57–93. Springer US, Boston, MA, 1998.

[33] M. Desrochers, J. Desrosiers, and M. Solomon. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Operations Research*, 40(2):342–354, 1992.

[34] J. Desrosiers and M.E. Lübbecke. Selected Topics in Column Generation. *Operations Research*, pages 1007–1023, 2005.

[35] DoxyGen. http://www.doxygen.nl/.

[36] Driver. http://www.cstv.it/prodotti/driver/.

[37] D. Dueck. Affinity propagation: clustering data by passing messages, 2009.

[38] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[39] S. Fekete, J. Mitchell, and K. Beurer. On the Continuous Fermat-Weber Problem. *Operations Research*, 53, 11 2003.

[40] F. Ferrucci, B. Stefan, and M. Gendreau. A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods. *European Journal of Operational Research*, 225:130–141, 2013.

[41] E. Frank and M. Hall. A Simple Approach to Ordinal Classification. volume 2167, pages 145–156, 08 2001.

[42] M. Gendreau, F. Guertin, J.Y. Potvin, and R. Séguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14(3):157 – 174, 2006.

[43] G. Ghiani, E. Manni, A. Quaranta, and C. Triki. Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):96 – 106, 2009.

[44] Git. https://git-scm.com/.

[45] GitKraken. https://www.gitkraken.com/.

[46] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8:156 – 166, 01 1977.

[47] F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

[48] F. Glover, M. Laguna, and R. Marti. *Tabu Search*, volume 16. 07 2008.

[49] Google Maps API. https://developers.google.com/maps/.

[50] P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable neighbourhood search: methods and applications. *4OR*, 6(4):319–360, Dec 2008.

[51] W.G. Hansen. How Accessibility Shapes Land Use. *Journal of the American Institute of Planners*, 25(2):73–76, 1959.

[52] G. Hasle and O. Kloster. Industrial Vehicle Routing. In G. Hasle, K. A. Lie, and E. Quak, editors, *Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF*, pages 397–435. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[53] What is Industry 4.0—the Industrial Internet of Things (IIoT). https://www.epicor.com/en-us/resource-center/articles/what-is-industry-4-0/.

[54] S. Irnich and G. Desaulniers. Shortest Path Problems with Resource Constraints. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 33–65. Springer US, Boston, MA, 2005.

[55] S. Irnich and G. Desaulniers. *Shortest Path Problems with Resource Constraints*, pages 33–65. 03 2006.

[56] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001. [Online; accessed].

[57] JuPyter. https://jupyter.org/.

[58] M. Khemakhem, R. Lahyani, and F. Semet. Rich vehicle routing problems: from a taxonomy to a definition. *European Journal of Operational Research*, 241(1):1–14, 2015.

[59] Y. Kim, H. Mahmassani, and P. Jaillet. Dynamic Truckload Routing, Scheduling, and Load Acceptance for Large Fleet Operation with Priority Demands. *Transportation Research Record Journal of the Transportation Research Board*, 1882, 01 2004.

[60] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by Simulated Annealing. *Science (New York, N.Y.)*, 220:671–80, 06 1983.

[61] B. Korte and J. Vygen. *Combinatorial optimization*. Springer, 2012.

[62] J. R. Koza and R. Poli. *Genetic Programming*, pages 127–164. Springer US, Boston, MA, 2005.

[63] G. Laporte, S. Ropke, and T. Vidal. Heuristics for the vehicle routing problem. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications. MOS-SIAM Series on Optimization*, pages 87–116. 2014.

[64] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. In *Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001*, pages 160–167, 2001.

[65] K. Lund, O.B.G. Madsen, and J.M. Rygaard. Vehicle Routing Problems with Varying Degrees of Dynamism. 01 1996.

[66] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[67] O.C. Martin, S. W. Otto, and E. W. Felten. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5, 1991.

[68] W McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[69] S. Mitrović-Minić, R. Krishnamurti, and G. Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38:669–685, 09 2004.

[70] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100, 1997.

[71] Microsoft Visual Studio. https://visualstudio.microsoft.com/.

[72] MySQL. https://www.mysql.com/.

[73] MySQL Workbench. https://www.mysql.com/it/products/workbench/.

[74] Y. Nagata and O. Bräysy. Efficient Local Search Limitation Strategies for Vehicle Routing Problems. pages 48–60, 03 2008.

[75] T. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[76] R.R. Picard and R.D. Cook. Cross-Validation of Regression Models. *Journal of the American Statistical Association*, 79(387):575–583, 1984.

[77] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403 – 2435, 2007.

[78] C. Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. Computer & Operations Research 31(12), 1985-2002. *Computers & Operations Research*, 31:1985–2002, 10 2004.

[79] H.N. Psaraftis, M. Wen, and C.A. Kontovas. Dynamic vehicle routing problems: Three decades and counting. *Networks*, 67(1):3–31, 2016.

[80] V. Pureza and G. Laporte. Waiting and Buffering Strategies for the Dynamic Pickup and Delivery Problem with Time Windows. *INFOR: Information Systems and Operational Research*, 46(3):165–175, 2008.

[81] Python. https://www.python.org/.

[82] Delmia Quintiq. https://www.quintiq.es/optimization/vrptw-world-records.html.

[83] D. Rey and M. Neuhäuser. *Wilcoxon-Signed-Rank Test*, pages 1658–1659. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[84] U. Ritzinger, J. Puchinger, and R.F. Hartl. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research*, 54(1):215–231, 2016.

[85] S. Ropke. *Heuristic and exact algorithms for vehicle routing problems*. PhD thesis, Technical University of Denmark, 2006.

[86] S. Ropke and J. F. Cordeau. Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 43(3):267–286, 2009.

[87] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004.

[88] C. Schulz. Efficient local search on the GPU—Investigations on the vehicle routing problem. *Journal of Parallel and Distributed Computing*, 73:14–31, 2013.

[89] C. Schulz, G. Hasle, A. Brodtkorb, and T. Hagen. GPU computing in discrete optimization. Part II: Survey focused on routing problems. *EURO Journal on Transportation and Logistics*, 2, 2013.

[90] S. Seabold and J. Perktold. Statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[91] P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher and J.F. Puget, editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[92] J.A. Sicilia, C. Quemada, B. Royo, and D. Escuín. An optimization algorithm for solving the rich vehicle routing problem based on variable neighborhood search and tabu search metaheuristics. *Journal of Computational and Applied Mathematics*, 291:468 – 477, 2016. Mathematical Modeling and Computational Methods.

[93] Sintef (Transportation Optimization Portal). https://www.sintef.no/projectweb/top/pdptw.

[94] SciKit-Learn. https://scikit-learn.org.

[95] M.M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constrains. *Operations Research*, 35:254–264, 1987.

[96] SourceTree. https://www.sourcetreeapp.com/.

[97] P. H. Swain and H. Hauska. The decision tree classifier: Design and potential. *IEEE Transactions on Geoscience Electronics*, 15(3):142–147, July 1977.

[98] R. Tadei and F. Della Croce. *Ricerca Operativa e Ottimizzazione, Seconda Edizione*. Società Editrice Esculapio, 2002.

[99] Taiga. https://taiga.io/.

[100] É. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.Y. Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31:170–186, 05 1997.

[101] E.G. Talbi. *Metaheuristics, from design to implementation*. 2009.

[102] Tom Tom Telematics. https://www.telematics.tomtom.com/.

[103] TomTom Developers. https://developer.tomtom.com/.

[104] P. Toth and D. Vigo. An overview of vehicle routing problems. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*. SIAM - Society for Industrial and Applied Mathematics, Philadelphia, USA, 2002.

[105] P. Toth and D. Vigo. The Granular Tabu Search and Its Application to the Vehicle-Routing Problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003.

[106] P. Toth and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. SIAM - Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2014.

[107] Transporeon. https://www.transporeon.com.

[108] Trello. https://trello.com/.

[109] Vagrant. https://www.vagrantup.com/.

[110] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei. A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems. *Operations Research*, 60:611–624, 06 2012.

[111] 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*, 231(1):1–21, 2013.

[112] T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489, 2013.

[113] T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins. A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673, 2014.

[114] Visual Studio Code. https://code.visualstudio.com/.

[115] Workwave. https://www.workwave.com/route-manager/.

[116] G. Zhang, K. Smilowitz, and A. Erera. Dynamic planning for urban drayage operations. *Transportation Research Part E: Logistics and Transportation Review*, 47(5):764–777, September 2011.