Vehicle routing optimization in bicycle sharing systems

JUAN DAVID PALACIO DOMÍNGUEZ

Tesis

Asesor
Juan Carlos Rivera Agudelo

UNIVERSIDAD EAFIT
ESCUELA DE CIENCIAS
DOCTORADO EN INGENIERÍA MATEMÁTICA
MEDELLÍN
2022

Vehicle routing optimization in bicycle sharing systems

Juan David Palacio Domínguez

Master of Science in Engineering

A thesis submitted for the degree of Doctor of Philosophy at Universidad EAFIT in 2022

School of Applied Sciences and Engineering

A mis padres, por su infinito amor y paciencia durante estos años.
A mi abuela, su fortaleza nunca dejará de ser un motivo de inspiración.

Acknowledgments

Firstly, I thank my parents, Ana and Jesús, for their love and company. Their unconditional support makes this process easier.

I thank my advisor Juan Carlos Rivera, for giving me the opportunity to develop this project and for share his creativity and his virtually infinite number of ideas in every step of the way. I really enjoy to work with Juan learning a lot from the very beginning.

I also would like to thank Juan G. Villegas for his valuable advice throughout this process, my master and undergraduate studies. Thanks for being a friend and role model from early beginnings.

I thank my friends Valentina Gutiérrez and Carolina Castañeda for their unconditional support and for being there even in the hard days. This process cannot be done without their company, advise, laughs and so many special moments. One of the most important learning during these years is about pure and real friendship.

I also thank Paola, Leandro, Jhonatan and Santiago at Modelado Matemático research group. My stay as student at EAFIT was a great experience thanks to their support. Thanks to Paula Escudero for giving me the opportunity to work with her and for help to activate my mind and body sharing many high-speed rides to EAFIT.

I thank Felipe Isaza for give me the opportunity to start a life as full professor at Universidad de Medellín. I would like to thank Ximena Gaviria, Natalia Gómez, Luisa Villa, María Isabel Mejía and Alina Avendaño for their advice, support (and even predictions) in the last steps of this process.

Thanks to a very special team: Ana María, Isabela, Catalina, Jhon Jander, Sebastian, Juan José and Daniel; lunches, coffees, laughs and classes were fuel to finish this thesis.

I would like to thank Claudia Velasquez and Ricardo Bernal for show me how to live with well-being when days turned gray.

Thanks to Professors Juan José Salazar, José Manuel Belenguer, Hasan Murat Afsar, Pablo Maya, William Guerrero and Mario César Vélez for accepting the invitation to be part of the jury of my thesis proposal evaluation and qualifying exams. The projects I developed and reviews on thesis proposal help me to complete successfully this process.

Finally, I would like to thank Lucia Quintero for taking care of many key administrative matters during the PhD studies. Thanks also to the Mathematical Engineering PhD program committee at EAFIT, specially to Andrés Sicard for reading and giving feedback on the thesis proposal.

Abstract

Vehicle routing problems (VRPs) are a set of combinatorial optimization problems with large number of applications mainly in logistics context for services delivery and good distributions. This thesis studies a set of VRPs that arises in a particular sharing mobility context: bicycle sharing systems (BSSs). BSSs operate with the aim to alleviate traffic congestion and air pollution. By the use of bikes for short trips in an urban area, the use of automobiles is reduced as well as potential health issues caused by CO_2 emissions. A successful BSS logistic operation involves decision making processes that may be supported via Operation Research (OR) techniques. Precisely, this document study three problems that may arise in BSSs logistic context: The the one-commodity pickup and delivery traveling salesman problem (1–PDTSP), the one-commodity pickup and delivery vehicle routing problem with length constraints (1–PDVRPLC) and the two-echelon bicycle repositioning problem with split demand (2E–BRPSD)

The first part addresses the 1–PDTSP. In the 1–PDTSP a single capacitated vehicle picks up and delivers commodity units in a set of locations with a predefined demand. In BSS contexts, this problem is tackled as the static bike repositioning problem (BRP). For this problem, this thesis describes a mixed-integer linear programming model (MILP) and a metaheuristic algorithm based on evolutionary local search (LS) and variable neighborhood descent (VND). Similarly, this document also studies the SD1PDTSP which is a 1–PDTSP version where the split demand is allowed. For the SD1PDTSP an MILP is also described.

The second part is devoted to the 1–PDVRPLC. The 1–PDVRPLC is a generalization of 1–PDTSP where multiple vehicles are available. To solve the 1–PDVRPLC, an MILP is described as well as two matheuristic algorithms based on large neighborhood search (LNS) procedures. The first matheuristic strategy is a multi-start iterated LNS, where destroy and repair operators are replaced by a single MILP. The second matheuristic is an adaptive LNS, where several destroy methods are available and an MILP repairs solutions. Additionally, an exact algorithm that efficiently enumerates possible repaired solutions is described. The algorithm replaces MILP in adaptive LNS strategy and some dominance rules are presented to speedup the enumeration process.

Finally, the third part defines 2E–BRPSD. In the 2E–BRP, the repositioning operation is completed within a two-echelon configuration. While a set of vehicles visits stations (second echelon), a single vehicle supports that operation by picking up or delivering bikes in stations selected as satellite depots. For the 2E–BRPSD, an MILP is presented as well as four metaheuristic algorithms.

Keywords: Operation research, Combinatorial optimization, Metaheuristics, Matheuristics, Vehicle routing problem Mixed-integer linear programming, Bicycle sharing systems, Bicycle repositioning problem.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

Abbreviations	
1–PDTSP	One-commodity pickup and delivery traveling salesman problem
1–PDTSP–RD	One-commodity pickup and delivery vehicle routing problem with restricted depot
1–PDVRP	One-commodity pickup and delivery vehicle routing problem
1–PDVRPLC	One-commodity pickup and delivery vehicle routing problem with length constraints
2E–1PDP	Single-vehicle two-echelon one-commodity pickup and delivery problem
2E–LRP	Two-echelon location routing problem
2E–MTVRP-SS	Two-echelon multiple-trip vehicle routing problem with satellite synchronization
2E–BRP	Two-echelon bicycle repositioning problem
2E–BRPSD	Two-echelon bicycle repositioning problem with split delivery
2E–VRP	Two-echelon vehicle routing problem
ABC	Artificial bee colony
ALNS	Adaptive large neighborhood search
B&C	Branch and cut
BA	Bat algorithm
BRP	Bicycle repositioning problem
BSS	Bicycle sharing system
CVRP	Capacitated vehicle routing problem
D&R	Destroy-and-repair
DARP	Dial-a-ride problem
DBMEA	Discrete bacterial memetic evolutionary algorithm
DSS	Decision support system
E2E–VRP	Electric two-echelon vehicle routing problem
2E–EVRP–BSS	Electric two-echelon vehicle routing problem with battery swapping stations
ELS	Evolutionary local search

GA	Genetic algorithm
GRASP	Greedy randomized adaptive search procedure
GTSP	Generalized traveling salesman problem
HHCP	Home health care problem
ILS	Iterative local search
ILP	Integer linear program
LNS	Large neighborhood search
LP	Linear program
m–PDTSP	Multi-commodity pickup and delivery traveling salesman problem
MA	Memetic algorithm
MILP	Mixed-integer linear program
MS–ELS	Multi-start evolutionary local search
MS–ILNS	Multi-start iterated large neighborhood search
MS–ILS	Multi-start iterated local search
NN	Neural network
PRTTA	Population algorithm based on randomized tabu threesholding
SA	Simulated annealing
SD1PDTSP	Split delivery one-commodity pickup and delivery traveling salesman problem
SDTSP	Split delivery traveling salesman problem
SPP	Set partitioning problem
TS	Tabu search
TSP	Traveling salesman problem
TTRP	Truck and trailer routing problem
VND	Variable neighborhood descent
VNS	Variable neighborhood search
VRP	Vehicle routing problem
VSS	Variable space search

Chapter 1

General Introduction

The International Federation of Operational Research Societies (IFORS) describes OR as "*the development and the application of a wide range of problem-solving methods and techniques applied in the pursuit of improved decision-making and efficiency, such as mathematical optimization, simulation, queuing theory and other stochastic models. The OR methods and techniques involve the construction of mathematical models that aim at describing a problem. Because of the computational and statistical nature of most of the techniques, OR also has strong ties to computer science and analytics*" (IFORS, 2022). OR deals with decision problems by formulating and analyzing mathematical, simulation and statistical models of complex engineering and management problems with the aim to provide insights about possible solutions.

Precisely, to design and analyze OR models, mathematics, economics and computer science fields works jointly to develop, for instance, optimization models and algorithms able to find the best possible solution to quantitative problems. In that sense, this thesis is devoted to the solution of a family of vehicle routing problems in a particular nonprofit context of sustainable transportation: bicycle sharing systems logistics. The document provides problem definitions, mathematical formulations and solution strategies for the studied problems through operations research techniques.

Routing decisions in bicycle sharing system contexts and other nonprofit or public applications are not typically made through classical capacitated vehicle routing formulations and algorithms. Apart from total costs minimization, a key factor in logistics for decision makers in nonprofit and public operations, is to reduce the user *disatisfaction*. Then, features as pickups and deliveries, split demands or even synchronization issues must be considered to ensure an efficient operation of the studied system.

This chapter presents an overview of nonprofit and public operation concept and some of its applications where vehicle routing problems arise. This chapter also describes BSSs logistics as a family of combinatorial optimization problems showing the relevance of transportation and routing decisions in these systems as the main context of this document.

1.1 Nonprofit and public operations: definition and context

The commercial and nonprofit sector organizations are different from each other in several aspects. Objectives, conditions, *providers*, *users* and even the type of services offered may be different if the organization profile is profit or nonprofit. These differences among the sectors have been studied in several disciplines and OR is not the exception (Balcik et al., 2010).

Precisely, this document follows the ideas in Balcik et al. (2010) for public services and their providers. The authors state public services as *"services provided to the society by public and/or nonprofit sector organizations"*. Although many public services are provided by government organizations, there exist nonprofit agencies offering some of these public services. Services provided by government agencies include healthcare and public health, education, recreational facilities, water and energy supply and telecommunications. Moreover, transportation also arises in several contexts as public service: emergencies, disaster relief services, special care for elderly people and nursing (home healthcare), public transportation systems.

Organizations providing public services and the commercial sector have substantial differences in goals and consequently, metrics for operations performance in both sectors must be also different. In OR contexts, for commercial services, profits and operational costs are commonly used to measure *efficiency* as a metric for companies performance. *Effectiveness* also emerges as a second metric for private and commercial sector as the level to which these companies reach their goals.

Aside profits, costs or resource utilization levels to measure efficiency and effectiveness in commercial companies, for public sector and nonprofit organizations there exists a different set of metrics. *Equity* and *fairness* criteria arise as measures for their operation performance. While private organizations work for a desired efficiency, in nonprofit contexts, additional questions emerge: is that efficiency gained in a fair way? does it provide a similar benefit for all the population involved or affected with decisions? In other words, are those who are affected by the decision treated fairly or equitably? (Marsh and Schilling, 1994). Balcik et al. (2010) agree with Marsh and Schilling (1994) and state that there are three main questions when equity and fairness are involved in making decisions processes for public or nonprofit sectors: fairness based on what?, who should benefit?, and how is fairness measured?

From a broader perspective, Pollock and Maltz (1994) elaborate a discussion when defining the term *public sector* in an OR context. The authors propose four possible criteria to determine whether decisions are made within a public context, as follows:

- The public (or its representatives) make decisions.
- A public servant directs the organization within which the decision is made.
- The general public pays for the analysis and/or bears the major effects of the decision.
- The ultimate measure of the outcome of a decision or policy is non-monetary.
- Analyses, or other methods used to inform decisions, are subject to public scrutiny.

Based on these criteria, the reader may notice that the research detailed in this document fits in a set of decision making processes from the public sector definition for OR contexts. As next sections in this chapters detail, BBSs are defined as a public service and typically operate under public policies and invariably, the general public has the major effects of decisions made for the BSSs operation.

1.2 Vehicle routing and transportation in public and nonprofit contexts

As mentioned before in Section 1.1 there exist many services provided by public and nonprofit organizations: emergencies, postal services, libraries, health, disaster relief, among others. Although decision making within this contexts are related to several OR and optimization problems as location, inventory, scheduling (Castañeda and Villegas, 2017; Gutiérrez and Vidal, 2013; Ni et al., 2018; Rottkemper et al., 2011), this document focuses on transportation and vehicle routing problems. Before introducing BSSs routing problems, this section is devoted to describe briefly some of the vehicle routing problems for most common public services or services provided by nonprofit organizations. From an extensive list of applications and contexts for transportation problems, a short description of health care, disaster relief, food distribution (food banks and *meals on wheels*) is presented.

Health care (HC) optimization problems have received attention recently since average longevity and population size are increasing. Therefore, many countries are concerned about how to allocate budget and ensure resources availability to meet healthcare needs (Rais and Vianaa, 2011). Moreover,

a different motivation relies on the fact that modern technology allows to offer more decentralized services to population with the aim to increase coverage rates. From a tactical perspective, many vehicle routing problems can be addressed in the HC context. In particular, home health care (HHC) delivery services consider a set of patients previously prescribed by medical staff with treatments that must take place at patient's homes. This kind of service is suitable when, based on patient conditions, it is more comfortable and secure to give care at home, or the beds and rooms are scarce in hospitals. Additionally, apart from security and comfort, these services are less costly for hospitals and medical service centers and allow to reduce recovery periods and improve life quality of patients (Gutiérrez and Vidal, 2013). Classical decisions in HHC contexts include, at least, to determine a professional assignment, a frequency of visits for each patient and, a service time and time windows for each visit. In this case, the HHC provider must design routes for each medical professional in order to satisfy frequencies and the prescribed periods to serve patients. Even though staff and routing costs represent key values for HHC providers, quality in service should be also considered as a quality solution measure. From the patient point of view, quality in service includes minimum waiting times, consistency in terms of the time the service is provided, the prescribed times to be visited and, the number of different medical professionals that provide the services (Kovacs et al., 2014).

Disaster relief problems cover four different phases (Altay and Green III, 2006): mitigation, preparedness, response and recovery. First two concern the pre-crisis. Mitigation is the application of measures that either prevent the onset of disasters or reduce their impacts, while preparedness activities train the community to react when a disaster occurs. When the crisis occurs, the post-crisis phases take place. Response is the deployment of resources and emergency procedures to preserve life, property, the environment, and the social, economic, and political structure of the community. Recovery involves the actions in order to stabilize the community and to restore some semblance of normalcy after the immediate impact of the disaster. Every phase can have different constraints, features or objectives. For instance, in humanitarian relief distribution problems after a disaster strikes, response time encloses the main efficiency and equity measures while monetary goals play a secondary role in such situations. This kind of problems are faced by humanitarian organizations after a man made or natural disaster such a earthquake or flood occurs and a significant number of people have been injured. In this context, vehicle routing problems deal with the distribution of live-saving commodities to injured people while minimum waiting time of visited demand points are required. Issues related to the amount of goods required are also considered if their availability is not enough to satisfy the total demand. Then, minimize the unsatisfied demand is a major goal (De la Torre et al., 2012).

In food distribution logistics there is a particular application arising in charitable contexts. Meals on Wheels (MOW) programs were firstly created in the United States by nonprofit organizations with the aim to provide social services for the elderly and/or poor people. MOW programs operation is based on delivery of food for that people that is not able to shop or prepare lunches for themselves. As Bartholdi III et al. (1983) point out, the funding for MOW programs is unstable and often insufficient, then any additional resource is used to purchase and prepare more food and cover more people from a *waiting list* of citizens. Particularly, in the United States, volatility of a MOW program clients is determined by several founding constraints from federal and state governments to nonprofit organizations with MOW programs. Due to these constraints, vehicle routing decisions for MOW operation must be efficient. In this case, efficiency is measured by serving the maximum number of clients within a list of candidates.

Similarly to MOW programs, food banks are also chariable organizations. Nonetheless, food banks do not prepare the meals. These banks collect and then, deliver the food directly to needy people or other nonprofit organizations (e.g., companies offering MOW services). For food banks organizations, the vehicle routing decisions are made based on problems with pickup and delivery operations. Moreover as pointed out in Balcik et al. (2010) equity objectives are more important than cost/distance related objectives when optimizing this operation.

It is worth to mention that public and nonprofit applications are not only related to health care, humanitarian logistics or food distribution. Previous mentioned contexts are only a subset of examples of several public or nonprofit applications in which vehicle routing decisions are made.

1.3 Bicycle sharing systems: service planning problems and OR perspective

The goal of this section is twofold: firstly, it aims to describe the BSSs operation. Then, a set of service planning problems within a BSS operation from a strategic, tactical and operational point of view are enumerated. These service planning problems are stated as a justification to study BSSs using OR techniques. Needless to say, vehicle routing decisions are highlighted within this section showing the relevance of optimization models and efficient solution strategies. Moreover, this thesis focuses on station-based BSS problems. There exists a different type of service called free-floating BSSs. In the free-floating BSS, bikes are freely parked in defined areas along a urban region. In general, these defined areas are marked with a large rectangle (Zhang et al., 2022). From now, and since free-floating BSSs are not studied in this thesis, we call BSSs to station-based BSSs. Thus, a

definition for BSSs follows.

A BSS is mainly composed by a set of stations with limited capacity (i.e., number of bike slots) distributed along an urban area and a set of bicycles at stations for users. These users can use the system by taking a bike from one of the available stations (origin) and then, after a short trip, returning it to the same or a different station (destination). These systems contribute to a more sustainable mobility and to decrease traffic and pollution caused by combustion based vehicles transportation. The first BSS started operating in 1965 in Amsterdam, The Netherlands (Dell'Amico et al., 2014). Nowadays, there exist more than 2100 BSSs around the world offering approximately 17 millions of bikes (Shui and Szeto, 2020).

As Shui and Szeto (2020) point out, a successful operation of a BSS involves several stages of managerial decisions. Indeed, there are eight main steps to describe the whole service operation of BSSs: (a) bikeway network design, (b) bicycle station design, (c) fleet sizing design, (d) static bicycle relocation, (e) static demand management, (f) inventory level management, (g) dynamic bicycle relocation, and (h) dynamic demand management. A brief description of these steps follows:

(a) **Bikeway network design:** Lin and Yu (2013) define three types of bikeways: bike paths, bike lanes, and bike routes. Bike paths are bikeways completely separated from pedestrians and any vehicle different from bikes. Bike lines are small parts of a sidewalk previously painted for cyclists; the bike lines may be shared with pedestrians. A bike route is a delimited portion of a road generally marked with roadside signs for bike users. Sometimes, in bike routes motorized vehicles are allowed. Bikeway network design problems are related to determining location of bike paths, lines and routes. In these problems several constraints may arise: safety for cyclists, service level, impacts on driving space for vehicles and parking lots (Lin and Yu, 2013; Mesbah et al., 2012; Sohn, 2011).

(b) **Bicycle station design:** In bicycle station design problems two main decisions are made: location for stations and capacity (number of bikes slots). Decisions for station location are relevant as a strategic planning horizon since final locations determine the demand coverage for each station and therefore, for the BSS service. For this kind of problems, financial constraints or even objective functions as investment costs or profits are modeled (Frade and Ribeiro, 2015). Moreover it is possible to find bicycle station design problems in which decisions are also supported in geographic information systems (GISs). These kind of systems allow decision makers to know spatial demand distribution and consider in a precise way, geographical aspects affecting demand patterns (Garcia-Gutierrez et al., 2014; García-Palomares et al., 2012; Larsen et al., 2013; Wang et al., 2016). However, a drawback when using GISs for bicycle station

design problems relies on a higher complexity for models since the amount of data may increase significantly.

Despite the relevance of capacity decisions in bicycle station design problems, they are not considered in free-floating bicycle systems in which bike docks are not required (i.e, users can leave or pick up a bike at any point of the covered zone of the BSS). In this kind of systems, costs related to stations operation are negligible but bike relocation in covered zones become more difficult when compared with traditional BSSs (Shu et al., 2013).

(c) **Fleet sizing design:** Fleet sizing design problems aim to determine a number of bikes for the BSS and a number for bicycles at each station. The total number of bikes to deploy for the whole system is a key decision in fleet sizing design since investment costs must be included within a strategic planning horizon. Nonetheless, to determine this number, the amount of bikes for each station must be also estimated. On the other hand, the number of bikes at each station affects directly the station capacity as a strategic service planning decision (Fricker and Gast, 2016).

(d) **Static bicycle reposition:** This operation, also called bike relocation, is the problem addressed in this research and it is probably the most studied research problem in BSSs service planning context (Shui and Szeto, 2020). A daily successful operation of a BSS requires an adequate number of available bikes at each station in such a way that the desired number of allocated bikes is met and the station capacity (i.e., number of bike slots) is not violated. To do so, a set of capacitated vehicles must visit (all) the stations in order to pick up or deliver bikes according to the lack or surplus at each location. In the static bike repositioning problem (BRP) changes on demands are negligible. In fact, the static version of the repositioning operations is usually performed during the night when the BSS is not available for users or the bikes demand is minimum. As a matter of fact, impact of repositioning operation is more relevant during nighttime than daytime (Laporte et al., 2015).

In scientific literature, it is possible to find several variants for the BRP. Objective functions may include minimization of distance or time required to complete the operation, the minimization of user demand dissatisfaction or even a combination of both metrics within a bi-objective representation of the problem. Variants on the strategies for loading an unloading bikes may also appear: number of visits per station using a vehicle, temporary storage and split delivery (Palacio and Rivera, 2019; Salazar-González and Santos-Hernández, 2015).

(e) **Static demand management:** The static demand management aims to determine strategies and policies to conduct or direct the bicycle demand in order to maximize the available resource utilization in the BSS (i.e., stations, bikeways and bicycles). To do so, it is possible to design policies and regulations or provide deterministic incentives to affect user decisions within a short and mid term horizon. Princing is also considered as a demand management problem (e.g., defining bike rental price, booking prices and station-based prices) (Kumar et al., 2016; Shaheen et al., 2013). In spite of the relevance of static demand management problems, they have not widely studied so far (Shui and Szeto, 2020).

(f) **Inventory level management:** Inventory management in BSSs service planning mainly aims to determine inventory levels of usable and damaged bikes while repositioning operation is performed (Schuijbroek et al., 2017). As a consequence of these decisions is also possible to design a preventive and corrective maintenance programs for usable and damaged bikes respectively. Inventory management problems can be tackled considering a single station independently or also checking the bicycle inventory levels interaction between several stations (Datner et al., 2019).

(g) **Dynamic bicycle reposition:** Similar to the static bicycle repositioning operation, the dynamic version of the problem uses a set of vehicles with the aim to ensure that the total number of required bikes and slots are available at each station while the system is available for users (i.e., during daytime). Thus, the dynamic reposition requires real-time usage levels as an input within a time-dependent problem (Reiss and Bogenberger, 2017). Some of the research based on the dynamic bike repositioning operation as in Regue and Recker (2014) and Zhang et al. (2017) decompose the problem in three main phases: user demand forecasting, pickup and delivery quantities determination and vehicle routing.

(h) **Dynamic demand management:** The dynamic version of demand management problems mainly seeks to design regulations and incentives to reduce differences between real and projected state of stations at any time. These incentives can be created to motivate users to pickup and deliver bikes at stations where excess of bikes and slots is remarkable during the BSS operation, respectively. One of the most common regulations is called dynamic pricing incentive which consists on setting prices for bikes usage once each trip has ended. These prices may depend not only on traveling times but also on origin and destination stations (Singla et al., 2015). Moreover, BSSs users may also perform a small repositioning task by taking or delivering a bike in a near station to start or end trips, respectively (Chung et al., 2018). In the

later case, these repositioning operations may represent a benefit for users (e.g., discount for next trips).

1.4 Purpose of the thesis

As Shui and Szeto (2020) discuss in their recent review, BSSs have been widely study from its logistic and managerial operations fields. Decisions on transportation and vehicle routing in BSSs are not the exception and there exists an extensive number of publications and research about these type of decisions. Particularly, for the static bicycle repositioning problem, being one of the most studied problems within a BSS service planning context, is still possible to find research gaps. These gaps may be viewed from three different perspectives: modeling, solution strategies and even, from repositioning operation design itself.

More precisely, the aim of this thesis is to study the static bicycle repositioning problem designing new mathematical models and solution strategies. Consequently, this study becomes relevant not only from a practical position, but also from a theoretical and algorithmic point of view. Thus, part of this document addresses the BRP as a particular application of the one-commodity traveling salesman problem (1–PDTSP) or the one-commodity vehicle routing problem (1–PDVRP). Particularly, sections devoted to the 1–PDTSP and 1–PDVRP aim to describe competitive models and solution strategies even when large instances for the problem arises. On the other hand and apart from a theoretical and algorithmic view, this thesis also evaluates key features that naturally can be added to the operation with the aim to improve its performance. In addition to the inherent pickup and delivery operation in BRPs, split demand and two-echelon features are also included in this study.

Four different problems are addressed in this document:

a. The one-commodity pickup and delivery traveling salesman problem (1–PDTSP)

b. The one-commodity pickup and delivery traveling salesman problem with split demand (SD1PDTSP)

c. The one-commodity pickup and delivery vehicle routing problem with length constraints (1–PDVRPLC)

d. The two-echelon bicycle repositioning problem (2E–BRP)

The 1–PDTSP is the case where only a single vehicle is available to perform pickups and deliveries through the set of nodes (i.e., stations in a BSS). For small BSSs this problem becomes relevant but, if the number of stations is large enough, then the repositioning operation with one vehicle lacks of practical interest. In this case, a theoretical motivation allows to develop a mathematical model and a metaheuristic algorithm with the aim to find competitive results for the problem.

Similarly to the 1–PDTSP, in the SD1PDTSP, one vehicle must perform a pickup or delivery at each location. Nevertheless, since demand may be split, multiple visits to each node are allowed. Solving a new mixed-integer linear programming (MILP) model it is possible to prove that total traveling cost (e.g., distance) is less than those obtained if only one visit per location is allowed. From a theoretical perspective, when solving the proposed MILP for the SD1PDTSP via commercial solver, it is possible to outperform exact methods reported in the literature.

The motivation to study the one-commodity pickup and delivery vehicle routing problem with length constraints (1–PDVRPLC) is mainly practical. Since several vehicles are available to visit stations, mathematical model and solution strategies are suitable for real BSSs applications with a small or large number of stations. In a BRP, as well as in a VRP, optimal routes for the problem are not typically cost-balanced. Thus, route length conditions may be imposed. Initially, an MILP allows to include soft constraints on route length. However, to deal with medium and large size instances of the problem, two matheuristic algorithms are also presented in this document.

Finally, a two-echelon routing structure is combined with the BRP to define a new problem: the 2E–BRP. In this problem, a vehicle complete a first level (*central*) route and also support the repositioning operation for stations belonging to a second level (*secondary*) routing. In the 2E–BRP, a subset of stations (satellite depots) is visited twice and there, a split demand operation may also occurs. Given the structure of the problem and its inherent pickup and delivery feature, two visits to satellite depots may end up on a temporal storage of bikes or even on a temporal loan of bikes. These scenarios can be seen as a way to integrate the static repositioning with a dynamic version of the problem. For the 2E–BRP, a mathematical formulation and three matheuristic algorithms are described in this thesis.

1.5 Contributions

In the state of the art (Chapter 2), a survey devoted to optimization techniques applied to vehicle routing problems with pickups and deliveries in public and nonprofit contexts is presented. The state of the art emphasizes on vehicle routing in BSSs operations but it also describes research on other

applications as disaster relief, health care and food rescue where models and solution strategies are very similar to those used for BSS decision making processes.

The first problem studied in this document is the 1–PDTSP. Initially, for this problem, an MILP is proposed. This MILP addresses the classical objective function based on total traveling distance (as an operative cost) minimization. Moreovoer, in this mathematical formulation, an adaptation of the Miller, Tucker and Zemlin (MTZ) constraints are included to avoid *subtours* in the optimal solution (Miller et al., 1960). The use of these constraints allows to solve the MILP via commercial solver directly. Since the proposed model is able to solve optimally instances up to 50 nodes, a metaheuristic algorithm is also developed. The proposed metaheuristic is a hybrid multi-start evolutionary local search (MS–ELS) where the local search procedure is replaced by a variable neighborhood descent (VND). The MS–ELS is composed of a greedy randomized algorithm to construct Hamiltonian tours and these tours are then improved using seven neighborhoods within the VND. This solution strategy is finally compared with two state-of-art algorithms available in the literature for the 1–PDTSP.

Similarly, an MILP for SD1PDTSP is presented. In this problem, the pickup or delivery operation can be split into several smaller pickups or deliveries, and also locations can be used as temporal storage points with the aim of reducing the cost of the route. At the end of the route, all pickup and delivery requests must be completely performed. Since the proposed MILP for the SD1PDTSP is an adaptation of the 1–PDTSTP MILP, the model that includes split demand also aim to minimize total traveling cost and avoid *subtours* via MTZ constraints. With this MILP, different scenarios for vehicle capacity are tested and it is possible to show the benefits of split delivery and storage when such capacity is tight. MILP results are also compated with a reported exact approach (a *branch-and-cut* algorithm) in terms of computational times and solution quality.

From a multi-vehicle perspective, an MILP for the one-commodity pickup and delivery vehicle routing problem is developed. As mentioned before, since optimal routes for the problem are not typically cost-balanced an MILP for the 1–PDVRPLC is also presented. In this case, performance and balance for different length parameters are compared. To address the 1–PDVRPLC two matheuristic algorithms are developed: A large neighborhood search (LNS) and an adaptive large neighborhood search (ALNS). Both algorithms include MILPs as destroy and/or repair operators as exact procedures for the algorithms. Similarly, an exact procedure based on an exhaustive enumeration algorithm is presented as a solution repairing strategy. To speedup the performance of the enumeration algorithm, some dominance rules for partial repaired solution are described. Computational experiments compare the performance on the solution strategies for the 1–PDVRPLC and on the enumeration algorithm as repair operator.

Finally, a new variant for the BRP is proposed: the 2E–BRP, where not only routing decisions are made but a location for satellite depots is also decided. This problem is firstly tackled solving an MILP via commercial solver. As the number of stations increase, an approximated solution strategy is required. Thus, MILPs and heuristic procedures are combined to design three different matheuristic algorithms for the 2E–BRP. The first matheuristic is composed of a constructive phase to create Hamiltonian tours which are improved via a VND algorithm. To find a multi-vehicle solution, a split algorithm with limited fleet is also developed. A linear programming model (LP), checks for feasibility of obtained solutions after central routes are heuristically created. The second matheuristic is based on a set partitioning problem (SPP). Before solving the adapted SPP, two sets are heuristically created : the set of central routes and the set of secondary routes. To do so, the constructive algorithm, VND and split procedure are used. As a final step of the matheuristic the enhanced SPP formulation retrieve a 2E–BRP solution. The last matheuristic procedure also combines the constructive algorithm, VND and split procedure to create a large set secondary routes. A suitable combination of these routes is selected via a SPP integer programming (IP) model. Then an adapted generalized traveling salesman problem (GTSP) finds the optimal central route subject to the selected secondary routes. This process is performed in an iterative way with the aim to improve 2E–BRP solution when varying the subset of selected routes.

To the best of our knowledge, not any of the mathematical formulations and solutions methods proposed in this document are reported in scientific literature. Thus, they denote the contribution of this thesis. Similarly, results obtained solving these mathematical models and solution strategies, provide insights and opportunities for future work in vehicle routing problems with pickups and deliveries and, particularly in BSS contexts.

1.6 Structure of the manuscript

This manuscript is structured as follows:

Chapter 2 presents a review on problems, mathematical models and solution strategies for vehicle routing operations in nonprofit and public contexts and particularly in BSS systems. The chapter is divided in three sections: a) vehicle routing problems with pickup and delivery features, b) vehicle routing optimization in non-commercial contexts, c) vehicle routing optimization models in BSSs. The section devoted to pickup and delivery problems summarize two main type of problems: the one-commodity and multi-commodity problems and, dial-a-ride problems. Then, section related vehicle routing in non-commercial contexts describes some of the research based on home and health care logistics, disaster relief logistics, food rescue and delivery problems and other applications.

Lastly, from a more practical perspective, section related to vehicle routing problems in BBSs describes prior work on the static and dynamic rebalancing problem.

Chapter 3 describes mixed-integer linear programming models for the one-commodity pickup and delivery traveling salesman problem and one of its variants when split delivery is allowed (the split delivery one-commodity pickup and delivery traveling salesman problem, SD1PDTSP) . Then, the chapter also presents a hybrid multi-start evolutionary local search algorithm and variable neighborhood descent (VND) procedure for the 1–PDTSP. Lastly, the algorithm performance is compared with two approaches previously reported in the literature.

Chapter 4 presents a mathematical model for the 1–PDVRPLC as well as two metaheuristic algorithms. The mathematical formulation is an MILP which includes route length aspects as a soft constraint. With this MILP it is possible to describe several scenarios for route length and evaluate the impact of this feature on optimal costs and model feasibility. The metaheuristic procedures are based on an LNS and its adaptive version (ALNS). For both algorithms, an adapted version of split algorithm is presented as well as exact and heuristic repair strategies for 1–PDVRPLC solutions. Numerical experiments are described to evaluate the performance on MILP and metaheuristic strategies.

Chapter 5 introduces the 2E–BRPSD where two-level routing decisions are made. A mathematical formulation is presented for the 2E–BRPSD as well as three different matheuristic algorithms. These solution strategies are based on a constructive algorithm, a VND within an improvement phase and, the split procedure. To find 2E–BRPSD solutions, mathematical models based on SPP and GTSP are described. Computational experiments are performed to compare the matheuristic algorithm performance.

Lastly, Chapter 6 as the final section of this manuscript summarizes general conclusions and future research paths.

Chapter 2

State of the Art

In this chapter, a literature review is presented. This review is classified in three main sections: a) pickup and delivery features on vehicle routing problems, b) vehicle routing problems in non-commercial contexts and, c) vehicle routing problems in BSS contexts. Since the main contributions of this thesis are based on optimization and metaheuristic algorithms, the literature review is devoted to these techniques. Simulation, decision analysis, game theory, and other techniques are not included in this thesis.

The first section reviews three pickup and delivery vehicle routing problems: a) the one-commodity pickup and delivery VRP, b) the multi-commodity pickup and delivery VRP and, c) the dial-a-ride problem. The second section, vehicle routing problems in non-commercial contexts, describes prior work on pickup and delivery routing problems in health care, disaster relief and other non-commercial or public contexts. Finally, last section provides details on BSS routing problems. Particularly, the BRP and its static and dynamic version.

2.1 Vehicle routing problems with pickup and delivery features

2.1.1 One-commodity and multi-commodity pickup and delivery VRPs

One-commodity pickup and delivery traveling salesman problem

One of the most relevant approaches to pickup and delivery routing problems from a theoretical perspective is presented in Hernández-Pérez and Salazar-González (2004a). They design a branch and cut (B&C) algorithm based on an integer linear programming model (ILP) for the 1–PDTSP in which the total travel distance is minimized. The solution strategy includes Benders cuts,

and also an adaptation of *nearest insertion* as a heuristic to strengthen upper bounds in the tree. Optimal solutions are reported for instances with up to 50 nodes. Then, in Hernández-Pérez and Salazar-González (2004b), authors propose two different heuristics. The first one, based on the *nearest neighbor* algorithm, computes a modified distance matrix in order to penalize movements leading to infeasible solutions (e.g., by avoiding edges connecting nodes with similar demand). The second heuristic consists on modifying the original B&C algorithm described in Hernández-Pérez and Salazar-González (2004a) to include inequalities is such a way a subset of neighbor solutions can be explored after a 1–PDTSP solution is found. Both heuristic strategies are able to find small gaps even when values for the vehicle capacity are tight and the number of nodes is not larger than 60. Hernández-Pérez and Salazar-González (2007) present new inequalities for the 1–PDTSP adapted from the CVRP (capacitated vehicle routing problem). These inequalities are added to a B&C framework and instances up to 100 customers are optimally solved. Results obtained with this new strategy outperform those reported in Hernández-Pérez and Salazar-González (2004a). Later, Hernández-Pérez et al. (2009) combine a GRASP with a VND procedure to solve the 1–PDTSP minimizing the tour total distance. After the constructive phase, the local search is replaced by a VND (called VND_1) based on edge exchange neighborhoods ($2-opt$ and $3-opt$). After the GRASP scheme is executed, a second VND (VND_2) performs a post-optimization phase using vertex exchange neighborhoods (forward and backward operators). The instances are randomly generated and the number of nodes vary from 20 to 100 while the vehicle capacity goes from 10 to 1000. This GRASP/VND approach is able to find optimal solutions for 96.7% of the instances with up to 50 nodes while new best known solutions are reported for larger instances.

Louveaux and Salazar-González (2009) address a stochastic 1–PDTSP variant in which demand for some customers is modeled as a random variable with a discrete known distribution. To solve this problem, authors propose to create many scenarios where each random demand is replaced by one of its realizations. In this particular problem, apart from routing decisions and initial loads, vehicle capacity must be also determined to handle feasibility issues. Since large values for vehicle capacity are unpractical, the minimization of penalties under a fixed vehicle capacity is tackled. An extensive analysis on vehicle capacity under stochastic demands is also provided in this paper. Computational experiments include instances up to 400 nodes.

Zhao et al. (2009) describe a genetic algorithm (GA) with a pheromone-based crossover operator. This operator uses local and global information to generate new offsprings. While the local information includes edge lengths, adjacency relations, and demands on nodes, the global information is based on pheromone trails. Each offspring is locally improved using $2-opt$ moves and then, the mutation procedure is based on a $3-$exchange operator. This evolutionary algorithm

outperforms the hybrid GRASP/VND presented in Hernández-Pérez et al. (2009) for instances up to 500 nodes, and many new best known solutions for large instances are reported. Later, Mladenović et al. (2012) describe a variable neighborhood search procedure with four neighborhoods based on double-bridge and insertion operators. To test this strategy, computational experiments in Mladenović et al. (2012) include 1–PDTSP instances with up to 1000 nodes.

Han et al. (2016) describe a particular variant of the 1–PDTSP: the 1–PDTSP with restricted depot (1–PDTSP–RD). The motivation to study the 1–PDTSP–RD relies on the non-practical assumption in Hernández-Pérez and Salazar-González (2004a) that allows depot to absorb or provide an infinite number of units. Thus, the 1–PDTSP–RD states that vehicle must leave and return to depot with no load. To deal with this problem, authors propose an ILP and a heuristic algorithm. This algorithm firstly creates two Hamiltonian tours, a pickup tour and a delivery tour in which only nodes with pickup and delivery operations are considered, respectively. As a second step of the algorithm, both tours are merged following one of three possible rules. Finally, $2-opt$ and $3-opt$ operators are applied as improvement phase. Computational experiments are based on a randomly generated set of instances as well as on some of the TSP library (TSPLIB) instances with random values generated for customers demand. In particular, the heuristic algorithm deals with instances up to 1000 nodes while ILP solves instances with up to 15 nodes.

Tüű-Szabó et al. (2020) propose a discrete bacterial memetic evolutionary algorithm (DBMEA) for the 1–PDTSP. This strategy is mainly composed of four steps: create an initial population, mutate bacteria, local search procedures and gene transfer. Firstly, the bacteria population is randomly created. To mutate, bacteria are cloned and divide in segment which are randomly modified. Improvement phase includes $2-opt$ and $3-opt$ operators. Last step in the DBMEA transfers part of *good* quality bacteria to elements classified in a *bad* quality solution group. The proposed approach is able to solve instances with up to 100 customers and outperforms the GRASP/VND results reported in Hernández-Pérez et al. (2009). Recently, Hernández-Pérez et al. (2021) describe a generalization of the 1–PDTSP: the single-vehicle two-echelon one-commodity pickup and delivery problem (2E–1PDP). To determine a first echelon in the problem, a capacitated vehicle visits a subset of customers. For the second echelon, the rest of customers are allocated to visited customers in the first echelon. In the 2E–1PDP, customers are not previously associated with any echelon. Authors propose three different mathematical formulations (two MILPs and an ILP) but they are solved within a B&C framework. After solving instances with up to 60 nodes, results show that the B&C based on the ILP outperforms those algorithms based on MILP.

One-commodity pickup and delivery traveling salesman problem with split demand

As mentioned before, in the 1–PDTSP exactly one visit to each location is mandatory. The SD1PDTSP arises when it is possible to split the pickup or delivery quantity if multiple visits are allowed. The SD1PDTSP is introduced in Salazar-González and Santos-Hernández (2015) as a generalization of the 1–PDTSP and the *split delivery vehicle routing problem* (SDVRP). Since problems addressed on this thesis do not include the SDTSP, the reader may find a detailed description of the problem and solution methods in Archetti and Speranza (2012), Archetti et al. (2014) and Ozbaygin et al. (2018).

Salazar-González and Santos-Hernández (2015) propose an MILP to deal with the SD1PDTSP in which a maximum number of visit to locations is allowed and determined by a parameter. Therefore, the MILP is also able to deal with the 1–PDTSP if that parameter is set to one. As solution strategy, the authors adapt the B&C algorithm in Hernández-Pérez and Salazar-González (2004a) to solve SD1PDTSP instances up to 50 locations. While results are not competitive with other strategies dealing with the 1–PDTSP, this exact approach provides near to optimal results for the split delivery case. Apart from Salazar-González and Santos-Hernández (2015), the SD1PDTP is also study in Hernández-Pérez et al. (2018) with a matheuristic algorithm that applies a constructive procedure and then, a refinement phase to improve solution quality. The constructive procedure is based on a graph in which nodes represent potential visits to locations (i.e., each location is represented by several nodes). Then, the refinement phase uses an adaptation of the MILP presented in Salazar-González and Santos-Hernández (2015) to improve the quality of route pieces. This matheuristic approach is used to solve instances with up 500 locations. Recently, Hernández-Pérez and Salazar-González (2022) solve SD1PTSP with up to 60 nodes via B&C algorithm. In this algorithm, a relaxed MILP is solved, then feasible and invalid solutions are checked in the subproblem where valid cuts are generated for invalid solutions.

Multi-commodity pickup and delivery traveling salesman problem

Multiple commodities may be also allowed for PDVRPs. Firstly, Hernández-Pérez and Salazar-González (2009) present the one-to-one m–PDTSP as a generalization of the TSP, where each commodity has a single origin and a single destination. In their paper, authors describe two MILPs and decomposition techniques based on path and flow formulations to solve the problem. Instances up to 47 nodes were solved and several scenarios for vehicle capacity and number of commodities were also tested. Next, Rodríguez-Martín and Salazar-González (2012) present a matheuristic for the one-to-one m–PDTSP. The authors describe a hybrid approach based on a

GRASP with a VND for the local search procedure. One of the neighborhoods used within the VND is based on an MILP. Given a solution and a small set of arcs to be removed, this MILP finds several solutions by creating different arcs if objective function is improved. This matheuristic is able to deal with instances up to 100 customers. Later, and apart from heuristic techniques, Hernández-Pérez and Salazar-González (2014) propose an MILP for the one-to-one m–PDTSP. They also present two set of valid inequalities. The first one with the aim of strength the linear relaxation of the mathematical model and the second set aims to remove unfeasible arcs in solutions. This strategy is embedded on a B&C framework and it is tested on instances up to 30 customers and three commodities.

Hernández-Pérez et al. (2016) deal with the m–PDTSP within a many to many operation. In this case, each commodity can be transported from several sources to several destinations. Authors propose a three-stage heuristic to solve the problem. Firstly, a greedy procedure based on the *nearest neighbor* algorithm constructs initial Hamiltonian tours. Next, six local search operators are used within a VND procedure in order to improve initial solutions. Finally, with a perturbation based on $3-opt$, the algorithm escapes from local optima. The proposed algorithm is able to solve instances with up to 400 customers and five products. Similar to Hernández-Pérez et al. (2016), Lu et al. (2019) study the many-to-many m–PDTSP and present the population algorithm based on randomized tabu thresholding (PRTTA). PRTTA combines the search intensification of tabu thresholding with diversification advantages of evolutionary algorithms. Similarly to multi-start strategies, PRTTA takes a population of solutions as starting point for the tabu thresholding algorithm. Authors compare the performance of their strategy with results obtained in Hernández-Pérez et al. (2016) and prove that PRTTA is able to improves best known upper bound for 96 out of 108 medium and large instances with up to 400 customers.

2.1.2 Dial-a-ride problems

As mentioned in Doerner and Salazar-González (2014), the dial-a-ride problem (DARP) is a well-known pickup and delivery routing problem for people transportation in public and non-commercial contexts. Therefore, it is important to summarize some of the research on this problem. Even though the DARP has been extensively studied, we focus on the last 12 years developments.

Initially, Parragh et al. (2010) describe a VNS-based heuristic algorithm to deal with the DARP. After a preprocessing procedure based on time windows tightening techniques, authors generate an initial solution under spatial closeness considerations. Then, three different operators for shaking

solutions are embedded within the VNS before the local search is performed. The proposed VNS is able to find new best known solutions for 16 out of 20 tested instances. Later, Parragh and Schmid (2013) solve the problem via a hybrid solution strategy based on column generation and LNS. In this problem, pickup and drop off nodes are fixed (i.e., requested) by users and the total cost of transportation is minimized in presence of time windows, maximum duration of routes and vehicle capacity constraints. The solution strategy is able to find new best solutions for some of the tested instances that vary from 24 to 144 requests. Moreover, the hybrid algorithm delivers new benchmarks on computational times to solve the instances.

Muelas et al. (2013) use a VNS algorithm to solve the DARP. This VNS includes seven different neighborhoods andallowing to deteriorate solutions with a decreasing probability with the aim to scape from local optima. The approach is tested on 12 realistic instances taken from San Francisco city. Solution quality and computational times are competitive with two different algorithms for the DARP: the tabu search proposed in Cordeau and Laporte (2003) and the VNS in Parragh et al. (2010). Later, Liu et al. (2015) present a variant of the DARP which considers multiple trips, heterogeneous fleet, and manpower planning. The authors propose two different MILPs and a set of valid inequialities to strength the formulations. They solve the mathematical models via B&C showing how the inequalities improve significantly lower bounds. These strategies are tested on instances with up to 22 requests.

Braekers and Kovacs (2016) include service quality in the DARP by modeling driver consistency in a multi-period planning horizon. The authors propose two MILPs for the problem and solve them via commercial solver (CPLEX) and also with a B&C strategy. Finally, a metaheuristic algorithm based on LNS is also tested on instances with three levels of driver consistency and up to 627 requests. Masmoudi et al. (2017) consider a DARP in which not only vehicles but requests are heterogeneous. This heterogeneous DARP is solved using a GA that includes local search procedures (i.e., memetic algorithm). The proposed hybrid GA is able to find best known solutions for the homogeneous and heterogeneous DARPs in well-known instances. Authors also compared their algorithm with the state of the art method finding best results in more than 30% of the instances. Ho et al. (2018) publish a literature review on DARPs. They also provide a taxonomy for the problem as well as a complete list of solution strategies.

In Masmoudi et al. (2020) a mixed fleet including heterogeneous conventional and alternative fuel vehicles is tackled. In this problem, decisions on refuel from fuel stations are also made. Moreover, $2-opt$ and relocate procedures are used as local search operators, as well as remove and insert stations for refuel. This ALNS also incorporates five removal operators and four insertion operators. The solution strategy is tested solving the instances described in Masmoudi et al. (2017)

outperforming the previously mentioned GA. Similarly to Masmoudi et al. (2020), Pfeiffer and Schulz (2021) develop an ALNS to solve a variant of DARP: the DARP with ride and waiting time minimization. In this problem, authors do not minimize travel times but the difference between arrival time at delivery location and request time at origin. This ALNS contains seven different removal operators and five insertion operators. The algorithm is tested on 1600 instances from real-case scenarios in Hamburg, Germany, with up to 160 locations. Final analysis concludes that it is possible to reduce waiting times for participating people while the number of vehicles used do not increase significantly.

Recently, Rist and Forbes (2021) present an MILP and a B&C algorithm to solve the DARP where apart from classical time windows, ride time constraints are also addressed. The proposed mathematical model is based on segments of routes that may represent DARP routes. With this approach, authors are able to prove that their algorithm solves nine DARP well-known instances to optimality for first time. For large instances, the B&C requires less computational when compared with state-of-the-art methods. Finally, Ham (2021) proposes a new MILP and three different constraint programming formulations for the DARP with time windows. The proposed MILP uses a reduced number of variables since node merging strategies are applied. The best constraint programming formulation outperforms MILP and it is able to solve instances with up to 60 locations.

2.2 Vehicle routing optimization in non-commercial contexts

This section briefly summarize the literature based on vehicle routing problems in non-commercial contexts. Since the problems we study in this thesis include pick up and delivery operations, the scope of this section is also based on pick up and delivery problems.

2.2.1 Home and health care logistics

There exists several particular problems in the health and home care contexts. For research purposes we are not limited to a single application. We may be interested on home care, home health care and medicine distribution problems, among others. Here, we describe the main of the research in these fields.

Melachrinoudis et al. (2007) present a multi-vehicle and multi-depot DARP based on a real case of a center for addictive behavior health and recovery services (CAB) in Boston, USA. The DARP aims not only to minimize the total traveled distance but the total excess of riding time for patients as well as early and late deliveries (before service) and pickups (after services). The

authors develop an MILP and a TS to solve small instances of the problem including up to four requests and two vehicles. Although the MILP solved via commercial solver is able to find optimal solutions in hours, the TS approach takes seconds to find near to optimal solutions. Liu et al. (2013) address a PDVRP with time windows. They study a problem in which is possible to (a) pick up biological samples from patients home and then deliver them to the labs, (b) pick up unused medicine, materials and medical waste and take them back to depots (e.g. pharmacies), (c) transport products from the depots to patients locations and (d) transport materials and special drugs (e.g. cancer treatments) from hospitals to patients. In this particular problem, delivery nodes can be also pick up points (patients home). This PDVRP is modeled with two different MILPs and also a GA and TS procedures are designed. Authors conclude that TS delivers better solutions than the GA approach for large instances of the problem.

Fikar and Hirsch (2015) design a matheuristic procedure for routing decision of a home health care provider in Austria. This provider operates multiple vehicles with the aim of deliver nurses to patients home and then, pick them up after the service is done. In this context, walking routes may be also defined if a delivery point is different from a pickup location. The authors define this problem as a many-to-many multi-trip DARP in which all the jobs have to be served. The matheuristic includes two stages: firstly, walking routes are created and optimized with a set partitioning model. Then, routing decisions are made with a savings heuristic and a TS algorithm.

Zhang et al. (2015) deal with a special case of the DARP when multiple trips are available (MTDARP) and time windows are imposed. This patient transportation problem also arises from the public patient transportation service in Hong Kong. To solve the MTDARP, authors firstly propose an MILP and then, they describe a memetic algorithm with a population management procedure. Some real-world instances with up to 185 requests and 11 ambulances are taken from the Hong Kong hospital authority. Some benchmark instances for the DARP are tested with the algorithm and the memetic strategy overwhelms state-of-the-art algorithms for this problem. In a similar context, Lim et al. (2016) describe an application from the non-emergency ambulance transfer service in Hong Kong in which a multi-trip pickup and delivery problem with time windows is addressed. Moreover, routing decisions are combined with a manpower planning problem to formulate a lexicographic set of objectives: (a) maximize the total number of attended clients (b) minimize the operational costs and (c) balance the staff workload. To solve this problem, an ILS is proposed where the local search procedure is replaced with a VND. The solution strategy is tested with instances up to 104 requests and 17 vehicles. The strategy is also adapted to solve the multi-trip VRP with time windows and results are compared with state-of-the-art algorithms.

Detti et al. (2017) present a DARP with side constraints to model a real-world case in Italy for a non-emergency patient transportation problem. The DARP includes multiple depots (e.g., hospitals), heterogeneous fleet with vehicles that are compatible with patients conditions (e.g., patients on wheelchairs). A VNS and a TS algorithm are developed to solve the problem with real instances up to 100 requests and 80 vehicles. Both strategies are compared and the VNS with five different shaking procedures provides better solutions than TS on average. Shi et al. (2018) include stochastic travel and service time in a home health care routing problem. In this kind of problems, caregivers must deliver drugs to patients in their location and also must pickup bio-waste and bio-samples. To deal with the stochastic features, authors propose to reduce the problem to a deterministic MILP and solve it using a commercial solver. This MILP is able to find optimal solutions for instances up to 25 nodes. Therefore, four heuristic strategies are also presented: a hybrid GA, SA, bat algorithm (BA) and a firefly algorithm. Since the SA outperforms the other metaheuristics, the stochastic version of the problem is also addressed via SA with instances up to 100 nodes.

Osaba et al. (2019) describe a BA to solve a drug distribution with pharmacological waste collection problem. The main decisions rely on deliveries of prescription drugs and collection of pharmacological waste and expired medicines. This problem is modeled as a multi-attribute VRP (also known as a rich VRP) including pickups and deliveries, asymmetric travel times, forbidden roads and constraints on the cost of each route. The solution strategy is compared with an evolutive algorithm, a evolutionary SA and a firefly algorithm on a real instance based on Bizkaia (Spain) street map and pharmacy locations. To emulate large instances (up to 1000 nodes) authors randomly generate new locations.

Fathollahi-Fard et al. (2020) address the home health care problem (HHCP). In the HHCP, nurses must visit patients at home starting their route in a pharmacy where drugs and medical instruments are collected. Once all patients are visited, nurses must leave biological samples in a laboratory. To tackle this problem, authors present an MILP and a lower bound based on Lagragian relaxation for the home health care problem (HHCP). Similarly, and due to the problem complexity, three heuristic algorithms and a metaheuristic based on SA and VNS are also developed. Tested instances include up to 200 patients and 20 nurses where hybrid metaheuristic is able to find solutions while Lagrangian relaxation provides lower bounds only for small and medium size problems.

Finally, and due to the COVID-19 pandemic, Pacheco and Laguna (2020) describe a pickup and delivery vehicle routing problem for face shields distribution in Burgos (Spain). Since large manufacturer were unable to satisfy total demand, small companies and even citizens volunteered to manufacture the commodity. Thus, governments agencies and non-profit companies seek to deliver raw material to manufactures and also to pickup masks at manufactures and deliver them at hospitals

and health centers. To solve this problem, authors develop an iterative tabu search procedure and solve real instances based on data from Burgos. Daily, up 30 pickup points and 65 delivery points on average were included as locations to solve the problem with eight available vehicles, on average.

2.2.2 Disaster relief logistics

In disaster relief operations, Yi and Kumar (2007) address a multi-commodity PDVRP with split demand in which the main decisions are based on dispatching goods to distribution centers and the evacuation process of injured people to hospital and medical centers after a disaster occurs. Since in this kind of context is not proper to think that total demand of a node is always less than the vehicle capacity, a split demand delivery may be required. The objective function of this problem is the weighted sum on the total number of unsatisfied injured people and the unsatisfied demand. An MILP and an ACO algorithm is proposed to solve this problem. In particular, the ACO procedure builds vehicle routes in a first stage, and then decision on how to dispatch the multiple commodities (i.e., goods and injured people). The MILP is solved via commercial solver and the ACO is able to deal with instances up to 80 nodes and 65 vehicles. As expected, computational times increase for large instances showing that ACO outperforms the commercial solver performance.

Jotshi et al. (2009) design a methodology for dispatching and routing emergency vehicles in post-disaster situations. Authors integrate two different problems: the patient pickup problem and the patient delivery problem. To deal with pickup operations, some key aspects are considered: patient priorities, distance from dispatching location and clustering criteria. Similarly, delivery problem takes into account the distance from patient location to hospitals, waiting time on hospital once the patient is delivered and hospital capacity. Although the methodology to solve this integrated problem is based on a simulation model, decisions for routing are made via shortest path algorithms. Wohlgemuth et al. (2012) deal with the last mile planning problem and logistic operations in a disaster relief chain. From a dynamic perspective, authors model this problem as a PDVRP since most of the locations (nodes) can receive and send goods. Additionally, time dependent travel times and time windows are included in the formulation of the problem with the aim to estimate non-served on time locations. As solution strategies, an MILP and a TS algorithm in which total travel time and the number of vehicles are minimized using a weighted sum objective function.

Sabouhi et al. (2019) describe an integrated evacuation and distribution logistic systems in which routing and scheduling decisions are made. Pickup and delivery operations are determined by evacuation of affected people to shelters and by providing supplies, respectively. In this process, split delivery is also allowed since demand at some locations can exceed vehicle capacity. This

problem is modeled as an MILP minimizing the arrival times of vehicles. The authors also present a memetic algorithm (MA) able to find optimal and near optimal solutions for small (up to 15 affected areas and seven shelters) and large instances, respectively. This metaheuristic strategy is also tested for real case study in Tehran (Iran). Sakiani et al. (2020) incorporate inventory management to routing decisions for a multi-period problem where operational and deprivation costs are minimized. In a first stage of the problem, relief goods and supply are distributed from distribution centers to local depots. Moreover, redistribution decisions can be made between local depots. Thus, pickup and delivery operations arise. Lastly, vehicles transport commodities from local depots to demand points. To solve the problem, a SA based matheuristic is proposed. While SA finds values for routing decision variables, an MILP determine inventory levels and loading/unloading quantities. Sample instances do not exceed 20 locations (distribution centers and local depots). A real case study based on a earthquake in Iran is also described.

2.2.3 Other applications

Despite home health care and disaster relief are two of the main applications for pickup and delivery operations in non-commercial contexts, there exist other emerging applications as food rescue and delivery problems and library VRPs.

Food rescue and delivery problems

Rey et al. (2018) define the food rescue and delivery problem (FRDP) theoretically inspired on the PDVRP and in a practice sense on the collection and re-distribution of food for hunger relief. Initially, authors state a 2-index formulation (MILP) for the FRDP with multiple vehicles and homogeneous capacity. To solve this formulation, they propose a Bender's decomposition approach within a cutting plane algorithm to generate subtour elimination constraints. On the other hand, a heuristic strategy is proposed with a cluster-first route-second algorithm in which ideas from **?** are taken to make routing decisions. Three different set of experiments are performed in Rey et al. (2018): small instances up to 50 nodes solved via cutting-plane algorithm, large instances up to 300 nodes with the heuristic approach and a real case from Sydney, Australia with 200 nodes. Later, Nair et al. (2018) present the periodic version of the FRDP by modeling a scheduling and routing scheme in which days and times to visit food providers and welfare agencies are determined. For routing decisions, the total transportation cost is minimized and pickup and delivery quantities are established. This combined problem is tackled via commercial solver using an MILP and with a TS procedure which final solution is improved using a post-optimization LS algorithm based on $2-opt$

movements. As well as Rey et al. (2018), Nair et al. (2018) solve a real-case instance from Sydney, Australia and also some literature instances with up to 334 nodes.

Eisenhandler and Tzur (2019) describe a *humanitarian pickup and distribution problem* which is particularly relevant for perishable products distribution (i.e., to store products in warehouses for long periods is not allowed). Decisions in this problems are not only based on routing and pickup or delivery quantities but also on which pickup and delivery locations are visited (donation suppliers and agencies, respectively). Authors propose an MILP for the problem and also, a LNS algorithm with 12 possible neighborhoods is presented. Instances in this study include a real case scenario from Israel food bank with up to five suppliers and 62 welfare agencies. A second set of instances includes from four to 12 suppliers and 10 welfare companies. A final data set with 22 instances is randomly generated with up to 100 locations. Alhindi et al. (2020) address a similar problem for a non-profit institution in Makkah (Saudi Arabia). In this particular case, an MILP and a simulation model are described. Authors show the benefits on optimization techniques by solving the mathematical formulation via commercial solver if 20 pickup locations are available.

Library vehicle routing problem

Apart for the previously mentioned problems, it is possible to find research on PDVRPs based on other nonprofit contexts. One example is based on the library vehicle routing problem (LVRP) which include pickup and delivery features. Chen et al. (2015) address the LVRP, formulated as a paired many-to-many pickup and delivery problem. More precisely, this LVRP consists on finding routes in which items (e.g., books, videos, library materials) are picked up and delivered in satellite libraries (branches). In this particular application, three minimization objectives are combined in a weighted sum: total travel time, deviations from the average vehicle travel time and unsatisfied demand of items. To deal with this problem, authors propose an MILP and a two-stage solution algorithm based on: first, insertion methods to create initial solutions and second, a bee colony optimization method to improve solutions found in the first stage. These strategies are tested with real data obtained from San Francisco (USA) library system where 26 library branches are available.

2.3 Vehicle routing optimization models in bicycle sharing systems

This section presents a state of the art on static and dynamic bike repositioning problems. Since free-floating BSSs are not studied in this thesis, all the literature presented in this section is devoted

to station-based BBS services. Some of the most relevant work in free-floating services may be found in Cheng et al. (2021); Du et al. (2020); Liu et al. (2018); Mahmoodian et al. (2022); Pal and Zhang (2017); Usama et al. (2019).

2.3.1 Static bicycle repositioning problems

In BSSs, probably the most studied problem is the BRP. The most recent research based on the problem, begin with Raviv et al. (2013). They formulate two MILPs for the BRP which consider the minimization of the weighted sum of operational cost and users dissatisfaction as objective function. To measure the user dissatisfaction with the system, authors propose an index based on the number of shortage events. These shortages can emerge not only if a user wishes to take a bike and the station is empty but also if the user want to return a bike in a station with no parking slots available. To solve this problem, two different MILPs models are tested on instances with up to 60 stations based on certain locations of Vélib (BSS in Paris) and then, a complete real instance of 104 stations and one or two vehicles. Relative small gaps are obtained within a maximum time of 18,000 seconds. Chemla et al. (2013) code a B&C procedure for the BRP. This algorithm is based on an MILP relaxation for the problem and provides lower bounds when several visits to each vertex are allowed. On the other hand, a TS with four different neighborhoods is also designed to find upper bounds. Instances that vary from 20 to 100 stations are evaluated and it is possible to find small gaps (less than 5% as an average) for instances with up to 60 stations. For the set of largest instances, gaps increase and the local search procedure is not quite efficient since the size of neighborhoods grows significantly.

Dell'Amico et al. (2014) present four MILPs for the multi-vehicle BRP in which the total distance of the routes is minimized. They solve these mathematical models via B&C algorithm. The solution strategy was tested in 65 instances adapted from 22 different BSSs around the world and the formulation with best computational performance is able to solve optimally all the instances with up to 50 nodes in less than 15 minutes. Authors also present a real-world case using data from Reggio Emilia in Italy. Although the Reggio Emilia BSS has only 13 stations and about 100 bikes, they analyze travel flows, user behavior and, finally, the occupation levels in each station in order to build the real instance. Similar to Raviv et al. (2013), Ho and Szeto (2014) model penalty functions to minimize the cost associated to unsatisfied demand as a single objective function. Nevertheless, they do not consider the operational cost of the route (i.e., total distance). To deal with the single-vehicle case, the authors use a TS procedure and also test an MILP in CPLEX. The TS algorithm includes three different neighborhoods based on removal, insertion and exchange moves

and the computational experiments are based on 156 instances varying from 30 to 400 stations.

Forma et al. (2015) propose a 3-step matheuristic based on: (a) a clustering process supported on savings heuristic, (b) an MILP for vehicles routing though clusters and (c) an MILP (adapted from Raviv et al. (2013)) for repositioning decisions in a reduced network (i.e., each cluster). In this paper, authors also include in their mathematical formulations new constraints to ensure that total time allocated for static repositioning is not violated. For instances up to 150 stations, the matheuristic outperforms the arc-indexed formulation in Raviv et al. (2013) obtaining smaller gaps. For some larger instances (up to 200 stations) it was not possible to find optimal solutions but gaps are not larger than 2.52%. Rainer-Harbach et al. (2015) propose a greedy randomized adapted search procedure (GRASP) to solve the BRP. This GRASP is also extended to a PILOT algorithm able to evaluate candidates to add to solutions in a recursive way. Authors also develop a VND as a third solution strategy which outperforms GRASP/PILOT in medium size instances. However, GRASP/PILOT finds better solutions than those reported by VND in large instances.

Similar to Chemla et al. (2013), Erdoğan et al. (2015) deal with the BRP when several visits to a single node are allowed. Apart from the presented heuristic algorithms for the problem in previous articles, authors describe an exact approach based on a separating algorithm for Bender's cuts. To test the algorithm, instances with up to 60 nodes are solved to optimallity within a computational time of two hours. Kadri et al. (2016) propose a *branch-and-bound* (B&B) procedure to solve the BRP when minimizing the total waiting time of stations. While lower bounds are computed by skipping some constraints of the problem and via lagrangian relaxation, the upper bounds are calculated by means of a GA, a greedy search (GS) or a nearest neighbor procedure (NNP). For instances with up to 30 stations the B&B delivers solutions with gap up to 13% from the optimal solution within a maximum computational time of 7,200 seconds. For larger instances, GS finds better quality solutions than GA and NNP. Nevertheless, GS requires larger computational times.

Dell'Amico et al. (2016) propose a destroy and repair (D&R) metaheuristic for the BRP with maximum length tour, a variation of the BRP that includes constraints for maximum duration for each route. Initially, the D&R algorithm starts with a constructive phase in which a variant of *savings* heuristic is used to find an initial solution. Next, after some nodes are removed from routes, the solution is repaired via insertion procedure or the adapted *savings* algorithm. Finally, local search procedures are embedded in a VND framework. This D&R metaheuristic is tested on instances with up to 500 stations and the previously reported B&C algorithm (Dell'Amico et al., 2014) is also adapted in order to find lower bounds for problem. For small instances (less than 50 stations) it is possible to find optimal solutions. Larger instances are solved but gaps increase, nevertheless new best known solutions are presented. Alvarez-Valdes et al. (2016) present a two-stage methodology

to address the rebalancing problem. First, they estimate shortages of bicycles and free slots at each station for each possible number of available bikes at the beginning of a time period. Second, they propose an MILP to find the optimal number of bikes that each station must have in order minimize the total dissatisfaction. Then, the routes for the vehicles are designed minimizing not only the total cost but also the variation over the duration of the set of routes. To do so, these authors solve a minimum cost flow problem to estimate the number of bikes that should be transported along the network. Then, an insertion heuristic guides the route construction. Results for this approach are based on the Palma de Mayorca's BSS case which is a small system with 28 stations. Results show that although a perfect balance on service times is not possible for a time horizon of one week, the routes never differ by more than 15 minutes. Espegren et al. (2016) propose an MILP to solve the BRP with an heterogeneous fleet. In this case, multiple visits to each station are allowed and the objective of the model relies on minimize a non-perfect repositioning operation (i.e., minimize the dissatisfaction of users). The authors tested this MILP on real-world instances with up to 14 stations from a BSS in Oslo, Norway.

Cruz et al. (2017) design an iterated local search (ILS) algorithm for the single-vehicle case of the BRP. The proposed ILS includes up to four different perturbation operators and a VND procedure instead of a local search. The authors coded six different neighborhoods and they are explored in a random fashion (RVND). This algorithm allows split demand and deliver solutions in which only stations with non-zero demand must be visited. After a parameter tuning and a selection of perturbation operators subset, this ILS outperforms the tabu search presented by Chemla et al. (2013) for instances with up to 60 stations. Moreover, for larger instances with up to 100 nodes, the ILS finds new best known solutions for instances previously reported in the literature. Finally, Ho and Szeto (2017) also address the BRP with penalty functions for unsatisfied demand. Authors propose a hybrid large neighborhood search (HLNS) algorithm. This hybrid metaheuristic includes five removal (destroy) operators, five insertion (repair) operators and a TS applied to the most promising solutions. Testing instances with up to 518 stations, the HLNS is able to outperform a proposed MILP coded on CPLEX and the matheuristic described in Forma et al. (2015).

Wang and Szeto (2018) propose a new variant for the BRP called the green repositioning problem with broken bikes. This study is the first one that includes environmental issues for static and multi-vehicle BRPs. The authors present an MILP for the problem in which the total CO_2 emissions are minimized. The mathematical model is tested using small real-world instances of Citybike in Vienna via commercial solver. To solve a large instance with 90 stations, a clustering method based on nearest neighbor heuristic is used before the solver is called. They also discuss some properties and critical factors that increase vehicle emissions. Indeed, emissions increase as

the percentage of broken bikes also increase. Nontheless, the study also analyze multiple-visit to stations and prove that this is a key factor to reduce CO_2 emissions. Bulhões et al. (2018) tackle the multi-vehicle and multiple-visit BRP via ILS. A B&C algorithm is also described in order to compute lower bounds. These solution strategies jointly finds optimal solutions for most of the real-world instances originally presented in Dell'Amico et al. (2014) using up to three vehicles. Later, Wang and Szeto (2021b) also deal with the green repositioning problem with broken bikes for the single-vehicle version of the problem. This study includes an enhanced artificial bee colony algorithm to generate the routes and two different methods (one network flow mathematical model and one heuristic) are used to compute the loading and unloading quantities. The algorithm also includes a local search procedure to improve solutions. The authors test the performance of the algorithm solving instances with up to 300 stations.

Lu et al. (2020) describe a MA based on a greedy constructive procedure to generate initial solutions, an ELS for the improvement phase and an adaptive randomized mutation procedure. This algorithm is able to improve best known solutions for 46% of the evaluated instances with up to 564 stations. Computational times also outperform the D&R algorithm described in Dell'Amico et al. (2016). Pan et al. (2020) develop a TS algorithm and a heuristic algorithm called *capacity range length heuristic* which is based on some of the properties for feasible insertions moves in a BRP route. Authors compare the performance of these strategies and conclude that TS is able to improve solutions reported with the B&C in Dell'Amico et al. (2014) and the D&R algorithm in Dell'Amico et al. (2016) even for large instances with 564 stations.

Lee et al. (2020) describe a selective BRP in which not all the pickup stations must be visited. In this study, the authors show that it is possible to improve service level for bikes availability if this repositioning strategy is implemented. The proposed selective pickup and delivery BRP aims to minimize the total travel time and the number of used vehicles for the relocation operation. To solve the problem, a genetic algorithm is coded and real-world instances based on a BSSs from Gangnam-district in Seoul with 95 stations are tested.

Recently, Fu et al. (2022) proposed a robust optimization approach to address demand uncertainty by jointly considering strategic and tactical decisions as station location and bike rebalancing, respectively. Specifically, the model determines stations location, initial inventories and service areas for vehicles. To deal with this problem, authors present an MILP with non-convex constraints. Therefore, duality theory is applied to reformulate the problem and it is solved via row generation. Tested instances include up to 55 stations. Finally, Jia et al. (2021) include traffic conditions to BRP with a mixed fleet of internal combustion vehicles and electric vehicles. The problem is described via MILP and also a hybrid SA algorithm with variable neighborhood structures is used to solve it.

Authors randomly generated nine instances with up to 100 stations and prove that hybrid SA with variable neighborhood structures, outperforms SA and VNS as separated algorithms also developed for the problem.

2.3.2 Dynamic bicycle repositioning problems

Initially, Contardo et al. (2012) describe the dynamic BRP motivated by shortages of bikes at some station during peak hours. The authors present a mathematical formulation able to solve small instances when total unmet demand is minimized. Moreover, they present two decomposition schemes. The first one, based on Dantzig-Wolfe decomposition, finds lower bounds for the problem. The second strategy follows a Benders decomposition strategy to find values for continuous decision variables. To test these strategies, Contardo et al. (2012) solve 120 instances with three different sizes: 25, 50 and 100 stations. Despite decomposition strategies are able to find lower and upper bounds, final gaps increase significantly as the size of the instances also increases. Caggiani and Ottomanelli (2013) describe a simulation model for the dynamic BRP where variations of demand are considered. Thus, distribution patterns, repositioning flows and time intervals between repositioning operations are evaluated. Authors show the benefits on this simulation model by computing users satisfaction measured as the probability of finding an available bike or a free slot to deliver a bike.

Kloimüllner et al. (2014) propose four different algorithms to deal with the dynamic version of the BRP: a greedy construction heuristic, a preferred iterative look ahead technique (PILOT) algorithm, a VNS and GRASP. Instances with a number of stations that varies from 30 to 90 were used to test the algorithms. Some scenarios (i.e., demand values) were taken from Vienna BSS. Authors conclude that VNS metaheuristic outperforms the other three proposed algorithms for most of the evaluated instances. Zhang et al. (2017) describe a multi-commodity time-space network flow model. Since this formulation is non-linear, authors reformulated it as an MILP. The proposed solution strategy is mainly based on a two-stage approach where firstly a linear relaxation of the MILP is solved and then, routes are assigned to determine upper bounds. Tested instances were taken from real scenarios from Washington and Paris BSSs with up to 200 stations.

Shui and Szeto (2018) introduces a new BRP based on the minimization of user dissatisfaction (i.e., unmet demand) and CO_2 emission costs within a bi-objective model. To compose the dynamic nature of the problem, authors model a multi-period operational horizon in which demands vary at each station and at each period. This problem is split in a set of steps, each step solves a single period of the problem as a static version of the repositioning operation. The solution strategies rely

on an enhanced ABC algorithm and a GA. The size of the instances vary from 30 to 180 and for most of the instances, ABC algorithm outperforms the GA. Authors also present an analysis of the impact of several variables (e.g., weight of each objective, length of the service time horizon, time for load and unload bikes) on unmet demand and CO_2 emissions.

Later and apart from mathematical programming models and heuristic approaches, Legros (2019) develop a DSS based on a Markov decision process. This Markov process aims to minimize the rate of arrival of unsatisfied users. In this paper, authors design a one-step policy with the aim to determine prioritization rules throughout the set of stations. This one-step policy outperforms other policies build as simpler prioritization rules. Recently, Zhang et al. (2021) propose a model in which station demand is defined via neural network (NN). The NN forecasts the number of bikes and bike slots required in fixed time intervals for each location. To solve the problem, a hybrid metaheuristic is proposed with the aim to minimize penalty costs for unmet demand. This strategy combines an adaptive genetic algorithm and a granular tabu search named AGA-GTS. While the GA finds an initial repositioning plan, the GTS solves the repositioning problem under demand patterns provided by NN. To test the solution strategy, instances with 91 stations and based on Shouguang (China) BSS are solved.

Lastly, Wang and Szeto (2021a) consider a mixed fleet of internal combustion engine vehicles and battery electric vehicles in the dynamic BRP with multiple charging technologies. As objective, this problem aims to minimize the of penalty costs and the charging costs of vehicles within a weighted sum function. The solution strategy is based on a hybrid ABC algorithm and a dynamic programming model. Test instances include a subset of the 100 most active bike stations from Washington BSS.

2.4 Concluding remarks

The vehicle routing problem and its associated features in shared mobility contexts as BBSs (e.g., pickup and delivery) have been widely studied. Exact methods made possible to find optimal solutions for small instances of the problem. However, given the complexity of VRPs, many authors consider the use of heuristic, metaheuristic and hybrid strategies to solve instances associated with real case studies of BSSs. On the other hand, solution strategies for BRPs and 1–PDVRPs have been adapted to include other desirable features related to the operation design for BSSs (e.g., split delivery, routes lengths), or related to operation efficiency (e.g., CO_2 emissions, users dissatisfaction).

In spite of the extensive research on pickup and delivery vehicle routing optimization within theoretical and practical contexts as in BSSs, there still exist gaps between repositioning problems and solution strategies to solve them. Moreover, so far there is not evidence of publications based on several VRP key features for bike sharing mobility (e.g., collaborative and two-echelon routing). Therefore, this research aims to fill some of those gaps by developing models and metaheuristic or hybrids strategies to support decision making processes within vehicle routing optimization in BSSs contexts.

Chapter 3

One-commodity pickup and delivery traveling salesman problems: mathematical models and metaheuristic approaches

3.1 Introduction

This chapter addresses the 1–PDTSP and the SD1PDTSP, which are generalizations of the well-known traveling salesman problem. The 1–PDTSP aims to find a Hamiltonian tour in which a set of supply points (pickup locations) and demand points (delivery locations) are visited once while the total traveled distance is minimized. The SD1PDTSP, is a relaxed case of the 1–PDTSP where locations can be visited several times. In the SD1PDTSP, the pickup or delivery operation can be split into several smaller pickups or deliveries, and also locations can be used as temporal storage points with the aim of reducing the cost of the route.

With the aim to solve large instances of the 1–PDTSP, this chapter describes a hybrid metaheuristic based on multi-start evolutionary local search and variable neighborhood descent to solve the problem. To test the performance of the algorithm, instances with up to 500 nodes available in the literature are solved. It is possible to demonstrate that the approach is able to provide competitive results when comparing to other existing strategies. Since a direct application of the 1–PDTSP arises as the bicycle repositioning problem (BRP), the proposed metaheuristic algorithm is used to solve a set of real-case instances based on EnCicla, the BSS in the Aburrá Valley (Antioquia, Colombia).

Moreover, only Salazar-González and Santos-Hernández (2015) and Hernández-Pérez et al.

(2018) have tackled the SD1PDTSP. This chapter addresses this problem by providing four main contributions: a) a new mathematical formulation for the SD1PDTSP, b) a numerical analysis of the benefits on split demand and temporal storage in PDTSPs, c) a metaheuristic algorithm able to solve the 1–PDTSP and, d) a set of computational experiments solving instances with up to 60 locations, showing a competitive performance based on results in Salazar-González and Santos-Hernández (2015). Properties and benefits of split demand for the 1–PDTSP are considered again in Chapter 5.

The remainder of this chapter is structured as follows. Section 3.2 formally defines the 1–PDTSP and the SDPDTSP, and describes the proposed MILPs to deal with both problems. Section 3.3 describes a multi-start evolutionary local search algorithm to solve the 1–PDTSP. Then, Section 3.4 summarizes the main results on MILPs and the metaheuristic approach. Finally, Section 3.5 outlines some conclusions.

3.2 Problem definition and mixed integer linear models

This section presents a mathematical formulation for the 1–PDTSP as well as an MILP for the SD1PDTSP by including new constraints to the 1–PDTSP model. A graphical description for the split delivery and the temporally storage operations and their impacts in two small instances are also presented.

3.2.1 The 1–PDTSP

In the 1–PDTSP a set of locations and a capacitated vehicle are given. The locations are classified as supply points (pickup locations) and demand points (delivery locations). One commodity is transported between the locations using the available vehicle. Each supply point provides a certain amount of the commodity and these units can be delivered to one or several demand points (Hernández-Pérez and Salazar-González, 2004b). Additionally, one of these locations is set as a depot where the vehicle starts and ends the tour. Thus, the 1–PDTSP aims to find a Hamiltonian tour in which the total traveled distance is minimized and the vehicle capacity is satisfied. Moreover, an initial load must be determined for the vehicle.

A well-known application for the 1–PDTSP relies on the operation of BSSs. The BRP has been mainly tackled in its static version which assumes that there are no changes on stations demands (this mainly occurs at night when BSSs are not available for users). On the other hand, on dynamic BRPs, pickup and delivery quantities may vary (e.g. the repositioning operation is performed during the BSS operation).

The 1–PDTSP is modeled on a complete and directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N} = \{0, 1, ..., n\}$ is the set of nodes and $\mathcal{A}$ is the set of arcs between each pair of nodes. Without loss of generality, the location 0 is the depot but it is also considered as a node in the problem. For each arc $(i, j) \in \mathcal{A}$, there exists a positive traveling cost from i to j ($i \neq j$) as c_{ij}. Additionally, for each node i ($i \in \mathcal{N}$), a parameter q_i (where $q_i \in \mathbb{Z}$) represents the demand on node i. If $q_i < 0$, then node i requests a pickup of $|q_i|$ units. On the other hand, if $q_i > 0$, then a delivery operation is requested in node i and q_i units must be unloaded. These operations are performed by one available vehicle with capacity Q. It is assumed, without loss of generality, that $|q_i| \leq Q$. With the aim to describe a mathematical model for the 1–PDTSP, a binary decision variable, y_{ij} is defined where $(i, j) \in \mathcal{A}$. Variable y_{ij} takes the value of one if the vehicle traverses the arc (i, j) and zero otherwise. Moreover, a variable l_{ij} denotes the load of the vehicle on arc (i, j) when it is used in the solution. Finally, variable z_{ij} assigns a label for arc (i, j) with the aim to avoid subtours in final solution. The proposed mathematical formulation for the 1–PDTSP is a mixed-integer linear programming model (MILP) as follows:

$$\min f = \sum_{(i,j)\in\mathcal{A}} c_{ij} \cdot y_{ij} \tag{3.1}$$

subject to,

$$\sum_{\substack{j\in\mathcal{N} \\ i\neq j}} y_{ij} = 1, \qquad \forall\, i \in \mathcal{N} \tag{3.2}$$

$$\sum_{j\in\mathcal{N}} y_{ij} = \sum_{j\in\mathcal{N}} y_{ji}, \qquad \forall\, i \in \mathcal{N} \tag{3.3}$$

$$l_{ij} \leq Q \cdot y_{ij}, \qquad \forall\, (i, j) \in \mathcal{A} \tag{3.4}$$

$$\sum_{j\in\mathcal{N}} l_{ji} - \sum_{j\in\mathcal{N}} l_{ij} = q_i, \qquad \forall\, i \in \mathcal{N} \tag{3.5}$$

$$\sum_{j\in\mathcal{N}} z_{ji} - \sum_{j\in\mathcal{N}} z_{ij} = 1, \qquad \forall\, i \in \mathcal{N}\backslash\{0\} \tag{3.6}$$

$$z_{ij} \leq |\mathcal{N}| \cdot y_{ij}, \qquad \forall\, (i, j) \in \mathcal{A} \tag{3.7}$$

$$y_{ij} \in \{0, 1\}, \qquad \forall\, (i, j) \in \mathcal{A} \tag{3.8}$$

$$z_{ij}, l_{ij} \geq 0, \qquad \forall\, (i, j) \in \mathcal{A} \tag{3.9}$$

The objective function in (3.1) aims to minimize the total traveled distance. Equations in (3.2) ensure that each node is visited exactly once, while constraints in (3.3) enforce to leave a node once it is visited. Constraints in (3.4) limit the maximum load when traversing any arc, to the

vehicle capacity. Equations in (3.5) force the model to ensure a flow conservation along the used arcs. Inspired on Miller et al. (1960), equations in (3.6) state a coefficient for each arc and avoid subtours in the solution. Constraints in (3.7) limit the maximum value of arc coefficients. It is worth to mention that for classical TSP and VRP formulations, constraints (3.4) and (3.5) prevent subtours. Nevertheless, for pickup and delivery operations, as q_i can be negative, (3.4) and (3.5) are not enough. Constraints in (3.8) and (3.9) define the domain of decision variables. Note that variables l_{ij} and z_{ij} could be integer, but given the structure of the formulation, a continuous domain will lead to integer values.

It is worth to mention that for 1–PDTSP meet that $q_i \leq Q \; \forall i \in \mathcal{N}$. Each node in $\mathcal{N}$, even the depot (node 0) is visited exactly once (see equation (3.2)). However, the depot is able to absorb or provide the remaining number of units to ensure flow conservation for any 1–PDTSP instance as follows:

$$q_0 = -\sum_{i=1}^{|\mathcal{N}|} q_i \tag{3.10}$$

Conditions described in (3.2) also force the vehicle to visit even the nodes with demand equal to zero. Note that if nodes where $q_i = 0$ must not be visited, they can be removed from the set $\mathcal{N}$ as a preprocessing procedure. To describe graphically a 1–PDTSP solution, Figure 3.1, shows the optimal tour for a vehicle with capacity $Q = 10$ considering an instance with 20 nodes. While the first value near to each location represents the number of units to pick up or deliver, the second one denotes the load of the vehicle when entering to such location. To compute this load, l_k is defined as the number of units in the vehicle after visiting k nodes. Thus, if i is the k-th visited location, l_k is computed as: $l_k = l_{k-1} - q_i$. For example, in this solution, the vehicle traverses the arc $(0, 10)$ with a load of 10 units. Then, in node 10, nine units are delivered and the vehicle arrives to node 9 with a load equal to one. In node 9 there is not pickup or delivery operation, thus arc $(9, 18)$ is traversed with one unit in the vehicle. In location 18, the vehicle picks up seven units. Note that at the end of the path, the vehicle goes into 0 with three units which represent the same initial load to begin the tour.

Any Hamiltonian tour is a feasible solution for the 1–PDTSP if the load of the vehicle does not exceed the capacity Q and it does not become negative. In more general way, when completing a Hamiltonian tour for 1–PDTSP it is possible to check feasibility if the difference between the minimum and the maximum load of the vehicle is not greater than Q. This is a straightforward way to check feasibility in a 1–PDTSP solution:

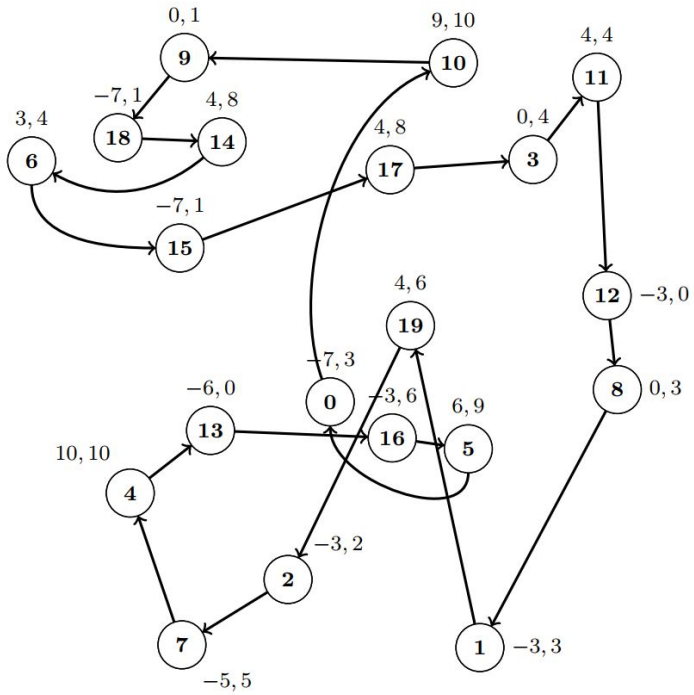


Figure 3.1: Optimal solution for a 1–PDTSP instance with 20 nodes

$$\max_{i\in\mathcal{N}}\{l_i\} - \min_{i\in\mathcal{N}}\{l_i\} \leq Q \tag{3.11}$$

The reader may notice that depending on initial values for l (i.e., l_0) the minimum load in a path may end up in values less than zero or for maximum loads, values could be greater than Q. However, if (3.11) is meet, feasible values for initial load can be deduced. If $\min_{i\in\mathcal{N}}\{l_i\} < 0$, then a real value for minimum load is $|\min_{i\in\mathcal{N}}\{l_i\}|$. On the other hand, if values for $\max_{i\in\mathcal{N}}\{l_i\}$ are greater than Q, then real maximum load is $Q - \max_{i\in\mathcal{N}}\{l_i\}$.

3.2.2 The SD1PDTSP

As mentioned before, in the 1–PDTSP exactly one visit is mandatory to each location . Nevertheless, it is possible to split the pickup or delivery quantity if multiple visits are allowed. This variant, which is known as the SD1PDTSP, is introduced in Salazar-González and Santos-Hernández (2015) as a generalization of the 1–PDTSP and the *split delivery traveling salesman problem* (SDTSP). The SD1PDTSP can be modeled by introducing new constraints on model in (3.1)–(3.9). In the split delivery scenario, the vehicle can visit a location i several times in order to meet the total pickup or delivery quantity (i.e., q_i). Let q_i^v be the number of units to pickup or deliver in location i on visit v. Then, a feasible solution must satisfy that $\sum_v q_i^v = q_i$. On MILP (3.1)–(3.9) (henceforth called

MILP$_{SD}$), the total demand of each location is described by equations in (3.5). Therefore, to allow multiple visits to node i ($i \in \mathcal{N}$), constraints in (3.2) can be replaced by constraints (3.12):

$$\sum_{\substack{j \in \mathcal{N} \\ i \neq j}} y_{ij} \geq \left\lceil \frac{|q_i|}{Q} \right\rceil, \qquad \forall\, i \in \mathcal{N} \tag{3.12}$$

It is worth to mention that constraints in (3.12) also allow to that demand in nodes can exceed vehicle capacity (i.e., $|q_i| > Q$). In addition, when one or more locations are visited several times by the vehicle, the number of arcs in the solution becomes greater than $|\mathcal{N}|$. Then, constraints in (3.7) are replaced for a similar set of constraints defined in 3.13:

$$z_{ij} \leq 2 \cdot |\mathcal{N}| \cdot y_{ij}, \qquad \forall\, (i,j) \in \mathcal{A} \tag{3.13}$$

The expression $2 \cdot |\mathcal{N}|$ is an upper bound for the value of variables z_{ij}. Given the continuous nature of these variables, the value of the upper bound does not affect the performance of MILPs. Replacing equations in (3.2) by expression in (3.12) and updating the upper bound for z_{ij} as in (3.13), a set of constraints Ω is defined by expressions in (3.3) to (3.6), and (3.8) to (3.13). Then, the SD1PDTSP can be modeled as follows:

$$\min f = \sum_{(i,j) \in \mathcal{A}} c_{ij} \cdot y_{ij}$$

$$\text{subject to,} \qquad \Omega$$

Finally, if multiple visits can be performed to location i, expressions in (3.5) allow to a couple of variables l_{ij} and l_{ji} to take values not necessarily equal to q_i. This would lead to take location i as a temporal storage point where units can be picked up in a later visit or some units can be taken away from i temporally until a future visit occurs.

Next, two graphical examples for the the SD1PDTSP are presented. It is important to remember that if several visits are performed, then $\sum_v q_i^v = q_i$ must be satisfied. Nonetheless, conceptually speaking, if $\sum_v |q_i^v| = |q_i|$, the set of operations correspond to split, which means that a delivery (or a pickup) is divided into several smaller deliveries (or pickups); on the contrary, if $\sum_v |q_i^v| \neq |q_i|$, the set of operations correspond to temporary storage, which means that a delivery (or a pickup) is the result of several deliveries and pickups. In the second case, q_i and each q_i^v do not have the same sign.

Figure 3.2 depicts the optimal solution of instance showed in Figure 3.1 if it is solved as a SD1PDTSP. In this new solution, location 4 is visited twice. Firstly, the vehicle goes from location

2 to location 4 with a load of nine which is not enough to supply the demand of ten units. Secondly, a split delivery is performed; the vehicle delivers all the load in location 4. Thirdly, it visits location 7 picking up five units. Fourthly, it goes back to location 4 with a load of five and as a second delivery, supply the pending demand (i.e., one last unit). Finally, the vehicle leaves location 4 with four units. The total cost of this route is 4759.

In a similar way, Figure 3.3 depicts the optimal solution of a different instance solved as a 1–PDTSP. The total cost of this route 4976. If this instance is solved as a SD1PDTSP, the total cost of the route is 4787 and the solution is shown in Figure 3.4 . In this case, there are two locations visited twice (i.e., locations 18 and 15). Firstly, as in first instance, the demand on location 18 is split. Initially, with a load of 9 units, the vehicle visits 18, it leaves all the load and pickup eight units in location 13. Then, it goes back to 18, complete the supply and it moves to 15 with a load of seven. Although the vehicle arrives in 15 with seven units, an optimal decision is to supply the demand (one unit), store five there and then going to location 2 with a load of one. After location 16 is visited, the vehicle goes back to 15 completely empty and pickup the five units stored previously. The route continues visiting location 10.

3.3 A multi-start evolutionary local search algorithm for the 1–PDTSP

This section presents an MS-ELS to deal with the 1–PDTSP. Firstly, the general structure of the procedure is described. Also this section provides some details about the main components of the proposed algorithm: construction phase based on a greedy randomized procedure, improvement phase designed as a VND and the perturbation operator. This section also briefly describes GRASP and ILS as two particular cases of MS-ELS. It is worth to mention that in spite of the good performance of this algorithms in related routing problems, evolutionary components within these strategies have shown promising results (Prins, 2009).

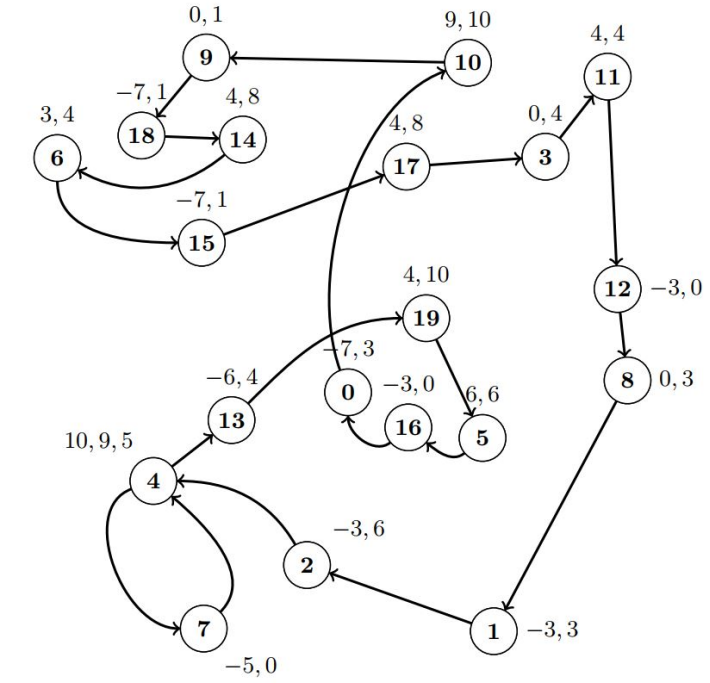


Figure 3.2: Optimal solution for a SD1PDTSP instance with 20 nodes

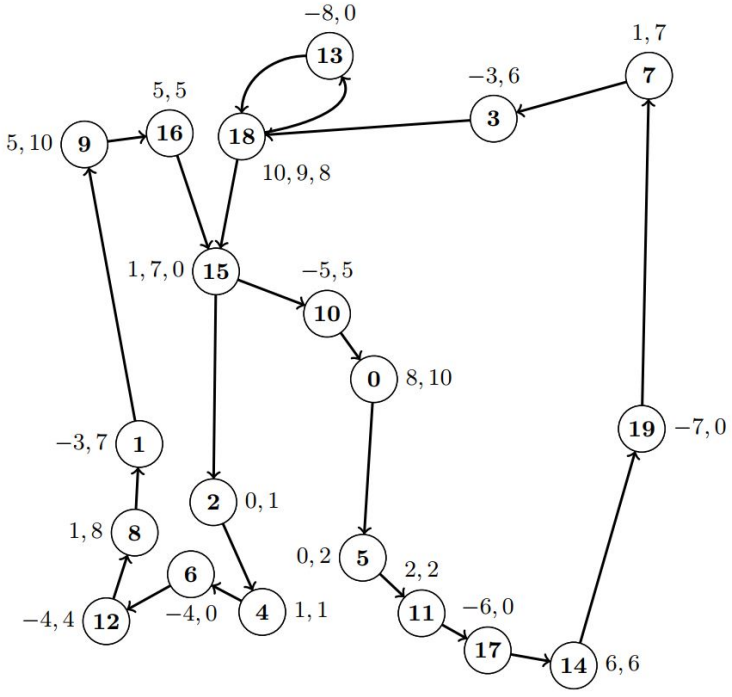


Figure 3.4: Optimal solution for second SD1PDTSP instance with 20 nodes

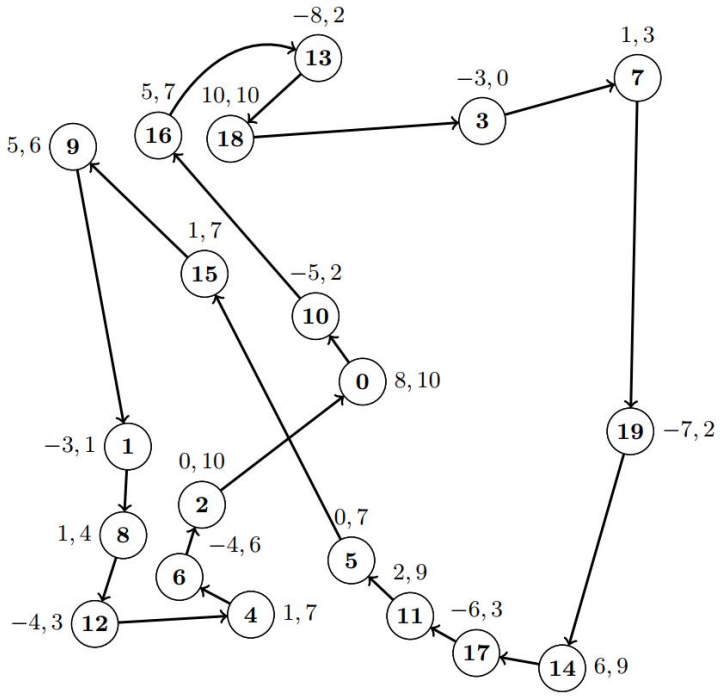


Figure 3.3: Optimal solution for a second 1–PDTSP instance with 20 nodes

Evolutionary local search (ELS) procedure is a metaheuristic framework mainly based on evolutionary strategies and local search algorithm. ELS is initially presented in Wolf and Merz (2007) for the solution of optimization problems in telecommunications. However, routing problems have been also addressed with ELS procedures. Some of these problems are the capacitated VRP (Prins, 2009), the capacitated arc routing problem with split delivery (Belenguer et al., 2010), the truck and trailer routing problem (Villegas et al., 2010) and the multitrip cumulative capacitated vehicle routing problem (Rivera et al., 2013). Figure 3.5, depicts the main steps of a MS-ELS: after each one of several starting solution is build and improved, perturbations are made for a number of iterations and best solutions update the incumbent in each iteration. This process continues until a number of start solutions are reached. MS-ELS can be also seen as a generalization of GRASP and multi-start iterated local search metaheuristics. Firstly, the greedy randomized construction heuristic ensures diversity in the search space. Then, the classical local search in GRASP, is replaced by an ELS which allows to explore in a better way the solution space near to a local optimum before restart a different search from a different starting solution (Duhamel et al., 2011).

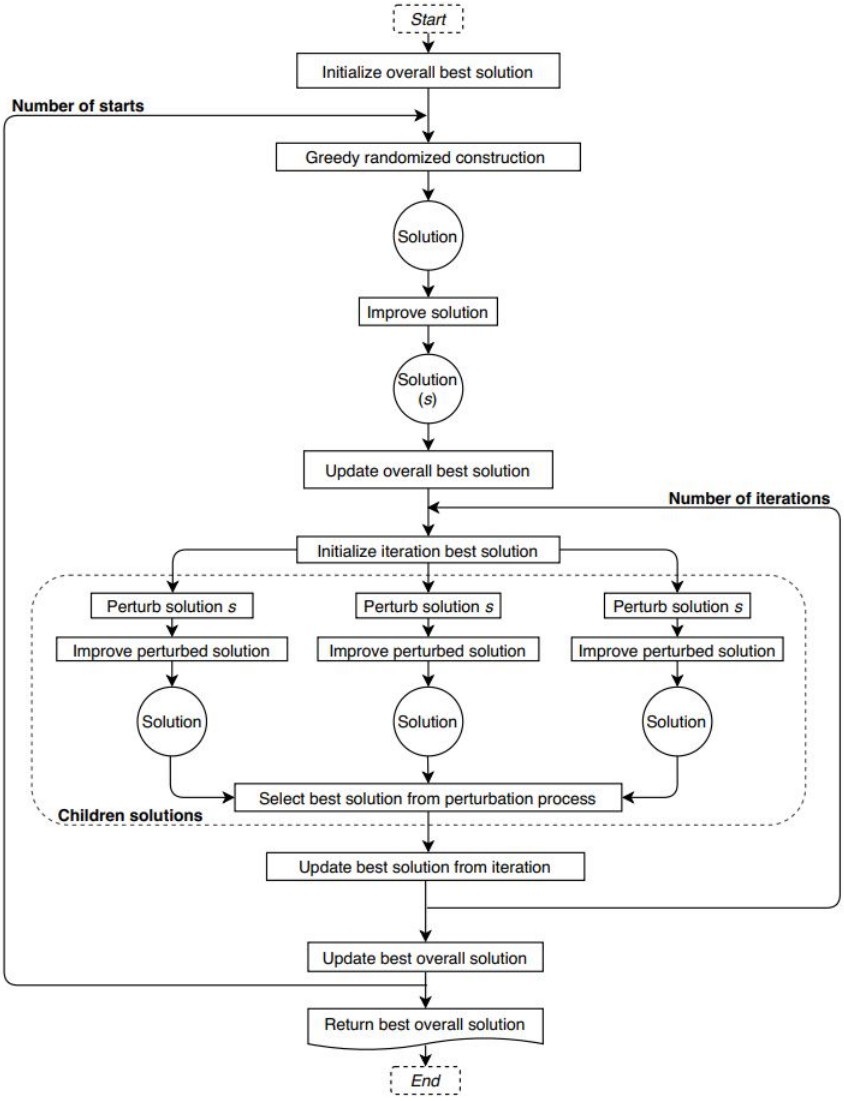


Figure 3.5: Flow chart for MS-ELS algorithm

3.3.1 General framework

As mentioned before, ELS is a metaheuristic based on an evolutionary algorithm and a local search procedure. In an ELS, a single solution is constructed and then, improved via local search. For a number of iterations (`MaxIterations`), the obtained solution is perturbed and next, it is improved `MaxChildren` times. The algorithm returns the best overall solution found. In a multi-start ELS, the ELS is performed `MaxStarts` times. For the sake of clarity, hereinafter, a list of nodes represents a solution. The order of the list denotes the path to follow by the vehicle, obtaining a Hamiltonian cycle. For solution s, an objective function value is computed as sum of the travel costs traversing the arcs between nodes in the path.

The Algorithm 3.1 depicts the MS-ELS for the 1–PDTSP. Starting with an incumbent value for the objective function $\bar{z} = \infty$, `MaxStarts` solutions (s_0) are constructed via greedy randomized algorithm (line 4) and improved using a VND procedure (line 5). If the objective function of resulting solution s is better than the incumbent solution $\bar{s}$, the latter is updated (lines 6 to 9). For each one of the predefined `MaxIterations`, a set of `MaxChildren` perturbation processes are performed on s and immediately, the perturbed resulting solutions s_p are improved calling the VND procedure again (line 14). After each improvement, the algorithm updates the best solution on the current iteration (s') if improved (lines 15-18). Similarly, lines 20 to 23 update the best solution of the current start and lines 25 to 28 update the incumbent solution.

Following sections describe the components of the algorithm depicted in Algorithm 3.1: the construction procedure based on a greedy randomized algorithm, the VND as improvement phase of the MS-ELS, and the perturbation procedure.

3.3.2 Greedy randomized construction

In order to generate several initial solutions, a randomized heuristic based on *nearest neighbor* greedy algorithm is proposed. However, as Hernández-Pérez and Salazar-González (2004b) point out, the nearest neighbor algorithm for the TSP, hardly finds a feasible 1–PDTSP solution if the vehicle capacity is tight. Then, following the procedure in Hernández-Pérez et al. (2009), the traveling costs c_{ij}, are redefined for arcs $(i,j) \in \mathcal{A}$ by penalizing connections between pair of stations of the same type (e.g. two stations with pickup requests). Thus, new values for c_{ij} are stored in $c'_{ij}\ \forall\ (i,j) \in \mathcal{A}$ as follows (Hernández-Pérez et al., 2009):

$$c'_{ij} = \begin{cases} c_{ij} + \frac{(K-Q)\cdot\sum_{(i,j)\in\mathcal{A}} c_{ij}}{10\cdot Q\cdot|\mathcal{N}|}\cdot(2Q-|q_i-q_j|) & \text{if } |q_i+q_j| \leq Q, \\ \infty & \text{otherwise,} \end{cases} \tag{3.14}$$

where K is the sum of units delivered or picked up. As described in Equation (3.10), 1–PDTSP instances are balanced in terms of the demands. Thus, the total units to deliver is equal to the total units to pick up. In order to illustrate the motivation to redefine traveling costs in c_{ij}, Figures 3.6 and 3.7 depict a feasible and an unfeasible solution for a 1–PDTSP instance with seven nodes, respectively. For both solutions and similar to example in Figure 3.1, the left side number near to each node, denotes its demand while the right side number represents the load of the vehicle when entering that node. Solution in Figure 3.6 is a Hamiltonian tour in which a certain amount of units is delivered immediately after a pickup operation is performed. On the contrary, in Figure 3.7, a constructive solution starts at depot (node 0) with an initial

Algorithm 3.1: MS-ELS for the 1–PDTSP: general structure

```
1: function MS-ELS(MaxStarts, MaxIterations, MaxChildren)
2:     z̄ ⟵ ∞, s̄ ⟵ ∅
3:     for i = 1 to MaxStarts do
4:         s_0 ⟵ GreedyRandomizedAlgorithm(seed)
5:         s ⟵ VND(s_0)
6:         if f(s) < z̄ then
7:             s̄ ⟵ s
8:             z̄ ⟵ f(s)
9:         end if
10:        for j = 1 to MaxIterations do
11:            z' ⟵ ∞, s' ⟵ ∅
12:            for p = 1 to MaxChildren do
13:                s_p ⟵ Perturbation(s)
14:                s'' ⟵ VND(s_p)
15:                if f(s'') < z' then
16:                    s' ⟵ s''
17:                    z' ⟵ f(s'')
18:                end if
19:            end for
20:            if f(s) < z' then
21:                s' ⟵ s
22:                z' ⟵ f(s)
23:            end if
24:        end for
25:        if f(s') < z̄ then
26:            s̄ ⟵ s'
27:            z̄ ⟵ f(s')
28:        end if
29:    end for
30:    return s̄
31: end function
```

load of zero units. Then, nodes 2 and 6 are consecutively visited picking up seven and three units, respectively. Note that after leaving node 6 and delivering six units at node 5, the solution become unfeasible since vehicle capacity is violated if any of the non-visited nodes (1, 3 and 4) are reached. Once values in c'_{ij} are computed, the construction phase for a solution s follows:

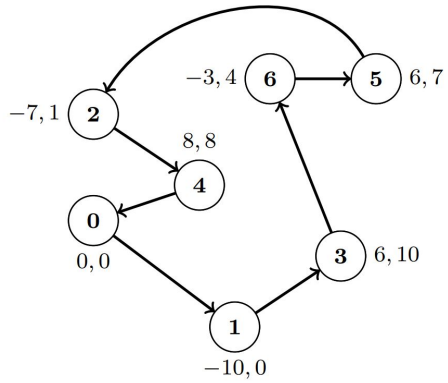


Figure 3.6: Example of a feasible solution for the 1–PDTSP.

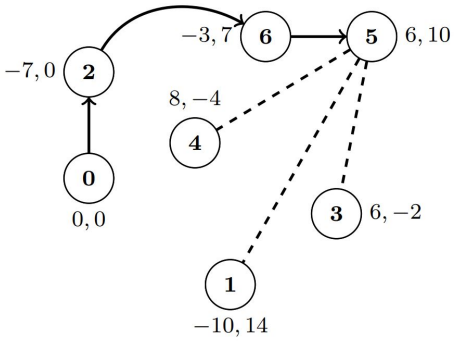


Figure 3.7: Example of an unfeasible solution for the 1–PDTSP

a. Select a node i as the first visited location at random. Let $p \longleftarrow 1$ and $s[p] \longleftarrow i$.

b. Compute ξ as the minimum between a restricted candidate list size (φ) and the number of feasible non-visited nodes. Define $\mathcal{R}$ as the set of the ξ closest and feasible (i.e. the vehicle capacity is not violated) non-visited nodes after the node i is served.

c. If $|\mathcal{R}| > 0$, then choose a node j from $\mathcal{R}$ at random. If $\mathcal{R} = \emptyset$, the constructed solution so far leads to an unfeasible path as in Figure 3.7. Therefore, go to step (a).

d. Let $p \longleftarrow p+1$ and $s[p] \longleftarrow j$. If $p < |\mathcal{N}|$ then define $i \longleftarrow j$ and go to step (b), else stop.

Since each iteration of the constructive phase, requires to include a set of nodes in $\mathcal{R}$, conditions in (3.11) are adapted to check whether a location j is feasible to add in a partial solution with size $p-1$ (Hernández-Pérez and Salazar-González, 2004b):

$$\max_{i=1,..,p-1}\{l_i, l_{p-1} - q_j\} - \min_{i=1,..,p-1}\{l_i, l_{p-1} - q_j\} \leq Q \tag{3.15}$$

The proposed constructive strategy differs from the one presented in Hernández-Pérez et al. (2009). While this new algorithm always end up with a feasible solution, the procedure described in Hernández-Pérez et al. (2009) does not guarantee a feasible result and then, not any repair procedure is required in the proposed solution strategy.

3.3.3 Variable neighborhood descent

VND is a deterministic variant of variable neighborhood search (VNS) which explores sequentially several neighborhoods represented by local search operators (Hansen and Mladenović, 2001; Hansen et al., 2017). Given an incumbent solution s, the set of solutions reachable from s, $N_k(s)$, is explored when the local search operator k ($k \leq k_{max}$) is applied via function `LocalSearch`. If `LocalSearch` function retrieves a solution with a cost smaller than $f(s)$, then s is updated and the search starts again from the first local search operator ($k = 1$). On the contrary, if s is not improved, then `LocalSearch` is performed using the operator $k+1$. The algorithm ends up if no improvement is found throughout the set of k_{max} neighborhoods.

As improvement phase for the MS-ELS, the proposed VND is depicted in Algorithm 3.2. While the general structure of VND described above is mainly stated in lines 4 to 13 of the algorithm, an additional function called `Reverse` (line 21) is used with the aim to explore different regions of the solution space. Function `Reverse` simply changes the orientation of the path in solution s (i.e, the function returns s in the opposite direction). As mentioned in Hernández-Pérez and Salazar-González (2004b), the feasibility of solution s is independent of the orientation of the path. Then, since VND does not handle unfeasible solutions, function `Reverse` always delivers a feasible solution as well. An additional cycle is added to VND algorithm (line 3) just to ensure that after a solution is improved, function `Reverse` is applied. The sequential exploration based on local search operators is performed h times, with $h_{min} \leq h \leq h_{max}$, and each time, after first iteration, the operator `Reverse` is applied. This iterative procedure also may stop before, if no improvement is found after its first call ($b =$`true`).

The sequential exploration based on several neighborhoods with the aim to improve solutions quality within our VND is based on the following two *edge-exchange* (EE) and five *chain-exchange* (CE) operators for the `LocalSearch` function (i.e., $k_{max} = 7$). All operators follow the best improvement rule instead of first improvement criterion.

$2-opt$ and $3-opt$: These neighborhoods are the first (for $k = 1$, $2-opt$) and the last one (for $k = 7$, $3-opt$) in our VND implementation. Hernández-Pérez et al. (2009) prove the good performance of

Algorithm 3.2: Variable neighborhood descent for the 1–PDTSP

```
1:  function VND(s, k_max, h_min, h_max)
2:      s' ⟵ s, h ⟵ 0, b ⟵ False
3:      while (h ≤ h_max and b = False) or (h < h_min) do
4:          k ⟵ 1
5:          while k ≤ k_max do
6:              s'' ⟵ LocalSearch(s', N_k(s'))
7:              if f(s'') < f(s') then
8:                  s' ⟵ s''
9:                  k ⟵ 1
10:             else
11:                 k ⟵ k + 1
12:             end if
13:         end while
14:         if f(s') < f(s) then
15:             s ⟵ s'
16:             b ⟵ False
17:         else
18:             b ⟵ True
19:         end if
20:         if h < h_max or b = False then
21:             s' ⟵ Reverse(s')
22:         end if
23:         h ⟵ h + 1
24:     end while
25:     return s'
26: end function
```

$2-opt$ and $3-opt$ as EE local search operators within a VND to solve the 1–PDTSP. As mentioned before and contrary to Hernández-Pérez et al. (2009), the proposed version of VND does not deal with unfeasible solutions. In this implementation of these operators, the local search procedures find less-cost solutions while unfeasible paths are discarded.

Both neighborhood structures follow the ideas proposed in Lin (1965) to speed up the search process. For each location i, the procedure stores a list of its k nearest neighbors and it also sorts them in increasing order according to travel costs in c_{ij}. Thus, $2-opt$ operator only scans for possible exchanges between each location i and the k closest nodes to i. Similarly, $3-opt$ procedure, evaluates k interchanges for each one of the k nearest neighbors for each location i. Therefore, the complexity of the $2-opt$ and $3-opt$ are $\mathcal{O}(|\mathcal{N}|\cdot k)$ and $\mathcal{O}(|\mathcal{N}|\cdot k^2)$, respectively. Finally, since the number of locations may vary significantly, adequate values for k depend on $|\mathcal{N}|$. Large values for k would lead to a scenario in which $k = |\mathcal{N}|$ and the complexity becomes $\mathcal{O}(|\mathcal{N}|^2)$ and $\mathcal{O}(|\mathcal{N}|^3)$ for

$2-opt$ and $3-opt$, respectively.

$Or-opt(\lambda)$: After $2-opt$ exploration, VND calls $Or-opt(\lambda)$ neighborhoods. $Or-opt$ operators are firstly described in Or (1976), and then mentioned in Babin et al. (2007) as one of the best known CE improvement heuristics for the TSP. It aims to improve a solution by first moving a chain of λ consecutive nodes (i.e. stations) to a different location in the solution path. Moreover, $Or-opt$ heuristic also allows to firstly reverse chains and then move them as described. The proposed VND includes four variations on $Or-opt(\lambda)$ setting $\lambda = \{2,3\}$ and then, reversing chains in both cases. From now, these movements are called $Or-opt(2)$, $Or-opt(3)$, $Or-opt_r(2)$, and $Or-opt_r(3)$, respectively. Thereby, these four combinations are neighborhoods two to five of the VND algorithm ($k = \{2,3,4,5\}$).

Given the structure of this VND and particularly, the use of function `Reverse`, $Or-opt(\lambda)$ operators are adapted only to check for feasible and improvement movements in which chains are moved to previous positions in the path. Therefore, intermediate stations are shifted forward λ positions. The complexity for each one of the four $Or-opt(\lambda)$ movements is $\mathcal{O}(|\mathcal{N}|^2)$.

Move backward: This is the sixth neighborhood within the VND ($k = 6$). This operator attempts to find better solutions by moving a location from its original position to a previous one in the path. This operator can be seen as a special case of our $Or-opt(\lambda)$ movements if $\lambda = 1$ and therefore, the complexity of move backward operator remains as $\mathcal{O}(|\mathcal{N}|^2)$.

Finally, as mentioned before, an order to evaluate neighborhoods within the VND is setting: (*i*) $2-opt$, (*ii*) $Or-opt(2)$, (*iii*) $Or-opt(3)$, (*iv*) $Or-opt_r(2)$, (*v*) $Or-opt_r(3)$, (*vi*) move backward and (*vii*) $3-opt$. This particular order established for the neighborhoods relies on their complexity. As Resende and Ribeiro (2016) point out, and appropriate order can save a significant amount of computation time. Thus, small neighborhoods may be explored first while more complex or large neighborhoods are evaluated later. In the case of this VND, the complexity of all the neighborhoods from (*i*) to (*vi*) is $\mathcal{O}(|\mathcal{N}|^2)$ while $3-opt$ is computed in $\mathcal{O}(|\mathcal{N}|^3)$.

3.3.4 Perturbation

Algorithm 3.3 depicts the proposed perturbation function which is mainly based on the $2-opt$ operator described before in Section 3.3.3. Perturbation function applies a $2-opt$ movement on np sequences of locations from a solution s. For each sequence, i.e. $\sigma_{ij} = (s_i, s_{i+1}, ..., s_{j-1}, s_j)$, the

starting point i is chosen in a random fashion between the first position in s and the position $|\mathcal{N}| - \beta$ where β is the size of the sequence to change (line 4). $2-opt$ operator used in the perturbation function (line 5) only searches the first feasible movement. Thus, perturbation strategy does not check whether the objective function improves and a perturbed solution s_p may end up with a higher value for the objective function (i.e. $f(s_p) > f(s)$). It is worth to mention that large values for np and β would lead to a significant increase in computational time and resulting solution s_p may be significantly different from s. Then, small values are desirable and are shown later in Section 3.4.5. Nonetheless, if no feasible movement is found along the np sequences, the perturbation procedure delivers the initial solution s.

Algorithm 3.3: Perturbation for MS-ELS

```
1: function PERTURBATION (s, np, β)
2:     s_p ⟵ s
3:     for i = 1 to np do
4:         Select at random r ∈ [1, |N| − β]
5:         s_p ⟵ 2-opt(s_p, r, r + β)
6:     end for
7:     return s_p
8: end function
```

3.3.5 Multi-start iterated local search and greedy randomized adaptive search procedure

ILS was introduced by Lourenço et al. (2003) as a hybrid metaheuristic based on a heuristic composed by a constructive algorithm, an improvement strategy as local search and a perturbation function. ILS metaheuristic improves a solution s by calling a local search algorithm. Then, an iterated process is started where the incumbent solution is perturbed and improved by a local search algorithm on each iteration. Solution s is replaced by the local optima solution at the end of each iteration in case of improvement. An ILS with more than one initial solution is called MS-ILS. Using the notation introduced in Section 3.3.1, MS-ILS and ILS can be seen as special cases of MS-ELS in Algorithm 3.1 where `MaxChildren` $= 1$ for MS-ILS, and `MaxStarts` $= 1$ and `MaxChildren` $= 1$ for ILS.

On the other hand, Feo and Resende (1995) define GRASP as a multi-start metaheuristic that consists of two steps: construction and improvement (e.g. local search procedure). While the first one aims to build a solution, the second one finds a local optimum using a local search algorithm. After a fixed number of constructive solutions, GRASP ends and returns the best overall

solution. Note that GRASP can also be described as a special case of MS-ELS metaheuristic in which `MaxStarts` > 1, `MaxIterations` $= 0$ and `MaxChildren` $= 0$ in Algorithm 3.1. In order to show the impact of having these values for parameters `MaxIterations` and `MaxChildren` or those as described previously for MS-ILS, a comparison between GRASP, MS-ILS and MS-ELS is performed in Section 3.4.5. It is worth to mention that our constructive algorithm described in Section 3.3.2 can be seen as greedy, randomized, and adaptive since at each step of the procedure, a node is added to solution in a random fashion after restricted candidate list is updated.

3.4 Computational experiments

The computational experiments presented in this section are based on MILPs described in Section 3.2 and the MS-ELS algorithm depicted in Algorithm 3.1. This section firstly describes the sets of instances used for computational experiments. Then, results solving the 1–PDTSP and SD1PDTSP MILPs are presented. These models were solved via commercial solver (Gurobi 8.1) setting a maximum computation time of 3600 seconds for each instance. Lastly, this section also reports the results obtained via MS-ELS for 1–PDTSP without a fixed maximum computation time. The mathematical models and the metaheuristic algorithm were coded in Visual C++ for Windows 10 running on an Intel Core i7 at 2.70GHz with 8.00 GB of RAM.

3.4.1 Data sets

To test MILPs and MS-ELS performance, two different data sets are solved. The first one is a set of benchmark instances previously reported in the literature. On the other hand, the MS-ELS is also tested on instances generated using data from the operation of EnCicla, the BSS in the Aburrá Valley (Antioquia, Colombia).

1–PDTSP benchmark instances

The benchmark instances are available at `http://hhperez.webs.ull.es/PDsite/`. These instances are classified in two sets. The first one, composed of small number of nodes with $|\mathcal{N}| \in \{20, 30, 40, 50, 60\}$ and the second set with large instances in which $|\mathcal{N}| \in \{100, 200, 300, 400, 500\}$. Node demands vary from -10 to 10 (i.e., $q_i \in [-10, 10]\ \forall\ i \in \mathcal{N}$). Vehicle capacity may vary from ten to 40, being ten the smallest possible value to find feasible solutions and therefore, the hardest configuration to solve (Hernández-Pérez et al., 2009). For each size of the problem, ten instances are available named from A to J. Thus, 100 instances were solved. Hernández-Pérez and

Salazar-González (2004a) and Hernández-Pérez and Salazar-González (2004b) provide a detailed description about the instances generation.

EnCicla BSS instances

The Aburrá Valley (Antioquia, Colombia) is a region located in the south central part of Antioquia department in Colombia. This valley is composed by ten urban areas from north to south: Barbosa, Girardota, Copacabana, Bello, Medellín, Envigado, Itagüí, Sabaneta, La Estrella and Caldas. Since 1980, *Área metropolitana del Valle de Aburrá* (AMVA) is the public entity responsible for planning and management on some common policies of Aburrá Valley territory as transportation, environmental policies, among others.

AMVA as authority on mobility and transportation management is the main sponsor of EnCicla, the public bicycle sharing system in Aburrá Valley. EnCicla began its operation in 2011 with six stations and 105 bikes. Later, in 2013, the system increased the number of stations and bicycles to 13 and 420, respectively. In 2017, 52 stations were available for users and currently, the system is under expansion again and a total of 100 new stations are under construction and 1000 new bikes will be added to the system. Bicycles from EnCicla are available for users from Monday to Friday, starting at 5:30 in the morning to 22:00. On Saturdays, the system operates from 6:30 to 16:00. Nowadays, EnCicla has more than 80 000 active users.

In order to test the performance of 1–PDTSP MILP and MS-ELS using real data from a BSS, a set of instances based on EnCicla were designed. The information required to build the instances was provided by *Subdirección de Movilidad* department (SMD) in AMVA for the operation of EnCicla for 36 days in March and April, 2017 when 52 stations served the system. SMD reports the number of bikes available at the beginning and end of the BSS operation in each station for each one of the 36 days. Thus, it is possible to compute for 35 days, the number of bikes picked up or delivered at each station during the night repositioning operation. To do so, b_i^t and e_i^t are defined as the number of bikes available at station i before starting and after finishing operations at day t, respectively. Then, the number of bikes (q_i^t) to pickup or deliver at station i at day t is calculated as:

$$q_i^t = e_i^{t-1} - b_i^t, \qquad \forall\, i \in \mathcal{N}\backslash\{0\},\ t = 1,..,T \tag{3.16}$$

where T denotes the number of days to analyze (i.e., number of instances to solve). Values for q_0^t are calculated as showed before in equation (3.10) to ensure flow conservation throughout the system. Google maps services were used to get the geographical position for each one of the 52 stations and then, calculate an euclidean distance between each pair (i, j) of them (c_{ij}). Finally,

for this set of instances the vehicle capacity was set to 45 (i.e. the largest capacity for a vehicle in EnCicla fleet).

3.4.2 Results on mixed integer lineal models

1–PDTSP small benchmark instances

Tables 3.1 and 3.2 summarize the computational results when solving the set of 1–PDTSP benchmark instances with MILP$_{PD}$ and MILP$_{SD}$, respectively. These tables report the number of times that each MILP is able to provide an optimal solution for each size of the instances and the three tested values for the vehicle capacity. Average gap (Avg. gap) is computed as $Avg.gap = \frac{UB-LB}{UB}$ where UB and LB are the best solution and the best lower bound reported by the optimizer. The average computational time in seconds required to find the solution (Avg. time) is also included in the tables.

In particular, as Table 3.1 shows, for the 1–PDTSP it is possible to find the optimal solution for all the instances with up to 40 locations. For larger instances with 50 and 60 locations, the number of optimal solutions increases and computation time decreases as the vehicle capacity is extended. This is an expected result since large values of Q leads to a classical TSP. For the largest instances (60 locations) and the smallest value for the vehicle capacity (the hardest case), solutions obtained with MILP$_{PD}$ do not deliver more than 5.64% of gap on average.

When solving MILP$_{SD}$, the optimizer is also able to find the optimal solution for all the instances up to 40 locations (see Table 3.2). Moreover, if split delivery and temporal storage are allowed, the optimal solution is found for an additional instance with 50 locations and the tightest value for Q. Average gaps decrease for hardest instances (with 50 and 60 locations and $Q = 10$) if locations may be visited more than once. In this particular case, the average gap decrease from 5.64% (see Table 3.1) to 2.17%. A similar behavior is obtained for larger values of vehicle capacity. When split delivery and storage are allowed, notice that computational times for small instances are larger when they are compared with the 1–PDTSP case. For example, while MILP$_{PD}$ requires 59.42 seconds, the MILP$_{SD}$ is solved in 120.74 seconds on average for $|\mathcal{N}| = 30$ and $Q = 10$. On the contrary, for instances with more than 30 locations in which split delivery and storage are allowed, computational times are always less than the reported in Table 3.1 for the 1–PDTSP.

As mentioned before, the hardest configuration for the 1–PDTSP arises when vehicle capacity coincides with the largest demand on nodes (i.e., $Q = \max_{i \in \mathcal{N}}\{|q_i|\}$. Then, we present some results on this particular configuration. Tables 3.3 and 3.4 summarize the results obtained when solving the 1–PDTSP benchmark instances using MILP$_{PD}$. Firstly, for the 30 instances with up to 40 nodes, it is possible to find the optimal solution in less than 3600 s. Thus, Table 3.3 reports for each

Table 3.1: MILP$_{PD}$ results for the 1–PDTSP

	$Q = 10$			$Q = 20$			$Q = 40$		
$\|\mathcal{N}\|$	# Optimal solutions	Avg. gap	Avg. time (s)	# Optimal solutions	Avg. gap	Avg. time (s)	# Optimal solutions	Avg. gap	Avg. time (s)
20	10/10	0.00%	3.46	10/10	0.00%	0.49	10/10	0.00%	0.17
30	10/10	0.00%	59.42	10/10	0.00%	61.19	10/10	0.00%	0.96
40	10/10	0.00%	989.57	10/10	0.00%	53.52	10/10	0.00%	1.81
50	5/10	1.73%	2368.12	9/10	0.20%	1133.59	10/10	0.00%	13.80
60	1/10	5.64%	3569.99	8/10	0.38%	1201.49	10/10	0.00%	18.44

Table 3.2: MILP$_{SD}$ results for the SD1PDTSP

	$Q = 10$			$Q = 20$			$Q = 40$		
$\|\mathcal{N}\|$	# Optimal solutions	Avg. gap	Avg. time (s)	# Optimal solutions	Avg. gap	Avg. time (s)	# Optimal solutions	Avg. gap	Avg. time (s)
20	10/10	0.00%	3.82	10/10	0.00%	0.90	10/10	0.00%	0.21
30	10/10	0.00%	120.74	10/10	0.00%	10.84	10/10	0.00%	1.52
40	10/10	0.00%	655.45	10/10	0.00%	25.82	10/10	0.00%	2.54
50	6/10	0.81%	2070.46	9/10	0.12%	821.88	10/10	0.00%	18.85
60	2/10	2.17%	3313.65	8/10	0.23%	1128.40	10/10	0.00%	26.56

Table 3.3: Computational results on MILP for 1–PDTSP instances with $|\mathcal{N}| \leq 40$

$\|\mathcal{N}\| = 20$			$\|\mathcal{N}\| = 30$			$\|\mathcal{N}\| = 40$		
Instance	Opt.	Time (s)	Instance	Opt.	Time (s)	Instance	Opt.	Time (s)
n20q10A	4963	7.48	n30q10A	6403	101.64	n40q10A	7173	1034.12
n20q10B	4976	1.45	n30q10B	6603	9.08	n40q10B	6557	661.14
n20q10C	6333	8.23	n30q10C	6486	80.44	n40q10C	7528	222.30
n20q10D	6280	2.19	n30q10D	6652	21.85	n40q10D	8059	1675.57
n20q10E	6415	4.11	n30q10E	6070	3.30	n40q10E	6928	1574.83
n20q10F	4805	1.86	n30q10F	5737	5.20	n40q10F	7506	1635.10
n20q10G	5119	1.01	n30q10G	9371	187.46	n40q10G	7624	842.43
n20q10H	5594	2.10	n30q10H	6431	6.91	n40q10H	6791	733.63
n20q10I	5130	6.02	n30q10I	5821	21.50	n40q10I	7215	140.12
n20q10J	4410	1.29	n30q10J	6187	25.27	n40q10J	6512	212.75
Avg.		3.57			46.26			873.20

instance, the value for the objective function (column Opt.) and the time in seconds that the solver required to prove optimality. Secondly, for the 20 instances with 50 and 60 nodes, the solver finds the optimal solution in eight cases (i.e. 40% of the instances) as Table 3.4 shows. Columns LB and UB report the lower and upper bound reported by the solver, respectively. Column gap, computed as $(UB - LB)/UB$ shows that this ratio is not greater than 5.50% and 11.70% for 50 and 60 nodes instances, respectively.

Table 3.4: Computational results on MILP for 1–PDTSP instances with $|\mathcal{N}| \in \{50, 60\}$

$\|\mathcal{N}\| = 50$					$\|\mathcal{N}\| = 60$				
Instance	LB	UB	Time (s)	gap	Instance	LB	UB	Time (s)	gap
n50q10A	6987	6987	619.84	0.00%	n60q10A	8190	8653	3600.00	5.35%
n50q10B	9488	9488	2721.35	0.00%	n60q10B	8514	8514	2957.56	0.00%
n50q10C	8913	9110	3600.00	2.16%	n60q10C	9141	9462	3600.00	3.39%
n50q10D	10085	10294	3600.00	2.03%	n60q10D	10650	11243	3600.00	5.27%
n50q10E	9492	9492	2190.77	0.00%	n60q10E	9224	9487	3600.00	2.77%
n50q10F	8398	8887	3600.00	5.50%	n60q10F	8388	9499	3600.00	11.70%
n50q10G	7126	7126	582.89	0.00%	n60q10G	8565	9153	3600.00	6.42%
n50q10H	8545	9019	3600.00	5.26%	n60q10H	8424	8424	2083.92	0.00%
n50q10I	8329	8329	3523.04	0.00%	n60q10I	8869	9524	3600.00	6.88%
n50q10J	8456	8456	249.89	0.00%	n60q10J	8236	9138	3600.00	9.87%
Avg.			2428.78	1.49%				3384.15	5.16%

1–PDTSP large benchmark instances

Results on instances with 100 nodes are presented in Table 3.5. For each one of the ten instances it was possible to find a feasible solution but no optimality proof was delivered by the optimizer. For this size of instances, gap is always less than 30% and around 20% on average. For instances in which $|\mathcal{N}|$ is greater or equal than 200, Table 3.6 reports lower bounds since the solver is not able to find any feasible solution in less than 3600 seconds. Moreover, there are five cases in which the optimizer does not report lower bounds in less than 3600. Nonetheless, it is possible set a naive estimation of this bound (NLB) for instances B, D, F, G and I with 400 nodes as:

$$NLB = \sum_{i \in \mathcal{N}} \min_{j:(i,j) \in \mathcal{A}} \{c_{ij}\} \tag{3.17}$$

EnCicla BSS instances

For the BSS instances from EnCicla operation, Table 3.7 reports the lower bound, upper bound, gap and computational time required to solve each instance. Similarly to 1–PDTSP benchmark instances, the column gap in Table 3.7 is computed as $(UB - LB)/UB$. Thus, the optimizer reports the optimal solution for 28 out of 35 instances obtaining a maximum gap of 5.29% (see instance 6) and also a gap of 0.34% on average. Vehicle capacity and traveling costs are constant for the whole set of 35 instances. Hence, only demand on stations differentiate each instance. Results in Table 3.7 show that objective function (UB) values are very sensitive to changes in stations demand since UB takes values from 2534 to 3440. This variation implies an increase in UB up to 35% with respect to its minimum value. Similarly, changes in units to pick up and deliver at each station have an impact

Table 3.5: Computational results on MILP for 1–PDTSP instances with $|\mathcal{N}| = 100$

Instance	LB	UB	gap
n100q10A	11021	14206	22.42%
n100q10B	12179	15277	20.28%
n100q10C	13118	17506	25.07%
n100q10D	13488	17822	24.32%
n100q10E	10955	13194	16.97%
n100q10F	10900	12736	14.42%
n100q10G	11052	15751	29.83%
n100q10H	11976	13680	12.46%
n100q10I	13072	15141	13.66%
n100q10J	12357	15865	22.11%
Avg.			20.15%

on the computation time required to solve instances. While instance 34 is solved optimally in 4.22 seconds, one hour is not enough to close the gap for the instance 6 (5.29% of gap). This allows to conclude that CPU times required to solve EnCicla instances via commercial solver are also highly sensitive to variations on demands.

3.4.3 Analysis on split delivery, temporal storage and vehicle capacity

Since Table 3.2 evidences that split delivery or temporal storage help to decrease gaps and computing times, it is interesting to analyze how these properties affect the objective function quality. Table 3.8 shows the main results for those instances with an optimal solution delivered by Gurobi when

Table 3.6: Computational results on MILP for 1–PDTSP instances with $|\mathcal{N}| \geq 200$

$\|\mathcal{N}\| = 200$		$\|\mathcal{N}\| = 300$		$\|\mathcal{N}\| = 400$		$\|\mathcal{N}\| = 500$	
Instance	LB	Instance	LB	Instance	LB	Instance	LB
n200q10A	14954	n300q10A	17416	n400q10A	23112	n500q10A	20387
n200q10B	15530	n300q10B	17420	n400q10B	10612 [a]	n500q10B	19259
n200q10C	14010	n300q10C	16374	n400q10C	20939	n500q10C	22456
n200q10D	18033	n300q10D	19443	n400q10D	10517[a]	n500q10D	22155
n200q10E	16215	n300q10E	20776	n400q10E	18021	n500q10E	22144
n200q10F	17999	n300q10F	18779	n400q10F	10282[a]	n500q10F	20735
n200q10G	14883	n300q10G	18151	n400q10G	10348[a]	n500q10G	19314
n200q10H	18032	n300q10H	16606	n400q10H	18463	n500q10H	26866
n200q10I	15245	n300q10I	18456	n400q10I	9885[a]	n500q10I	22348
n200q10J	16273	n300q10J	16929	n400q10J	18879	n500q10J	22335

[a] Naive lower bound (*NLB*) computed as in equation (3.17)

solving MILP$_{PD}$ and MILP$_{SD}$. Table 3.8 reports the number of times that MILP$_{SD}$ delivers a solution with split delivery or storage in at least one location and we compute the average and maximum cost saving (Avg. cost saving and Max. cost saving, respectively) for those instances in which

Table 3.7: Computational results on MILP for EnCicla instances

Instance	LB	UB	gap	Time (s)
1	2737	2737	0.00%	194.43
2	2769	2769	0.00%	115.25
3	2781	2781	0.00%	1315.59
4	2722	2722	0.00%	400.55
5	3074	3074	0.00%	3556.01
6	3258	3440	5.29%	3600.00
7	3002	3068	2.15%	3600.00
8	2746	2746	0.00%	961.45
9	2803	2803	0.00%	1731.30
10	2760	2760	0.00%	69.45
11	2838	2838	0.00%	944.08
12	2723	2723	0.00%	47.54
13	2648	2648	0.00%	19.56
14	2936	2936	0.00%	2223.79
15	2621	2621	0.00%	308.82
16	2793	2793	0.00%	155.91
17	2777	2818	1.45%	3600.00
18	2891	2891	0.00%	3254.71
19	2684	2684	0.00%	58.03
20	2569	2569	0.00%	24.20
21	2791	2817	0.92%	3600.00
22	2859	2859	0.00%	3443.89
23	2794	2794	0.00%	75.66
24	2653	2653	0.00%	35.50
25	2878	2878	0.00%	865.95
26	2595	2595	0.00%	178.14
27	2874	2888	0.48%	3600.00
28	2536	2536	0.00%	5.09
29	2781	2781	0.00%	74.07
30	2574	2574	0.00%	10.44
31	2793	2793	0.00%	2433.94
32	2852	2852	0.00%	819.73
33	2784	2799	0.54%	3600.00
34	2534	2534	0.00%	4.22
35	2752	2782	1.08%	3600.00
Avg.			0.34%	1386.50

$f_{MILP_{SD}} < f_{MILP_{PD}}$. Saving for one instance is computed as follows:

$$\text{cost saving} = \frac{f_{MILP_{PD}} - f_{MILP_{SD}}}{f_{MILP_{SD}}} \times 100\% \tag{3.18}$$

If vehicle capacity is 10, then at least in 80% of the instances (4 out of 5 when $|\mathcal{N}|$=40) is better to visit more than once at least one location. Similarly, for $Q = 10$, average cost improvements vary from 2.59% to 3.58% as the number of locations decreases. For instances with $|\mathcal{N}|$ =20 it is possible to find cost improvements up to 9.10%. As the value for Q increases, the vehicle may not need to visit several times a location if its capacity is large enough. If $Q = 20$, the number of instances in which split delivery or temporal storage is recommended as well as the average and maximum cost savings, decrease. For the largest Q value, allowing several visits to a location do not improve objective function value.

It is worth to mention that values in Table 3.8 are computed with instances in which the optimal solution is found when solving MILP$_{PD}$ and MILP$_{SD}$. Nevertheless, there exits a small subset of instances in which optimal solutions are not reported by the solver but an objective function improvement can be expected if split delivery or storage is allowed. Table 3.9 shows for each one of these instances that the solver provides a final lower bound (LB) for MILP$_{PD}$ greater than the upper bound (UB) found by MILP$_{SD}$. Then, similarly to equation (3.18), a minimum expected cost saving is computed as follows:

$$\text{Min. expected cost saving} = \frac{LB_{MILP_{PD}} - UB_{MILP_{SD}}}{UB_{MILP_{SD}}} \cdot 100\%$$

For the nine instances reported in Table 3.9, a minimum cost improvement equal to 2.06% is expected on average. Moreover, for these instances, final gaps obtained with MILP$_{SD}$, are tighter than those computed when solving MILP$_{PD}$.

Since split delivery or temporal storage improve the quality of objective function, it is worth to analyze briefly how an increase on vehicle capacity reduces the total cost of the route. To do so,

Table 3.8: Split delivery and temporal storage impact on objective function

$\|\mathcal{N}\|$	$Q = 10$			$Q = 20$			$Q = 40$		
	Split or storage	Avg. cost saving	Max. cost saving	Split or-storage	Avg. cost saving	Max. cost saving	Split or storage	Avg. cost saving	Max. cost saving
20	9/10	3.58%	9.10%	0/10	0.00%	0.00%	0/10	0.00%	0.00%
30	9/10	3.01%	5.44%	3/10	0.99%	2.00%	0/10	0.00%	0.00%
40	10/10	2.74%	6.83%	2/10	0.78%	1.44%	0/10	0.00%	0.00%
50	4/5	2.50%	3.47%	4/9	0.71%	1.24%	0/10	0.00%	0.00%
60	1/1	2.59%	2.59%	1/8	0.18%	0.18%	0/10	0.00%	0.00%

only those instances with optimal solution when solving both MILPs are considered. Table 3.10 reports the average cost improvement as a percentage if the vehicle capacity is doubled when Q takes the value of 10 and 20 and a single or multiple visits are allowed to locations. Table 3.10 shows that if only one visit per location is allowed, then an increase on vehicle capacity leads to a higher improvement on the objective function values. As expected, in presence of split delivery or temporal storage, cost saving are lower since the vehicle is able to use shorter arcs in the optimal solution with a tighter capacity.

3.4.4 MILPs benchmark

Up to date, only the B&C strategy described in Salazar-González and Santos-Hernández (2015) deals with the SD1PDTSP. In that article, authors use a subset of the benchmark instances solved to test MILP$_{PD}$ and MILP$_{SD}$. Particulary, when $|\mathcal{N}|$=30. Therefore, MILPs performance is compared with the solution strategy proposed in Salazar-González and Santos-Hernández (2015). Table 3.11 shows the results reported in Salazar-González and Santos-Hernández (2015): lower and upper bounds as well as computational times in seconds for the 1–PDTSP and the SD1PDTSP. In particular, for the 1–PDTSP, while the B&C is able to deliver four out of ten optimal solutions as upper bounds, the MILP$_{PD}$ closes the gap for the whole set of instances. If split deliveries are allowed, while the strategy described in Salazar-González and Santos-Hernández (2015) reports upper bounds equal to the optimal solution for nine out of ten instances, MILP$_{SD}$ reports a 0.00% of gap for all of them. Finally, objective function values and computational times required to solve the tested instances outperform those reported in Salazar-González and Santos-Hernández (2015).

Table 3.9: Expected cost improvement for instances with non-optimal solution

$\|\mathcal{N}\|$	Q	Instance	MILP$_{PD}$			MILP$_{SD}$			Min.expected cost saving
			UB	*LB*	*gap*	*UB*	*LB*	*gap*	
50	10	n50q10C	9221	8896	3.52%	8842	8706	1.54%	0.61%
50	10	n50q10D	10,275	10,052	2.17%	9944	9819	1.26%	1.09%
50	10	n50q10E	9492	9492	0.00%	9238	9154	0.91%	2.75%
50	10	n50q10F	8684	8374	3.57%	7716	7716	0.00%	8.53%
50	10	n50q10I	8329	8227	1.22%	8000	8000	0.00%	2.84%
60	10	n60q10B	8514	8406	1.27%	8384	8313	0.85%	0.26%
60	10	n60q10D	11,112	10,663	4.04%	10,626	10,482	1.36%	0.35%
60	10	n60q10F	9507	8441	11.21%	8335	8025	3.72%	1.27%
60	10	n60q10J	9302	8296	10.81%	8226	8226	0.00%	0.85%
		Average			4.20%			1.07%	2.06%

Table 3.10: Cost improvements for Q variations

$\|\mathcal{N}\|$	From $Q = 10$ to $Q = 20$		From $Q = 20$ to $Q = 40$	
	MILP_{PD}	MILP_{SD}	MILP_{PD}	MILP_{SD}
20	27.50%	23.52%	6.53%	6.53%
30	31.07%	28.02%	8.96%	8.57%
40	30.71%	27.50%	6.95%	6.77%
50	23.78%	21.07%	8.89%	8.54%
60	24.05%	20.95%	6.74%	6.71%
Average	27.42%	24.21%	7.62%	7.43%

Table 3.11: Benchmark results based on Salazar-González and Santos-Hernández (2015)

Instance	Q	1–PDTSP					SD1PDTSP				
		B&C [a]			MILP_{PD}		B&C[a]			MILP_{SD}	
		LB	UB	time (s)	f^*	time (s)	LB	UB	time (s)	f^*	time (s)
n30q10A	10	5724.8	6727	1253.9	6403	111.96	5654.1	6256	2266.4	6256	55.65
n30q10A	10	6193.2	6603	165.8	6603	14.65	5478.6	6603	591.1	6603	168.51
n30q10C	10	5215.5	6486	1197.9	6486	60.89	5149.3	6348	2278.5	6348	281.18
n30q10D	10	5450.0	6652	2698.4	6652	19.34	5279.2	6380	3811.8	6380	12.13
n30q10E	10	5691.2	6070	588.1	6070	4.25	5402.5	6052	942.6	6052	18.35
n30q10F	10	5392.5	5737	600.4	5737	7.48	5225	5727	1008.4	5727	18.99
n30q10G	10	8705.8	9371	2135.1	9371	311.63	8641.7	9005	3994.7	9005	549.31
n30q10H	10	5191.5	6433	457.9	6431	8.52	5020.4	6164	4598.6	6164	15.85
n30q10I	10	5156.2	5864	310.4	5821	28.14	4969.5	5596	4839.8	5596	71.96
n30q10J	10	5865.7	6192	224.8	6187	27.35	5601.4	6090	1712.5	5868	15.42

[a] Salazar-González and Santos-Hernández (2015)

3.4.5 Results on MS-ELS

This section presents the results based on the MS-ELS strategy. As mentioned before, as MS-ELS is a generalization of GRASP and MS-ILS, variations on values of `parametersMaxStarts`, `MaxIterations` and `MaxChildren` are performed to run MS-ILS and GRASP. Moreover, since running times of MS-ELS, MS-ILS and GRASP are roughly proportional to the number of calls to the VND (Rivera et al., 2013), experiments run with a number of 300 calls to control the execution time and fairly compare the algorithms performance. After testing several configurations for `MaxStarts`, `MaxIterations`, and `MaxChildren` within MS-ELS and MS-ILS, best results on average with values are reported in Table 3.12. For GRASP, `MaxIterations` and `MaxChildren` are always set to zero. Table 3.12 also shows the final values for parameters required in the construction phase, VND scheme, $2-opt$ and $3-opt$ operators and perturbation function.

Table 3.13 presents the computational results for the set of small instances (i.e. $|\mathcal{N}| \leq 60$).

Table 3.12: Parameter values for solution strategy

Solution framework			
	MS-ELS	MS-ILS	GRASP
MaxStarts	5	15	300
MaxIterations	12	20	0
MaxChildren	5	1	0
Greedy randomized construction			
φ	10		
VND			
h_{min}	1		
h_{max}	2		
$2-opt$ and $3-opt$			
k	$2 \cdot \sqrt{\lvert\mathcal{N}\rvert}$		
Perturbation			
np	4		
β	6		

This table shows the name of the instance and column Opt reports the value of the objective function for the optimal solution; these values were retrieved from results in Section 3.4.2 and from Hernández-Pérez et al. (2009). For each instance, Table 3.13 compares five different methods for the 1–PDTSP and BRP: the hybrid GRASP/VND in Hernández-Pérez et al. (2009), the GA in Zhao et al. (2009) and the three proposed strategies, GRASP, MS-ILS and MS-ELS. For each strategy, columns # opt. show the number of times the optimal solution was found over ten runs (for the GRASP/VND this information is not available in Hernández-Pérez et al. (2009)). Columns Avg. display the average value for the objective function and columns gap (%) show the difference between the average objective function retrieved (z_{Avg}) and the optimal solution reported in columns Opt (z^*). This gap is computed as follows:

$$gap(\%) = \frac{z_{Avg} - z^*}{z^*} \times 100 \tag{3.19}$$

For instances in which $|\mathcal{N}| = 20$, the five strategies deliver the optimal solution in each one of the ten runs. Similarly, GA and the proposed solution algorithms are able to provide the optimal solution for instances with 30 nodes in all runs. For instances with $|\mathcal{N}| = \{40, 50, 60\}$, gaps delivered by MS-ELS are smaller than those reported by the other four methods. Moreover, the number

Table 3.13: Computational results for small instances

		GRASP/VND[a]		GA[b]			GRASP			MS-ILS			MS-ELS		
Instance	Opt[c]	Avg.	gap (%)	# opt	Avg.	gap (%)	# opt	Avg.	gap (%)	# opt	Avg.	gap (%)	# opt	Avg.	gap (%)
$\|\mathcal{N}\|$=20															
n20q10A	4963	4963.0	0.00	10	4963.0	0.00	10	4963.0	0.00	10	4963.0	0.00	10	4963.0	0.00
n20q10B	4976	4976.0	0.00	10	4976.0	0.00	10	4976.0	0.00	10	4976.0	0.00	10	4976.0	0.00
n20q10C	6333	6333.0	0.00	10	6333.0	0.00	10	6333.0	0.00	10	6333.0	0.00	10	6333.0	0.00
n20q10D	6280	6280.0	0.00	10	6280.0	0.00	10	6280.0	0.00	10	6280.0	0.00	10	6280.0	0.00
n20q10E	6415	6415.0	0.00	10	6415.0	0.00	10	6415.0	0.00	10	6415.0	0.00	10	6415.0	0.00
n20q10F	4805	4805.0	0.00	10	4805.0	0.00	10	4805.0	0.00	10	4805.0	0.00	10	4805.0	0.00
n20q10G	5119	5119.0	0.00	10	5119.0	0.00	10	5119.0	0.00	10	5119.0	0.00	10	5119.0	0.00
n20q10H	5594	5594.0	0.00	10	5594.0	0.00	10	5594.0	0.00	10	5594.0	0.00	10	5594.0	0.00
n20q10I	5130	5130.0	0.00	10	5130.0	0.00	10	5130.0	0.00	10	5130.0	0.00	10	5130.0	0.00
n20q10J	4410	4410.0	0.00	10	4410.0	0.00	10	4410.0	0.00	10	4410.0	0.00	10	4410.0	0.00
Average			0.00			0.00			0.00			0.00			0.00
$\|\mathcal{N}\|$=30															
n30q10A	6403	6406.8	0.06	10	6403.0	0.00	10	6403.0	0.00	10	6403.0	0.00	10	6403.0	0.00
n30q10B	6603	6603.0	0.00	10	6603.0	0.00	10	6603.0	0.00	10	6603.0	0.00	10	6603.0	0.00
n30q10C	6486	6486.0	0.00	10	6486.0	0.00	10	6486.0	0.00	10	6486.0	0.00	10	6486.0	0.00
n30q10D	6652	6655.1	0.05	10	6652.0	0.00	10	6652.0	0.00	10	6652.0	0.00	10	6652.0	0.00
n30q10E	6070	6070.0	0.00	10	6070.0	0.00	10	6070.0	0.00	10	6070.0	0.00	10	6070.0	0.00
n30q10F	5737	5737.0	0.00	10	5737.0	0.00	10	5737.0	0.00	10	5737.0	0.00	10	5737.0	0.00
n30q10G	9371	9371.0	0.00	10	9371.0	0.00	10	9371.0	0.00	10	9371.0	0.00	10	9371.0	0.00
n30q10H	6431	6431.2	0.00	10	6431.0	0.00	10	6431.0	0.00	10	6431.0	0.00	10	6431.0	0.00
n30q10I	5821	5821.0	0.00	10	5821.0	0.00	10	5821.0	0.00	10	5821.0	0.00	10	5821.0	0.00
n30q10J	6187	6187.4	0.01	10	6187.0	0.00	10	6187.0	0.00	10	6187.0	0.00	10	6187.0	0.00
Average			0.01			0.00			0.00			0.00			0.00
$\|\mathcal{N}\|$=40															
n40q10A	7173	7188.5	0.22	8	7179.0	0.08	9	7175.4	0.03	8	7182.7	0.14	10	7173.0	0.00
n40q10B	6557	6568.5	0.18	5	6564.5	0.11	4	6566.4	0.14	6	6563.8	0.10	7	6561.5	0.07
n40q10C	7528	7528.4	0.01	10	7528.0	0.00	5	7529.6	0.02	7	7528.6	0.01	10	7528.0	0.00
n40q10D	8059	8135.6	0.95	8	8075.4	0.20	4	8097.8	0.48	5	8095.0	0.45	10	8059.0	0.00
n40q10E	6928	6959.3	0.45	10	6928.0	0.00	3	6941.2	0.19	3	6931.2	0.05	10	6928.0	0.00
n40q10F	7506	7590.5	1.13	10	7506.0	0.00	4	7544.1	0.51	5	7538.3	0.43	10	7506.0	0.00
n40q10G	7624	7682.8	0.77	10	7624.0	0.00	3	7660.6	0.48	6	7645.3	0.28	10	7624.0	0.00
n40q10H	6791	6795.7	0.07	10	6791.0	0.00	5	6801.0	0.15	9	6793.7	0.04	10	6791.0	0.00
n40q10I	7215	7219.0	0.06	8	7215.2	0.00	4	7220.9	0.08	6	7217.5	0.03	10	7215.0	0.00
n40q10J	6512	6513.3	0.02	10	6512.0	0.00	10	6512.0	0.00	10	6512.0	0.00	10	6512.0	0.00
Average			0.38			0.04			0.21			0.15			0.01
$\|\mathcal{N}\|$=50															
n50q10A	6987	6996.7	0.14	10	6987.0	0.00	1	6993.7	0.10	6	6989.6	0.04	10	6987.0	0.00
n50q10B	9488	9512.6	0.26	8	9501.8	0.15	2	9513.0	0.26	3	9497.9	0.10	10	9488.0	0.00
n50q10C	9110	9133.7	0.26	1	9119.5	0.10	1	9128.2	0.20	2	9143.9	0.37	5	9113.5	0.04
n50q10D	10260	10464.3	1.99	2	10354.8	0.92	1	10458.1	1.93	3	10428.5	1.64	5	10265.4	0.05
n50q10E	9492	9625.1	1.40	7	9574.5	0.87	1	9656.0	1.73	3	9589.3	1.03	10	9492.0	0.00
n50q10F	8684	8773.2	1.03	8	8692.5	0.10	1	8741.4	0.66	4	8735.4	0.59	10	8684.0	0.00
n50q10G	7126	7217.4	1.28	9	7133.5	0.11	1	7230.5	1.47	4	7178.9	0.74	10	7126.0	0.00
n50q10H	8885	9006.5	1.37	1	8956.9	0.81	2	9007.9	1.38	1	9062.1	1.99	3	8893.2	0.09
n50q10I	8329	8412.5	1.00	7	8357.5	0.34	1	8422.9	1.13	3	8391.2	0.75	3	8344.1	0.18
n50q10J	8456	8666.1	2.48	1	8475.8	0.23	0	8595.1	1.64	2	8529.3	0.87	10	8456.0	0.00
Average			1.12			0.36			1.05			0.81			0.04
$\|\mathcal{N}\|$=60															
n60q10A	8602	8726.6	1.45	5	8634.8	0.38	1	8719.9	1.37	1	8709.3	1.25	10	8602.0	0.00
n60q10B	8514	8683.2	1.99	10	8514.0	0.00	2	8604.8	1.07	2	8593.9	0.94	10	8514.0	0.00
n60q10C	9453	9565.6	1.19	3	9485.5	0.34	1	9582.8	1.37	1	9585.3	1.40	1	9479.8	0.28
n60q10D	11059	11320.6	2.37	1	11140.2	0.73	1	11282.5	2.02	1	11296.0	2.14	1	11121.6	0.57
n60q10E	9487	9724.8	2.51	1	9592.1	1.11	0	9614.7	1.35	1	9615.0	1.35	5	9494.7	0.08
n60q10F	9063	9437.2	4.13	1	9192.2	1.43	1	9292.2	2.53	1	9221.3	1.75	3	9115.6	0.58
n60q10G	8912	9107.9	2.20	1	8996.0	0.94	1	9082.0	1.91	1	9059.9	1.66	1	8955.8	0.49
n60q10H	8424	8467.3	0.51	3	8472.3	0.57	1	8460.2	0.43	1	8458.4	0.41	10	8424.0	0.00
n60q10I	9394	9529.6	1.44	1	9505.8	1.19	2	9502.2	1.15	1	9519.5	1.34	1	9452.9	0.63
n60q10J	8750	8956.5	2.36	1	8803.3	0.61	1	8894.1	1.65	1	8913.6	1.87	3	8788.7	0.44
Average			2.01			0.73			1.48			1.41			0.31

[a] Results taken from Hernández-Pérez et al. (2009)
[b] Results taken from Zhao et al. (2009)
[c] Results taken from Hernández-Pérez et al. (2009)

of times that MS-ELS finds the optimal solution is greater or equal that the number of optimal values obtained with GRASP and MS-ILS, except for instance n60q10I. Similarly, there exist improvements on average solutions if the evolutionary component is added to GRASP and ILS algorithms. Quality on solutions found by GA (Zhao et al., 2009) is higher than the one delivered by GRASP/VND from Hernández-Pérez et al. (2009) and proposed GRASP and MS-ILS. However, MS-ELS outperforms GA and GRASP/VND algorithms. Note also that proposed GRASP is able to find better solutions on average than the hybrid GRASP/VND from Hernández-Pérez et al. (2009). This result allows to conclude that $Or-opt(\lambda)$ operators and function `Reverse` within the VND help significantly finding better local optimum solutions for small instances. Hernández-Pérez et al. (2009) define $k = 4 \cdot \sqrt{|\mathcal{N}|}$ for $2-opt$ and $3-opt$ neighborhood size while MS-ELS runs with a fixed value for $k = 2 \cdot \sqrt{|\mathcal{N}|}$ and therefore, less computational effort is required for these local search operators.

Table 3.14 summarizes the results on large instances. Due to the outperforming behavior of MS-ELS over GRASP and MS-ILS for the smaller instances, this table only compares the evolutionary local search strategy with the GRASP/VND (Hernández-Pérez et al., 2009) and GA (Zhao et al., 2009). For each one of the strategies, columns Best report the minimum value for objective function found by the algorithm. It is worth to mention that optimal solutions have not been reported in the literature for these instances. Columns Avg. and Std. dev. show the average and standard deviation for objective function. Standard deviation values are not available in Hernández-Pérez et al. (2009) for the GRASP/VND. Finally, Table 3.14 also presents the differences on solution quality between our solution strategy and the GRASP/VND and GA in columns gap GRASP/VND and gap GA, respectively. As example, gap GA is computed as follows:

$$gapGA(\%) = \frac{Best_{MS-ELS} - Best_{GA}}{Best_{GA}} \times 100 \tag{3.20}$$

where $Best_{MS-ELS}$ and $Best_{GA}$ are the best solution found by MS-ELS and GA, respectively. Gap GRASP/VND is computed in a similar way. Lastly, Table 3.14 also shows column gap LB in which the lower bound (LB) obtained vian MILP and $Best_{MS-ELS}$ value are compared:

$$gapLB(\%) = \frac{Best_{MS-ELS} - LB}{Best_{MS-ELS}} \times 100 \tag{3.21}$$

The proposed evolutionary local search strategy is able to find better solutions than GA for 46 out of 50 large instances (i.e. 92% of the instances) and outperforms GRASP/VND in all cases. Standard deviation for the MS-ELS over ten runs is less than the reported in Zhao et al. (2009) for the GA, except for instances in which $|\mathcal{N}| = 300$. Nevertheless, for this ten instances, MS-ELS

Table 3.14: Computational results for large instances

	GRASP/VND[a]		GA[b]			MS-ELS					
	Best	Avg.	Best	Avg.	Std. dev.	Best	Avg.	Std. dev.	gap GRASP/VND	gap GA	gap LB
$\|\mathcal{N}\|$ =100											
n100q10A	11874	12087.6	11828	11922.6	71.3	**11760**	11834.7	41.5	-0.96%	-0.57%	6.28%
n100q10B	13172	13582.6	13114	13301.6	157.1	**12938**	13084.1	65.0	-1.78%	-1.34%	5.87%
n100q10C	14063	14421.3	13977	14095.2	147.2	**13958**	13991.2	24.5	-0.75%	-0.14%	6.02%
n100q10D	14490	14787.5	**14253**	14406.4	111.9	14297	14407.7	69.7	-1.33%	0.31%	5.66%
n100q10E	11546	12502.6	**11411**	11436.4	52.4	11419	11503.2	69.9	-1.10%	0.07%	4.06%
n100q10F	11734	12010.7	11644	11699	34.5	**11613**	11732.5	50.2	-1.03%	-0.27%	6.14%
n100q10G	12049	12366.9	12038	12120.2	104.8	**11889**	11972.3	39.9	-1.33%	-1.24%	7.04%
n100q10H	12892	13169.2	12818	12906.2	125.1	**12742**	12799.1	30.8	-1.16%	-0.59%	6.01%
n100q10I	14048	14390.2	14032	14137.2	95.9	**13799**	13918.6	63.5	-1.77%	-1.66%	5.27%
n100q10J	13430	13737.6	13297	13516.8	216.4	**13240**	13402.9	73.5	-1.41%	-0.43%	6.67%
Average					111.7			52.8	-1.26%	-0.59%	5.90%
$\|\mathcal{N}\|$ =200											
n200q10A	18013	18564	17686	17987	201.9	**17642**	17749.1	57.0	-2.06%	-0.25%	15.24%
n200q10B	18154	18932.5	17798	18069.4	243.1	**17393**	17888.2	232.5	-4.19%	-2.28%	10.71%
n200q10C	16969	17280.3	16466	16751.2	245.8	**16430**	16563.3	113.5	-3.18%	-0.22%	14.73%
n200q10D	21565	22285.7	21306	21564.4	207.3	**21244**	21520.6	153.0	-1.49%	-0.29%	15.11%
n200q10E	19913	20643.2	**19299**	19713	358.9	19422	19582.8	117.8	-2.47%	0.64%	16.51%
n200q10F	21949	22284.6	21910	22144	247.7	**21536**	21649.2	101.9	-1.88%	-1.71%	16.42%
n200q10G	17956	18627.7	17712	17797.8	80.6	**17564**	17721.3	107.4	-2.18%	-0.84%	15.26%
n200q10H	21463	22084.9	21276	21584	278.4	**19921**	21219.3	480.9	-7.18%	-6.37%	9.48%
n200q10I	18606	19184.8	18380	18509.8	149.6	**18068**	18415.2	213.9	-2.89%	-1.70%	15.62%
n200q10J	19273	19839.5	18970	19274.2	205.5	**18763**	19224.1	208.8	-2.65%	-1.09%	13.27%
Average					221.9			178.7	-3.02%	-1.41%	14.24%
$\|\mathcal{N}\|$ =300											
n300q10A	23244	24052.9	23242	23592	265.1	**22973**	23172.6	176.5	-1.17%	-1.16%	24.19%
n300q10B	23187	23845.6	22934	23028.6	114.9	**22779**	23011.6	145.9	-1.76%	-0.68%	23.53%
n300q10C	21800	22516.6	21922	22083.4	189.6	**21029**	21774.6	429.8	-3.54%	-4.07%	22.14%
n300q10D	25971	26462.1	25883	26289.8	253.5	**25448**	25664.3	186.8	-2.01%	-1.68%	23.60%
n300q10E	27420	27892.1	27367	27923.8	358.5	**26412**	26994.3	446.4	-3.68%	-3.49%	21.34%
n300q10F	24852	25278.2	24826	25055.4	171.8	**23507**	24391.3	589.8	-5.41%	-5.31%	20.11%
n300q10G	24308	24760.5	23868	24300.6	412.0	**23632**	23907.8	184.5	-2.78%	-0.99%	23.19%
n300q10H	22684	23116.5	21625	21965	278.5	**21513**	22031.6	289.6	-5.16%	-0.52%	22.81%
n300q10I	24633	25492.6	24513	24959.2	330.1	**24112**	24697.1	391.0	-2.12%	-1.64%	23.46%
n300q10J	23086	23530.2	22810	23045	351.1	**22796**	23097.3	201.4	-1.26%	-0.06%	25.74%
Average					272.5			304.2	-2.89%	-1.96%	23.01%
$\|\mathcal{N}\|$ =400											
n400q10A	31486	31912	31678	31964.4	309.9	**30522**	30977.2	257.7	-3.06%	-3.65%	24.28%
n400q10B	24883	25606.4	24262	24752.4	283.2	**24226**	24711.4	196.5	-2.64%	-0.15%	56.20%
n400q10C	28942	29463.2	28741	29287.4	603.6	**28405**	28580.0	121.6	-1.86%	-1.17%	26.28%
n400q10D	24597	25308.6	24508	24794.8	320.1	**23604**	24223.8	306.3	-4.04%	-3.69%	55.44%
n400q10E	25548	26120	25071	25473	276.4	**24497**	25087.4	330.1	-4.11%	-2.29%	26.44%
n400q10F	27169	27755.1	26681	27362.8	411.6	**26409**	26959.8	315.2	-2.80%	-1.02%	61.07%
n400q10G	24626	25088.4	**23891**	24290.4	273.0	24052	24287.6	159.2	-2.33%	0.67%	56.98%
n400q10H	26030	26468.8	25348	25811.4	351.5	**25245**	25496.9	194.2	-3.02%	-0.41%	26.86%
n400q10I	28992	29596.6	28714	29261.6	488.7	**28172**	28659.3	272.6	-2.83%	-1.89%	64.91%
n400q10J	26204	26916.2	26010	26489.4	281.6	**25494**	25711.3	191.9	-2.71%	-1.98%	25.95%
Average					360.0			234.5	-2.94%	-1.56%	42.44%
$\|\mathcal{N}\|$ =500											
n500q10A	28742	29323.6	28857	29258.8	478.3	**27923**	28218.9	265.1	-2.85%	-3.24%	26.99%
n500q10B	27335	27711.1	26648	27454.8	525.7	**26309**	26596.6	181.5	-3.75%	-1.27%	26.80%
n500q10C	31108	31692.7	30701	31426.8	609.4	**29787**	30469.1	336.6	-4.25%	-2.98%	24.61%
n500q10D	30794	31428.4	30994	31442.2	376.9	**30017**	30200.1	231.6	-2.52%	-3.15%	26.19%
n500q10E	30674	31371.7	30905	31154.6	231.3	**28731**	29897.1	497.7	-6.33%	-7.03%	22.93%
n500q10F	28957	29812.3	28882	29241	244.9	**28112**	28540.6	353.1	-2.92%	-2.67%	26.24%
n500q10G	27198	27958.2	27107	27473	212.5	**26519**	26738.8	197.0	-2.50%	-2.17%	27.17%
n500q10H	36857	37361.1	37626	38142.4	258.8	**35855**	36154.2	223.5	-2.72%	-4.71%	25.07%
n500q10I	31045	31536	30796	31044.6	306.0	**29713**	30239.3	293.7	-4.29%	-3.52%	24.79%
n500q10J	31412	31877.9	31255	32310	617.9	**30028**	30618.1	328.2	-4.41%	-3.93%	25.62%
Average					386.2			290.8	-3.65%	-3.47%	25.64%

[a]Results taken from Hernández-Pérez et al. (2009)
[b]Results taken from Zhao et al. (2009)

delivers better solutions than the genetic algorithm. Moreover, it is worth to comment that as the number of nodes increases, the average differences between MS-ELS and the other algorithms (i.e. gap GRASP/VND and gap GA), also increase. This is, the solution quality of MS-ELS strategy on average, becomes higher than the one provided by other methods if $|\mathcal{N}|$ is larger. It can be seen also, if the average solution provided by the MS-ELS is compared with the best solution delivered by GRASP/VND and GA. Table 3.15, presents the percentage of instances in which average objective function value found via MS-ELS is less than the best solution reported using other two strategies. For the largest subset of instances ($|\mathcal{N}| = 500$) average solutions from MS-ELS outperform the best solution found via GRASP/VND and GA.

Regarding computational times, MS-ELS performance is compared with GRASP/VND and GA. Table 3.16 presents the average CPU time (in seconds) required to solve each subset of large instances. This table also shows the time needed to construct solutions as well as the time that the MS-ELS requires to improve solutions via VND. In general, MS-ELS requires more computational effort to find solutions with a better solution quality. Nevertheless, the proposed strategy is able to find on average a similar solution quality within computational times as those reported in Hernández-Pérez et al. (2009) and Zhao et al. (2009). Fig. 3.8 depicts for instance n100q10A, the evolution of the objective function value for the best solution found. Values for objective function solving the 1–PDTSP via MS-ELS are similar for GRASP/VND and GA when computational times coincide with those reported in Table 3.16 for instances with 100 nodes. In addition, MS-ELS continues improving solution until the end of time horizon (28 seconds), which indicates that it can find even better solutions if more time is given.

Table 3.17 summarizes the results obtained on EnCicla instances solving the 1–PDTSP via MS-ELS. This table shows a comparison between the proposed metaheuristic strategy and the exact approach based on MILP in Section 3.2. For each instance, column Best reports the minimum value found by the MS-ELS for the objective function as well as the number of times this solution is delivered after 10 runs of the MS-ELS (column # best). Then, the value in column Best is compared with the lower bound reported by the optimizer after the MILP is solved (see Table 3.7). Table 3.17 reports whether the best solution found is the optimal. Average for the objective function value over the ten runs is also computed (column Avg). Since in Table 3.7, a lower and upper bound (LB and

Table 3.15: Average solution for MS-ELS vs. best solutions for GRASP and GA

	100	200	300	400	500
GRASP/VND	100%	100%	80%	100%	100%
GA	40%	20%	50%	50%	100%

Table 3.16: CPU times (s) comparison for large instances

$\|\mathcal{N}\|$	GRASP/VND[a]	GA[b]	MS-ELS		
			Construction	VND	Total
100	8.85	21.12	0.81	27.21	28.02
200	41.76	95.23	3.54	110.75	114.29
300	117.86	212.59	9.82	279.14	288.96
400	220.4	358.22	24.97	650.01	674.98
500	391.47	570.15	39.67	927.96	967.63

[a]Results taken from Hernández-Pérez et al. (2009)
[b]Results taken from Zhao et al. (2009)

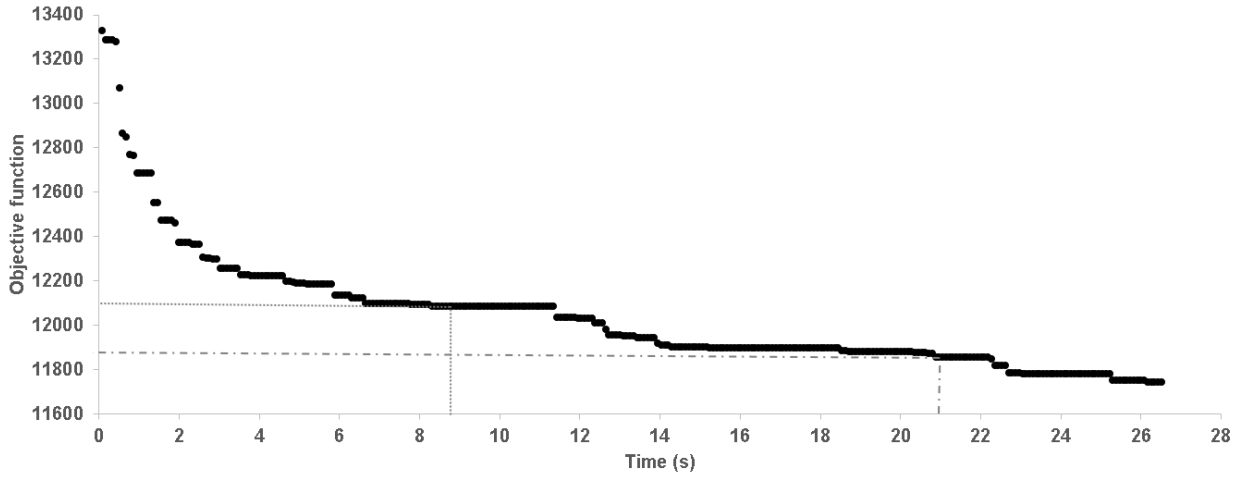


Figure 3.8: Objective function value versus time (s) for instance n100q10A

UB, respectively) are available, gaps between the best solution delivered by the MS-ELS and these bounds are reported. Thus, columns Min. gap in Table 3.17 are computed as follows: Min. $\text{gap}_{LB} = Best - LB/Best$ and Min. $\text{gap}_{UB} = Best - UB/Best$, regarding lower and upper bound values from MILP, respectively. Average gaps over the ten runs are also reported in columns Avg. gap. Finally, the average CPU time required to solve each instance via MS-ELS is also showed.

Note that MS-ELS is able to find the optimal solution for 28 out of the 35 EnCicla instances. For the remaining seven instances, in which MS-ELS do not deliver the optimal value for the objective function, the metaheuristic finds at least the same upper bound reported by the optimizer when MILP is solved. For instances 6, 7 and 35, the MS-ELS improves the upper bound found via commercial solver when the MILP is solved. In these cases, column Min. gap_{UB} shows negative values. Lastly, MS-ELS is able to find minimum gap of 0.27% on average, when solutions are compared with LB values.

There exist limitations if a real-scenario repositioning operation is treated as a 1–PDTSP. These limitations are derived from two main facts to remark. Firstly, in the static version of the BRP, stations are served during the night or while bike demands are negligible. Secondly, in the 1–PDTSP, a single vehicle is available to visit all the locations. Then, only small size BSS can be served since services times (load and unload bikes) and traveling times between stations are also included in the operation for one vehicle. The main motivation to study EnCicla instances is not related to test the 1–PDTSP in large bike repositioning operation instances. The purpose within these experiments is to evaluate the performance and solution quality of the solution strategy on a different set of instances in which demands, and node locations were not generated randomly. In real scenarios, for BSS with a large number of stations, the repositioning operation using a single vehicle would lead to non-practical problems.

When solving MILP for EnCicla instances (see Table 3.7) it is possible to find optimal solutions for 28 out of 35 instances in less than one hour and an average gap of 0.34%. These results evidence an adequate performance for the exact strategy. However, it is possible to conduct an additional experiment for this subset of instances in order to reduce the CPU time required and improve solution quality for instances with no optimality proof. This experiment consist on using the solution delivered by MS-ELS as a warm start for MILP_{PD}. This warm start (the final solution of MS-ELS), is read by the solver as an upper bound once the *branch-and-bound* process begins. Table 3.18 summarizes the main results obtained with this solution strategy. For each instance, Table 3.18 reports the upper and lower bound from the optimizer (see columns UB and LB, respectively). Thus, the gap is computed as $(UB-LB)/UB \times 100$. The CPU time required in seconds to solve the mathematical model (MS-ELS times are already reported in Table 3.17) is also shown. Table 3.18 recalls gap and CPU time if a warm start is not available with the aim to easily display for the reader variations on performance if warm start is used. Column Δ time shows the variation on CPU time if a start solution is available. This variation is computed as

$$\Delta \text{ time} = (\text{time}_{MILP} - \text{time}_{WS})/\text{time}_{MILP} \times 100$$

where time_{MILP} and time_{WS} are the CPU times required to solve the mathematical model avoiding and including the initial solution, respectively. Moreover, column Δ gap, s the improvement on gap between solving the MILP without a start solution and the warm start strategy: Δ gap= $\text{gap}_{MILP} - \text{gap}_{WS}$ where gap_{MILP} and gap_{WS} are the gaps reported by the optimizer when solving MILP including and skipping the initial solution, respectively.

Table 3.18 shows that warm start improves gaps for some instances. Nonetheless, there is not

Table 3.17: Computational results on MS-ELS for EnCicla instances

Instance	Best	# best	Opt.	Avg.	MILP LB		MILP UB		Avg.
					Min. gap_{LB}	Avg. gap	Min. gap_{UB}	Avg. gap	time (s)
1	2737	3	*	2775.57	0.00%	1.38%	0.00%	1.38%	12.85
2	2769	4	*	2790.60	0.00%	0.77%	0.00%	0.77%	14.69
3	2781	2	*	2807.20	0.00%	0.93%	0.00%	0.93%	14.30
4	2722	7	*	2730.57	0.00%	0.31%	0.00%	0.31%	15.20
5	3074	8	*	3080.67	0.00%	0.21%	0.00%	0.21%	17.12
6	3364	1		3415.93	3.15%	4.61%	-2.26%	-0.72%	13.32
7	3066	3		3100.23	2.09%	3.16%	-0.07%	1.03%	13.03
8	2746	4	*	2762.83	0.00%	0.61%	0.00%	0.61%	16.12
9	2803	5	*	2865.13	0.00%	2.15%	0.00%	2.15%	14.01
10	2760	6	*	2766.57	0.00%	0.23%	0.00%	0.23%	15.86
11	2838	6	*	2853.90	0.00%	0.55%	0.00%	0.55%	15.17
12	2723	4	*	2734.20	0.00%	0.41%	0.00%	0.41%	14.41
13	2648	7	*	2650.83	0.00%	0.11%	0.00%	0.11%	15.74
14	2936	3	*	2942.10	0.00%	0.21%	0.00%	0.21%	14.66
15	2621	5	*	2628.77	0.00%	0.29%	0.00%	0.29%	16.17
16	2793	6	*	2806.40	0.00%	0.47%	0.00%	0.47%	17.07
17	2818	3		2850.43	1.45%	2.55%	0.00%	1.11%	13.59
18	2891	7	*	2894.33	0.00%	0.11%	0.00%	0.11%	14.87
19	2684	6	*	2692.53	0.00%	0.31%	0.00%	0.31%	12.91
20	2569	7	*	2575.97	0.00%	0.27%	0.00%	0.27%	16.15
21	2817	3		2832.00	0.92%	1.45%	0.00%	0.53%	14.97
22	2859	2	*	2868.30	0.00%	0.32%	0.00%	0.32%	15.11
23	2794	4	*	2844.90	0.00%	1.76%	0.00%	1.76%	14.37
24	2653	6	*	2660.57	0.00%	0.28%	0.00%	0.28%	17.05
25	2878	6	*	2887.60	0.00%	0.33%	0.00%	0.33%	14.73
26	2595	5	*	2608.53	0.00%	0.51%	0.00%	0.51%	16.29
27	2888	4		2912.40	0.48%	1.31%	0.00%	0.83%	13.35
28	2536	1	*	2781.13	0.00%	2.56%	0.00%	2.56%	19.29
29	2781	3	*	2824.10	0.00%	1.52%	0.00%	1.52%	16.34
30	2574	6	*	2597.80	0.00%	0.90%	0.00%	0.90%	17.29
31	2793	2	*	2838.47	0.00%	1.60%	0.00%	1.60%	14.72
32	2852	1	*	2892.53	0.00%	1.40%	0.00%	1.40%	14.40
33	2799	3		2810.33	0.54%	0.94%	0.00%	0.40%	14.76
34	2534	7	*	2544.40	0.00%	0.40%	0.00%	0.40%	14.83
35	2777	2		2792.67	0.90%	1.45%	-0.18%	0.38%	15.94
Avg.		4.34		2812.01	0.27%	1.04%	-0.07%	0.70%	15.16

evidence to conclude that mathematical model performance is improved due to a initial solution. CPU time improves 16.96% while gap decreases 0.10% over the 35 instances and 0.44% over the

eight instances with a variation on gap. Computational time may improve up to 84.21% if an upper bound is computed for MILP (instance 20). Warm start also helps to find new optimality proofs (instances 27 and 35) and reduce gaps even if no optimal solution is reported within the maximum computational times (e.g. instances 6, 7 and 17). Finally, there are cases in which warm start deteriorates computational times for MILP (e.g. instances 1, 2, 3 and 32) and also gap (instances 21 and 22).

3.5 Concluding remarks

Two mathematical models based on mixed-integer lineal programming for the 1–PDTSP and the SD1PTSP have been presented. The MILP proposed for the 1–PDTSP can be adapted to tackle split delivery and temporal storage operations. These models are able to deal with instances up to 60 locations in decent computational times finding optimal solutions for all the instances up to 40 locations. Results in this chapter are compared with a reported exact approach concluding that proposed models are competitive outperforming computational times and solution quality.

This chapter also describes a MS-ELS for the 1–PDTSP. A GRASP and an ILS algorithm as particular cases of MS-ELS are also presented. The obtained results show that MS-ELS outperforms two of the algorithms reported in the literature to deal with the 1–PDTSP. Moreover, for small instances, it is possible to evidence the benefits if the evolutionary component is added to GRASP and ILS algorithms. MS-ELS is able to find better solutions than those reported via GRASP and MS-ILS for instances with 40 and 50 nodes. MS-ELS finds optimal solutions on instances with up to 60 nodes and it also delivers better solutions for large instances up to 500 nodes in which optimal solutions are not reported so far.

From a practical perspective, the 1–PDTSP is a simplifying representation of the static BRP. Nevertheless, rebalacing decisions can be made for bike-sharing operations as 1–PDTSPs in small and medimum size BSSs if a single vehicle is able to serve station requeriments.

Conferences and publications

A full-paper based on the metaheuristic procedure developed in this chapter has been published in Annals of Operations Research:

- Palacio, J.D., Rivera, J.C. A multi-start evolutionary local search for the one-commodity pickup and delivery traveling salesman problem. Ann Oper Res (2020). https://doi.org/10.1007/s10479-020-03789-0

Table 3.18: Computational results for EnCicla instances with warm start on MILP

Instance	MILP (with warm start)				MILP		Δ gap	Δ time
	UB	LB	gap	Time (s)	gap	Time (s)		
1	2737	2737	0.00%	307.19	0.00%	194.43	0.00%	-58.00%
2	2769	2769	0.00%	142.01	0.00%	115.25	0.00%	-23.22%
3	2781	2781	0.00%	1859.57	0.00%	1315.59	0.00%	-41.35%
4	2722	2722	0.00%	417.00	0.00%	400.55	0.00%	-4.11%
5	3074	3074	0.00%	2717.02	0.00%	3556.00	0.00%	23.59%
6	3258	3361	3.06%	3600.00	5.29%	3600.00	2.23%	0.00%
7	3006	3066	1.96%	3600.00	2.15%	3600.00	0.19%	0.00%
8	2746	2746	0.00%	650.15	0.00%	961.45	0.00%	32.38%
9	2803	2803	0.00%	641.21	0.00%	1731.30	0.00%	62.96%
10	2760	2760	0.00%	53.98	0.00%	69.45	0.00%	22.28%
11	2838	2838	0.00%	920.17	0.00%	944.08	0.00%	2.53%
12	2723	2723	0.00%	28.53	0.00%	47.54	0.00%	39.99%
13	2648	2648	0.00%	14.43	0.00%	19.56	0.00%	26.23%
14	2936	2936	0.00%	1498.38	0.00%	2223.79	0.00%	32.62%
15	2621	2621	0.00%	92.69	0.00%	308.82	0.00%	69.99%
16	2793	2793	0.00%	128.85	0.00%	155.91	0.00%	17.36%
17	2796	2818	0.78%	3600.00	1.45%	3600.00	0.67%	0.00%
18	2891	2891	0.00%	1094.39	0.00%	3254.71	0.00%	66.38%
19	2684	2684	0.00%	26.65	0.00%	58.03	0.00%	54.08%
20	2569	2569	0.00%	3.82	0.00%	24.20	0.00%	84.21%
21	2784	2817	1.17%	3600.00	0.92%	3600.00	-0.25%	0.00%
22	2831	2860	1.01%	3600.00	0.00%	3443.89	-1.01%	-4.53%
23	2794	2794	0.00%	43.71	0.00%	75.66	0.00%	42.23%
24	2653	2653	0.00%	19.32	0.00%	35.50	0.00%	45.58%
25	2878	2878	0.00%	1010.72	0.00%	865.95	0.00%	-16.72%
26	2595	2595	0.00%	36.66	0.00%	178.14	0.00%	79.42%
27	2888	2888	0.00%	2470.97	0.48%	3600.00	0.48%	31.36%
28	2536	2536	0.00%	4.04	0.00%	5.09	0.00%	20.63%
29	2781	2781	0.00%	18.71	0.00%	74.07	0.00%	74.74%
30	2574	2574	0.00%	7.52	0.00%	10.44	0.00%	27.97%
31	2793	2793	0.00%	1917.14	0.00%	2433.94	0.00%	21.23%
32	2852	2852	0.00%	2246.05	0.00%	819.73	0.00%	-174.00%
33	2788	2799	0.39%	3600.00	0.54%	3600.00	0.14%	0.00%
34	2534	2534	0.00%	3.30	0.00%	4.22	0.00%	21.80%
35	2777	2777	0.00%	3029.92	1.08%	3600.00	1.08%	15.84%
Avg.			0.24%	1228.69	0.34%	1386.49	0.10%	16.96%

A book chapter based on results and analysis on MILPs for 1–PDTSP and SD1PDTSP have been published in the Communications in Computer and Information Science book series:

- Palacio J.D., Rivera J.C. (2019) Mixed-Integer Linear Programming Models for One-Commodity Pickup and Delivery Traveling Salesman Problems. In: Figueroa-García J., Duarte-González M., Jaramillo-Isaza S., Orjuela-Cañon A., Díaz-Gutierrez Y. (eds) Applied Computer Sciences in Engineering. WEA 2019. Communications in Computer and Information Science, vol 1052. Springer, Cham. https://doi.org/10.1007/978-3-030-31019-6_62

Results and analysis on MILPs for 1–PDTSP and SD1PDTSP were presented in the Workshop on Engineering Applications (WEA) 2019:

- Palacio J.D., Rivera J.C. Mixed-Integer Linear Programming Models for One-Commodity Pickup and Delivery Traveling Salesman Problems. Workshop on Engineering Applications (WEA). Santa Marta, Colombia. 2019.

Preliminary results on an hybrid GRASP and VND algorithm for the 1–PDTSP were presented in II Congreso Colombiano de Investigación Operativa (ASOCIO), 2017:

- Palacio J.D., Rivera J.C. Diseño de rutas para la distribución de bicicletas compartidas: estrategias exactas y heurísticas. II Congreso Colombiano de Investigación Operativa (ASOCIO). Medellín, Colombia. 2017

Chapter 4

Mathematical models and solution approaches for one-commodity pickup and delivery vehicle routing problems

4.1 Introduction

This chapter addresses the 1–PDVRP. The 1–PDVRP can be seen as a generalization of the VRP. It aims to design a set of routes able to pick up or delivery a known amount of a single product supplied or demanded by two different types of locations (pickup or delivery nodes). To do so, a capacitated fleet of homogeneous vehicles is available to meet the demand while the total traveling cost is minimized. However, when several vehicles are available, optimal routes for the problem are not typically cost-balanced. For practical purposes (i.e., repositioning operation in BSSs), this situation must be avoided since decisions on workload for vehicles operators would not be fairly made. Therefore, apart from a 1–PDVRP mathematical formulation, this chapter also describes an MILP for the 1–PDVRPLC. The proposed strategy to model route length conditions allows to compare the performance and balance for different length parameters.

This chapter also describes two hybrid procedures for the 1–PDVRPLC based on large neighborhood search algorithms. These procedures are matheuristic strategies where destroy and repair methods are replaced by MILPs. Particularly, the first procedure consists on a multi-start iterative LNS (MS–ILNS). In the MS–ILNS, a single mathematical model partially destroys a solution and also repair it for a set of iterations under a fixed number of initial solutions. The second strategy is based on an adaptive LNS where several destroy methods are used and an MILP is able

to repair solutions. Given the adaptive nature of the algorithm, weights for destroy methods are dynamically computed as well as the weights to decide whether a solution may be improved by removing one route or by relocating nodes preserving the available number of routes. For this ALNS-based matheuristic, an *enumeration* algorithm is also described as an alternative way to repair solutions. Similar to MILP repair operator, this enumeration algorithm is able to retrieve optimal solution for repairing process. For this algorithm, dominance rules are described as a way to speedup its performance.

This chapter is structured as follows. Firstly, Section 4.2 presents the problem definition and an MILP for 1–PDVRP and 1–PDVRPLC. Next, Section 4.3 describes the LNS based matheuristic for the 1–PDVRPLC (MS–ILNS) as well as its main components. Thirdly, Section 4.4 is devoted to describe the adaptive version of the matheuristic algorithm. In Section 4.5, the enumeration algorithm able to replace MILP as repair method in ALNS matheuristic is outlined. Finally, sections 4.6 and 4.7 show computational experiments and some concluding remarks, respectively.

4.2 Problem definition and mixed integer linear models

This section describes two mathematical models. The first one, an MILP for the 1–PDVRP while the second one is an adaptation of the 1–PDVRP model for the 1–PDVRPLC. Similarly to the graphical description given in Section 3.2 for the 1–PDTSP and the SD1PDTSP, this section provides a comparison between 1–PDVRP and 1–PDVRPLC solution showing the main motivation to study the second problem.

4.2.1 The 1–PDVRP

Similarly to 1–PDTSP and SD1PDTSP, the 1–PDVRP is modeled on a complete and directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. The set $\mathcal{N}$ contains all the nodes and the depot (i.e., node 0). The set $\mathcal{A}$ denotes the arcs (i, j) between each pair of nodes which are labeled with non-negative cost c_{ij}. In order to visit all nodes, a fleet $\mathcal{K}$ of homogeneous capacitated vehicles is available at the depot. The demand on node $i \in \mathcal{N}$ is denoted by q_i. As defined in Section 3.2.1, negative q_i values indicate that node i is a pickup node while delivery locations are labeled with positive q_i values. Demand on depot, q_0, may also be positive, negative or zero. This imply that vehicles can leave depot with loads greater than zero. As the number of available commodity units is assumed as a known constant, the sum of all node demands is equal to zero. While a vehicle $k \in \mathcal{K}$ visits a subset of nodes, its load cannot overpass the limited capacity of Q units.

The proposed MILP is based on four types of decision variables. The binary decision variable w_i^k takes the value of one if vehicle k visits node i, and zero otherwise. Similarly, the binary decision variable y_{ij}^k denotes whether the vehicle k traverses the arc (i,j) or not. Continuous variable l_{ij}^k is the load of the vehicle k when it goes from node i to node j while z_{ij}^k determines a label for arc (i,j) avoiding subtours in the solution. The proposed mathematical model for the 1–PDVRP follows.

$$\min f = \sum_{(i,j)\in\mathcal{A}} c_{ij} \cdot \sum_{k\in\mathcal{K}} y_{ij}^k \tag{4.1}$$

subject to,

$$\sum_{k\in\mathcal{K}} w_i^k = 1, \qquad \forall\, i \in \mathcal{N}\setminus\{0\} \tag{4.2}$$

$$\sum_{j\in\mathcal{N}\setminus\{0\}} y_{0j}^k = 1, \qquad \forall\, k \in \mathcal{K} \tag{4.3}$$

$$\sum_{\substack{j\in\mathcal{N}\\ j\neq i}} y_{ij}^k = w_i^k, \qquad \forall\, k \in \mathcal{K},\ i \in \mathcal{N} \tag{4.4}$$

$$\sum_{\substack{j\in\mathcal{N}\\ j\neq i}} y_{ij}^k = \sum_{\substack{j\in\mathcal{N}\\ j\neq i}} y_{ji}^k, \qquad \forall\, k \in \mathcal{K},\ i \in \mathcal{N} \tag{4.5}$$

$$l_{ij}^k \leq Q \cdot y_{ij}^k, \qquad \forall\, k \in \mathcal{K},\ (i,j) \in \mathcal{A} \tag{4.6}$$

$$\sum_{\substack{j\in\mathcal{N}\\ j\neq i}} l_{ji}^k - \sum_{\substack{j\in\mathcal{N}\\ j\neq i}} l_{ij}^k = q_i \cdot w_i^k, \qquad \forall\, k \in \mathcal{K},\ i \in \mathcal{N} \tag{4.7}$$

$$\sum_{j\in\mathcal{N}} z_{ji}^k - \sum_{j\in\mathcal{N}} z_{ij}^k = w_i^k, \qquad \forall\, k \in \mathcal{K},\ i \in \mathcal{N}\setminus\{0\} \tag{4.8}$$

$$z_{ij}^k \leq |\mathcal{N}| \cdot y_{ij}^k, \qquad \forall\, k \in \mathcal{K},\ (i,j) \in \mathcal{A} \tag{4.9}$$

$$w_i^k \in \{0,1\}, \qquad \forall\, k \in \mathcal{K},\ i \in \mathcal{N} \tag{4.10}$$

$$y_{ij}^k \in \{0,1\}, \qquad \forall\, k \in \mathcal{K},\ (i,j) \in \mathcal{A} \tag{4.11}$$

$$l_{ij}^k, z_{ij}^k \geq 0, \qquad \forall\, k \in \mathcal{K},\ (i,j) \in \mathcal{A} \tag{4.12}$$

The objective function in (4.1) aims to minimize the total travel cost. Equations in (4.2) ensure that each node is visited by only one vehicle while those in (4.3) force each vehicle to leave the depot once. Expressions in (4.4) and (4.5) guarantee that a vehicle gets in and leaves each node only once. With constraints in (4.6) the load of vehicles does not exceed their capacity Q. The set of flow constraints in (4.7) forces the model to meet the demand of nodes. Expressions in (4.8) limit arc coefficients and prevent subtours on the solution. For VRPs, subtours are forbidden using constraints in (4.7). However, when pickup and delivery operations are performed, those constraints are not enough since q_i may be negative. Finally, (4.10)–(4.12) describe the nature and domain of

the decision variables. As described in Section 3.2.1, variables l_{ij}^k and z_{ij}^k could be integer, but given the structure of the formulation, a continuous domain leads to integer values.

4.2.2 The 1–PDVRP with tour length constraints

Equations in (4.3) force the model to design a route for each vehicle. This may end up in an optimal solution like the one depicted in Figure 4.1 in which three vehicles are available to serve 20 nodes. The total traveling cost for this solution is 5263 and route costs for vehicle 1, 2 and 3 are 2935, 1264 and 1064, respectively. A simple average cost per route for a balanced solution would be around $5263/3 \approx 1754$; however, the optimal solution in Figure 4.1 leads to an imbalance I, equal to 1871 (i.e., 2935-1064) computed as follows:

$$I = \max_{k \in \mathcal{K}} \left\{ \sum_{(i,j) \in \mathcal{A}} c_{ij} \cdot y_{ij}^k \right\} - \min_{k \in \mathcal{K}} \left\{ \sum_{(i,j) \in \mathcal{A}} c_{ij} \cdot y_{ij}^k \right\} \tag{4.13}$$

In order to reduce the mentioned imbalance and to model operational constraints like driver work shift duration, it is possible to fix a maximum route length (cost) T. Therefore, constraints to ensure this maximum length may be modeled as:

$$\sum_{(i,j) \in \mathcal{A}} c_{ij} \cdot y_{ij}^k \leq T, \qquad \forall\, k \in \mathcal{K} \tag{4.14}$$

Moreover, it is also possible to fix value for T based on an upper bound of the cost for the single vehicle case (1–PDTSP). This upper bound can be computed as the total cost if only one vehicle is available (i.e., the optimal solution for the 1–PDTSP). The proposed maximum route length is:

$$T = \frac{f^*_{\text{PDTSP}}}{|\mathcal{K}|} \cdot (1 + \alpha) \tag{4.15}$$

where f^*_{PDTSP} is the optimal solution of the 1–PDTSP and α is a real number where $0 \leq \alpha < |\mathcal{K}| - 1$. Note that if $\alpha = |\mathcal{K}| - 1$, T allows to find routes with cost up to f^*_{PDTSP}. Needless to say, since all the vehicles must visit a subset of nodes, a route with cost equal or greater than f^*_{PDTSP} is too inefficient and outperformed by a single vehicle solution. On the other hand, if $\alpha = 0$ then, T value forces the model to deliver a perfect balanced set of routes. For tight values of α, if no feasible solution exists, it is possible to find an unfeasibility measure. To do so, a new decision variable λ is defined. λ saves the greatest difference between a route cost and T. Moreover, variable λ must

be penalized in the objective function (4.1) and the set of constraints (4.17) for this variable are included in the mathematical formulation. These new expressions follow:

$$\min f = \sum_{(i,j)\in\mathcal{A}} c_{ij} \cdot \sum_{k\in\mathcal{K}} y_{ij}^{k} + M \cdot \lambda \tag{4.16}$$

$$\text{subject to,} \qquad \lambda \geq \sum_{(i,j)\in\mathcal{A}} c_{ij} \cdot y_{ij}^{k} - T, \qquad \forall\, k \in \mathcal{K} \tag{4.17}$$

$$\lambda \geq 0 \tag{4.18}$$

where M represents a large enough value in order to guarantee that solutions with larger λ values are dominated.

The new objective function (4.16) minimizes the total cost of the routes while penalizes unfeasible solutions in case that any route cost overpasses T. Constraints in (4.17) allow to λ to have the value of the maximum difference between route costs and T. Expression (4.18) avoids negative values for λ. Constraints (4.17) relax the expressions in (4.14).

If a solution satisfies expressions (4.14), λ becomes zero and the values of objectives (4.1) and (4.16) are equivalent. When $\lambda > 0$ indicates that expression (4.14) is not satisfied.

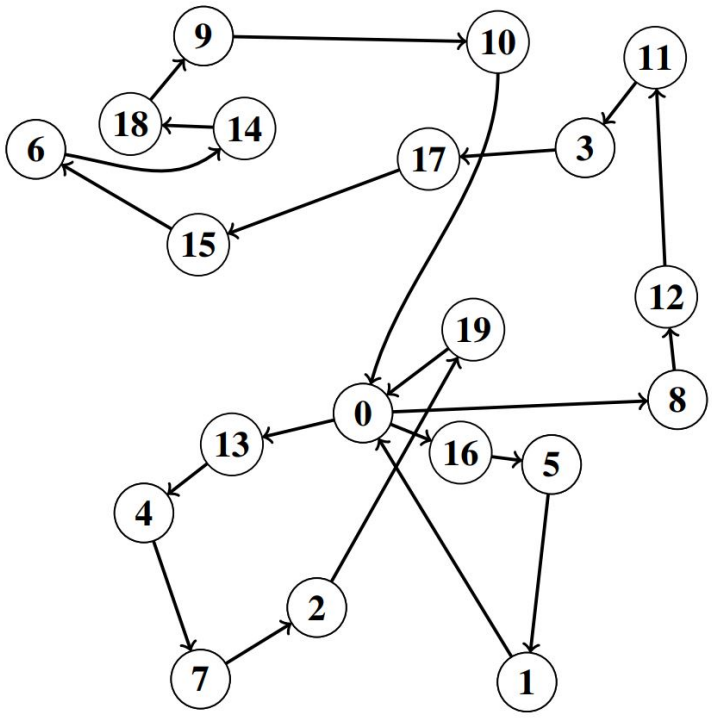


Figure 4.1: Instance 1 solved as 1–PDVRP without route length constraints ($f^* = 5263$)

Once objective function (4.1) is replaced by (4.16) and constraints in (4.17) and (4.18) are added to 1–PDVRP MILP with $\alpha = 0.5$ and $T = 2631$, the solution in Figure 4.1 becomes unfeasible and more balanced routes are obtained. The new optimal solution is depicted in Figure 4.2 in which cost of routes 1, 2 and 3 are 2 275, 2 130 and 1 264, respectively. Following Equation (4.13), $I = 1011$ for this new solution. The new total cost increases to 5669 and $\lambda = 0$. For this particular instance, the objective function value increases 7.2% and the imbalance I computed as in Equation (4.13) decreases 45.69%.

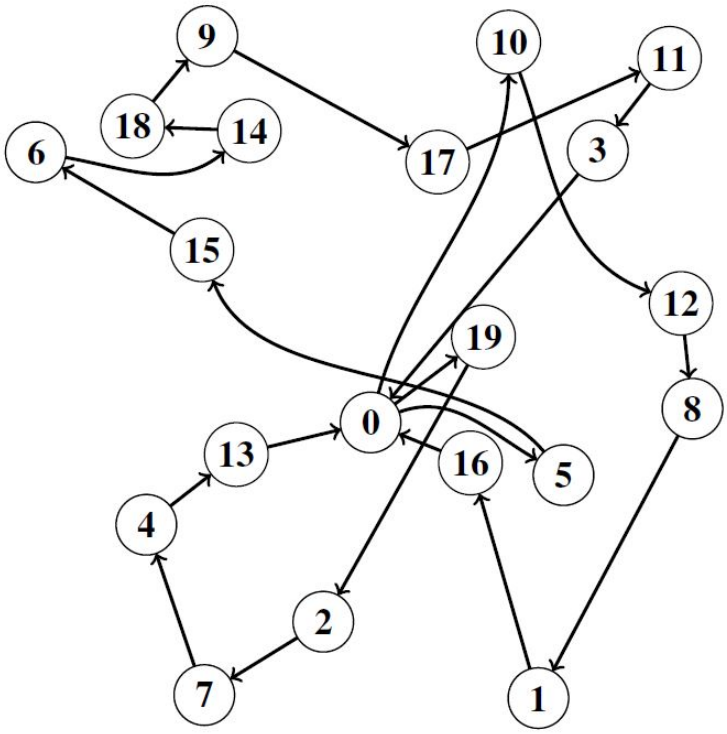


Figure 4.2: Instance 1 solved as 1–PDVRPLC with $T = 2631$ ($f^* = 5669$)

4.2.3 Symmetry breaking constraints

Since multiple vehicles are used to serve nodes, there exist a natural symmetry in 1–PDVRP and 1–PDVRPLC. With the aim to eliminate alternative symmetric solutions for the 1-PDVRP, one of the ideas proposed in Sherali and Smith (2001) is adapted to the 1–PDVRPLC MILP. Particularly, an order to use the vehicles depending on the cost of each route is imposed. To do so, the vehicles are assigned to routes in a decreasing fashion based on their cost by adding the next set of constraints to 1–PDVRPLC MILP:

$$\sum_{(i,j)\in\mathcal{A}} c_{ij}\cdot y_{ij}^{k} \geq \sum_{(i,j)\in\mathcal{A}} c_{ij}\cdot y_{ij}^{k+1}, \qquad \forall\, k \in \{1,..,|\mathcal{K}|-1\} \tag{4.19}$$

Finally, as first vehicle is always assigned to the route with largest cost value, the set of constraints in (4.17) can be replaced by:

$$\lambda \geq \sum_{(i,j)\in\mathcal{A}} y_{ij}^{1} - T \tag{4.20}$$

4.3 Large neighborhood search based matheuristic

This section presents a LNS based algorithm to solve the 1–PDVRPLC. LNS algorithms were firstly proposed in Shaw (1998) and a broad number of authors has developed LNS based algorithms to solve vehicle routing problems (Demir et al., 2012; Parragh and Schmid, 2013; Ropke and Pisinger, 2006; Sacramento et al., 2019; Smith and Imeson, 2017). A simple version of LNS algorithms starts with an initial solution which is gradually improved by iteratively destroying and repairing the solution. Algorithm 4.1 shows a LNS framework where s_0 denotes an initial solution and it is improved via a LS procedure as in Line 3. Then, part of the solution is destroyed and immediately, repaired with functions `Destroy` and `Repair`, respectively. Again, a local search procedure may also finds improvements on new solution s (Line 5 to 7). These steps are executed repeatedly until a stopping criterion (e.g., a maximum computational time) is reached. The algorithm returns the best found solution (s^*).

Algorithm 4.1: LNS framework

```
1:  input: initial solution (s_0)
2:  f* ← ∞, s* ← ∅
3:  s ← LocalSearch(s_0)
4:  repeat
5:      s_p ← Destroy(s)
6:      s' ← Repair(s_p)
7:      s ← LocalSearch(s')
8:      if f(s) < f* then
9:          s* ← s
10:         f* ← f(s)
11:     end if
12: until stopping criterion
13: return s*
```

For the 1–PDVRPLC, the proposed algorithm is based on a large neighborhood search procedure where a single MILP replaces the destroy and repair operations simultaneously. Moreover, and as a particular feature, the proposed LNS is also embedded on a multi-start iterated framework and a set partitioning problem (SPP) model attempts to improve the best solution found via multi-start iterated LNS (MS–ILNS). Next sections describe the general structure of the solution strategy as well as its main components.

4.3.1 General structure

As mentioned before, the proposed strategy to solve 1–PDVRPLC is based on a LNS within a multi-start iterated framework. Furthermore, destroy and repair operators in LNS are replaced by an MILP. Thus, the solution strategy is a matheuristic algorithm as the one depicted in Algorithm 4.2.

The algorithm requires two parameters: `MaxStarts` and `MaxIterations`. These values denote the number of initial solutions and the number of iterations to perform over each solution, respectively. The strategy starts defining an incumbent for the best found solution (z^*) and an empty set of solutions ($\mathcal{S}$). To compute an initial solution, a Hamiltonian tour T is constructed via a greedy randomized algorithm as described in Section 3.3.2. Then, a VND ($\texttt{VND}_T$) improves quality of tour T and a variation of Split algorithm (Prins, 2004) is used to find a multi-vehicle solution (s) (lines 5 and 6, respectively). Routes in s are added to set $\mathcal{S}$. For each solution, an MILP destroys and repairs part of the solution with the aim to improve objective function value (based on the total traveling cost) as shown in Line 15. Then, a VND for the solution ($\texttt{VND}_s$) finds local optima for s. This process is repeated until no improvement on s is found. As a diversification component, the iterative LNS concatenates and then, perturbs solution s represented as a Hamiltonian tour T (see lines 25 and 26). Similar to the procedure described for initial solutions, this tour T is improved via $\texttt{VND}_T$ and split with `Split` function (Lines 27 and 28). Finally, as a large number of routes are stored in $\mathcal{S}$, a SPP model is solved attempting to improve the best solution found with the MS–ILNS procedure.

In subsequent sections the main components of the MS–ILNS are described. As mentioned before, to construct the initial solution, the greedy randomized algorithm described in Section 3.3.2 is used. Thus, that particular component is not described again in this chapter. .

4.3.2 Variable neighborhood descent procedures

As mentioned in Section 3.3.3, VND searches improvements for an incumbent solution exploring increasingly the predefined set of neighborhoods and change from any of them to the first one

Algorithm 4.2: Multi-start iterative LNS matheuristic

```
1: function MS-ILNS(MaxStarts, MaxIterations)
2:     z* ⟵ ∞, s* ⟵ ∅, 𝕊 ⟵ ∅
3:     for i = 1 to MaxStarts do
4:         T ⟵ GreedyRandomizedAlgorithm(seed)
5:         T ⟵ VND_T(T)
6:         s ⟵ Split(T)
7:         𝕊 ⟵ 𝕊 ⋃ s
8:         if f(s) < z* then
9:             s* ⟵ s
10:            z* ⟵ f(s)
11:        end if
12:        for j = 1 to MaxIterations do
13:            repeat
14:                z' ⟵ f(s)
15:                s ⟵ Destroy & Repair MILP(s)
16:                s ⟵ VND_S(s)
17:                if f(s) < z' then
18:                    𝕊 ⟵ 𝕊 ⋃ s
19:                    if f(s) < z* then
20:                        s* ⟵ s
21:                        z* ⟵ f(s)
22:                    end if
23:                end if
24:            until z' = f(s)
25:            T ⟵ Concatenate(s)
26:            T ⟵ Perturbation(T)
27:            T ⟵ VND_T(T)
28:            s ⟵ Split(T)
29:            𝕊 ⟵ 𝕊 ⋃ s
30:            if f(s) < z* then
31:                s* ⟵ s
32:                z* ⟵ f(s)
33:            end if
34:        end for
35:    end for
36:    s* ⟵SPP IP(𝕊)
37:    return s*
38: end function
```

only if a better solution is found. Algorithm 4.3 depicts a general framework for VND to improve Hamiltonian tours with the aim to find solutions for 1–PDVRPLC. Given the incumbent from an

Algorithm 4.3: Variable neighborhood descent algorithm

```
1: function: VND (T)
2: T' ⟵ T
3: v ⟵ 1
4: while v ≤ v_max do
5:     T ⟵ LocalSearch(T', N_v(T'))
6:     if f(T) < f(T') then
7:         T' ⟵ T
8:         v ⟵ 1
9:     else
10:        v ⟵ v + 1
11:    end if
12: end while
13: return T'
```

initial Hamiltonian tour T', the set of neighbor solutions reachable from T', $N_k(T')$, is evaluated applying local search operator v (i.e., `LocalSearch`$(T',\ N_v(T'))$ where $v \leq v_{max}$. If the total traveling cost from the obtained solution (T) is better that the incumbent, then the incumbent and the best found solution T' are updated and, the search continues with $v = 1$. Otherwise the next neighborhood is explored ($v \longleftarrow v+1$). The algorithm stops when $v > v_{max}$.

The MS–ILNS for the 1–PDVRPLC described in Algorithm 4.2 calls two functions based on a VND procedure: $\texttt{VND}_T$ and $\texttt{VND}_S$, to improve Hamiltonian tours and 1–PDVRPLC solutions, respectively. A description on these two functions follows:

VND for Hamiltonian tours ($\texttt{VND}_T$)

The first proposed VND ($\texttt{VND}_T$) for the 1–PDVRPLC is based on three neighborhoods for Hamiltonian tours: *move vertex*, $2-opt$ and $3-opt$. These three operators are embedded on `LocalSearch` function for v=1, 2 and 3, respectively. All operators follow the best improvement criterion. A description of the implementation for these neighborhoods follows.

(i) Move vertex: the first local search operator ($v = 1$), attempts to improve the total traveling cost of a tour by moving a node from its original position p to a different one in the path. The new position p' may vary from 1 to $|\mathcal{N}|-1$ with $p' \neq p$. It is worth to recall that since tour T' is feasible, *move vertex* aims to improve the solution preserving its feasibility (see condition in (3.11)).

The complexity of *move vertex* operator is $\mathcal{O}(|\mathcal{N}|^2)$. Nonetheless, this implementation follows the ideas described in Lin (1965) and Section 3.3.3 for $2-opt$ and $3-opt$ operators. Firstly, a

list of m nearest locations for each node is stored. Then, the operator only scans a possible move for node i if it can be located one position before one of its m closest stations. Following this strategy, the complexity of *move vertex* operator is reduced to $\mathcal{O}(|\mathcal{N}| \cdot m)$.

(*ii*) $2-opt$: similarly to *move vertex*, $2-opt$ as second local search operator ($v = 2$) in VND, aims to improve total traveling cost for T' while preserving its feasibility. This neighborhood removes two edges and therefore the tour is split in two paths. Then, those paths are connected again in the other possible way. The computational complexity of $2-opt$ is $\mathcal{O}(|\mathcal{N}|^2)$ but as in *move vertex*, the number of potential edge exchanges to the m closest nodes to location i is reduced. Thus, $2-opt$ also runs in $\mathcal{O}(|\mathcal{N}| \cdot m)$.

(*iii*) $3-opt$: the third and final neighborhood within VND ($v = 3$) removes three edges of the original tour and then, four different solutions may be found. If all possible combinations when deleting three edges are explored, $3-opt$ complexity is $\mathcal{O}(|\mathcal{N}|^3)$. In this implementation, only m possible moves are explored for each one of the m nearest neighbors for each node i. This strategy leads to a complexity of $\mathcal{O}(|\mathcal{N}| \cdot m^2)$ instead of $\mathcal{O}(|\mathcal{N}|^3)$.

Finally, adequate values for m within these three VND operators depend on the size of the problem (i.e., $|\mathcal{N}|$) as mentioned in prior work for the 1–PDTSP (Hernández-Pérez et al., 2009; Palacio and Rivera, 2022) and in Section 3.3.3.

VND for 1–PDVRPLC solutions ($\texttt{VND}_S$)

The second VND function called in the MS–ILNS is $\texttt{VND}_S$, to improve total traveling times over multi-vehicle solutions (i.e., feasible solutions for 1–PDVRPLC). Similar to $\texttt{VND}_T$, $\texttt{VND}_S$ works as an intensification operator within the MS–ILNS algorithm via three neighborhoods exploration: string exchange, string relocation and string cross. These operators were initially defined in Van Breedam (1994) as inter-route moves. $\texttt{VND}_S$ explores on these neighborhoods with best improvement criterion. Moreover, since MS–ILNS algorithm searches only on feasible solution space, $\texttt{VND}_S$ and its operators requires to check whether a move is feasible. Thus, some of the properties described in Dell'Amico et al. (2016) for 1–PDVRP feasible movements on local search operators are checked on $\texttt{VND}_S$ algorithm. Before describing the local search operators for $\texttt{VND}_S$, the properties presented and proved in Dell'Amico et al. (2016) are also mentioned. Nevertheless, we also described the *amount of feasibility* concept as follows:

Let R be a route. As equation (3.11) states, R is feasible if

$$\max_{i\in\mathcal{N}}\{l_{R(i)}\} - \min_{i\in\mathcal{N}}\{l_{R(i)}\} \leq Q \tag{4.21}$$

Then, the surplus for the vehicle capacity after route R is performed is called the *amount of feasibility* and is calculated as follows:

$$\Delta_R = Q - \max_{i\in\mathcal{N}}\{l_{R(i)}\} + \min_{i\in\mathcal{N}}\{l_{R(i)}\} \tag{4.22}$$

Once Δ_R is computed, if $i \in R$ and $|q_i| \leq \Delta_R$, then node i can be feasibly removed from route R.

Based on the *amount of feasibility* for any route R, properties for feasible LS movements in 1-PDVRP solutions are:

a. If $i \notin R$ and $|q_i| \leq \Delta_R$, then the node i can be feasibly inserted in any position of route R.

b. If $i \in R$ and $|q_i| \leq \Delta_R$, then node i can be feasibly moved to any position along route R.

c. If i and $j \in R$, and $|q_i - q_j| \leq \Delta_R$, then the swap of nodes i and j is feasible.

d. Let R' be a route and $R \neq R'$. If $i \in R$, $j \in R'$, and $|q_i - q_j| \leq \min\{\Delta_R, \Delta_{R'}\}$, then the swap of nodes i and j is feasible.

With the described properties for local search moves in 1–PDVRP solutions, the search procedures used in $\texttt{VND}_S$ may be described as follows:

(*i*) *String exchange*: this operation takes two strings (where a string is defined as a chain of at most k nodes) from two different routes and exchanges them. In string exchange for $\texttt{VND}_S$, k always take the value of one. This is, the operator exchanges two nodes taken from different routes in a solution. Algorithm 4.4 describes the string exchange process. Firstly, for each route R, values for Δ_R are computed as in equation (4.22). Then, for each pair of routes R and R' where $R \neq R'$ and each pair of nodes $i \in R$ and $j \in R'$, the exchange of these nodes is evaluated (see Line 11) following property (c). If this exchange is feasible, then a variation on cost for solution s is computed within the function `StringExchangeCost`. Note that the variation can be obtained by adding the costs of new arcs in the solution and removing the costs of deleted arcs if exchange is executed. New traveling cost does not need to be computed from scratch. Following the best improvement criterion, nodes i and j are stored as well as cost improvement (Lines 13 to 17). For each pair of routes, if a feasible exchange reduces the total traveling cost ($\delta^* \leq 0$), routes in solution s are updated (Line 23). Finally, solution s is updated.

Algorithm 4.4: String exchange for 1–PDVRPLC

1: **function:** String exchange(s)
2: $\mathcal{R} \longleftarrow$ `ExtractRoutes(`s`)`
3: **for** $R \in \mathcal{R}$ **do**
4: $\quad \Delta_R \longleftarrow Q - \max_{i \in \mathcal{N}}\{l_{R(i)}\} + \min_{i \in \mathcal{N}}\{l_{R(i)}\}$
5: **end for**
6: **for** $R = 1$ to $|\mathcal{R}| - 1$ **do**
7: $\quad$ **for** $R' = R + 1$ to $|\mathcal{R}|$ **do**
8: $\quad\quad \delta^* \longleftarrow 0$
9: $\quad\quad$ **for** $i \in R$ **do**
10: $\quad\quad\quad$ **for** $j \in R'$ **do**
11: $\quad\quad\quad\quad$ **if** $|q_i - q_j| \leq \min\{\Delta_R, \Delta_{R'}\}$ **then**
12: $\quad\quad\quad\quad\quad \delta \longleftarrow$ `StringExchangeCost(`R_i, R'_j`)`
13: $\quad\quad\quad\quad\quad$ **if** $\delta \leq \delta^*$ **then**
14: $\quad\quad\quad\quad\quad\quad i^* \longleftarrow i$
15: $\quad\quad\quad\quad\quad\quad j^* \longleftarrow j$
16: $\quad\quad\quad\quad\quad\quad \delta^* \longleftarrow \delta$
17: $\quad\quad\quad\quad\quad$ **end if**
18: $\quad\quad\quad\quad$ **end if**
19: $\quad\quad\quad$ **end for**
20: $\quad\quad$ **end for**
21: $\quad\quad$ **if** $\delta^* \leq 0$ **then**
22: $\quad\quad\quad f(s) \longleftarrow f(s) + \delta^*$
23: $\quad\quad\quad \mathcal{R} \longleftarrow$ `UpdateExchange(`R, R', i^*, j^*`)`
24: $\quad\quad$ **end if**
25: $\quad$ **end for**
26: **end for**
27: $s \longleftarrow$ `UpdateSolution(`$\mathcal{R}, f(s)$`)`
28: **return** s

(*ii*) *String relocation*: the relocation operator removes a string from a route and finds a new position to insert it in a different route. Similar to *string exchange*, this operator for $\texttt{VND}_S$ takes strings of size 1 (i.e., only one node is removed and inserted). Algorithm 4.5 depicts the string relocation operator for 1–PDVRPLC. After the *amount of feasibility* is computed for each route, the algorithm checks for each node in route R whether it can be feasibly removed (Line 8) as property (a) describes. Then, for each position in a different route (R'), and applying property (e) in Line 11, the procedure determines if it is feasible to insert the node in R'. Next, a traveling cost improvement is computed if node i is inserted at position j with function `StringRelocationCost`. As described for *string exchange*, *string relocation* stores the best improvement (i.e., i^*, j^* and δ^*) and update routes and solution.

Algorithm 4.5: String relocation for 1–PDVRPLC

function: String relocation(s)
$\mathcal{R} \longleftarrow$ `ExtractRoutes(`s`)`
for $R \in \mathcal{R}$ **do**
    $\Delta_R \longleftarrow Q - \max_{i \in \mathcal{N}}\{l_{R(i)}\} + \min_{i \in \mathcal{N}}\{l_{R(i)}\}$
end for
for $R = 1$ to $|\mathcal{R}|$ **do**
    for $i \in R$ **do**
        if $|q_i| \leq \Delta_R$ **then**
            for $R' = 1$ to $|\mathcal{R}'|, R \neq R'$ **do**
                $\delta^* \longleftarrow 0$
                if $|q_i| \leq \Delta'_R$ **then**
                    for $j \in R'$ **do**
                        $\delta \longleftarrow$ `StringRelocationCost(`$R_i, R'_{[j]}$`)`
                        if $\delta \leq \delta^*$ **then**
                            $i^* \longleftarrow i$
                            $j^* \longleftarrow [j]$
                            $\delta^* \longleftarrow \delta$
                        end if
                    end for
                end if
            end for
            if $\delta^* \leq 0$ **then**
                $f(s) \longleftarrow f(s) + \delta^*$
                $\mathcal{R} \longleftarrow$ `UpdateRelocation(`R, R', i^*, j^*`)`
            end if
        end if
    end for
end for
$s \longleftarrow$ `UpdateSolution(`$\mathcal{R}, f(s)$`)`
return s

(*iii*) *String cross*: this operator removes one edge from two routes and then, the resulting strings are linked again in the other possible way. A small example of *string cross* is depicted in Figure 4.3. Firstly, let routes R and R' as in Figure 4.3a be two candidates for the local search operator. If edges (3,4) and (8,9) are removed, then routes are divided in four strings a, b, c and d as shown in Figure 4.3b. As described in equation (3.15), a feasible path for pickup and delivery route fulfills that the difference between maximum and minimum loads is less or equal than vehicle capacity (Q). Moreover, if $l^-_{P(i)}$ and $l^+_{P(i)}$ denote the minimum and maximum load of route P after i nodes are visited, thus $l^+_{R(4)} - l^-_{R(4)} \leq Q$ and $l^+_{R'(5)} - l^-_{R'(5)} \leq Q$. To determine

whether a string cross as the one shown in Figure 4.3c is feasible, following conditions must be met:

$$\max\{l^{+}_{R(4)}, l_{R(4)} - q_9\} - \min\{l^{-}_{R(4)}, l_{R(4)} - q_9\} \leq Q \tag{4.23}$$

$$\max\{l^{+}_{R'(5)}, l_{R'(5)} - q_4\} - \min\{l^{-}_{R'(5)}, l_{R'(5)} - q_4\} \leq Q \tag{4.24}$$

where q_9 and q_4 denote the demand at nodes 9 and 4 respectively. Additive inverse of these two values also represent minimum as well as maximum load for strings b within routes R and R'. Explicitly, if $\theta^{+}_{b,R}, \theta^{+}_{b,R'}, \theta^{-}_{b,R}$ and $\theta^{-}_{b,R'}$ store maximum and minimum loads for strings b in routes R and R', then $\theta^{+}_{b,R} = -q_4, \theta^{+}_{b,R'} = -q_9, \theta^{-}_{b,R} = -q_4$ and $\theta^{-}_{b,R'} = -q_9$. Since *string cross* must be evaluated for each pair of edges in R and R', second possible neighbor is depicted in Figure 4.3d. Nonetheless, minimum and maximum loads over string b in route R' (i.e., $\theta^{+}_{b,R'}$ for $b = [8,9]$) must be firstly computed. This can be done recursively if information about string $b = [9]$ is available (computed while neighbor 1 is evaluated as in Figure 4.3c. Minimum and maximum loads for string [8,9] in route R' are computed as:

$$\theta(2)^{+}_{b,R'} = \max\{\theta(1)^{+}_{b,R'} - q_8, -q_8\} \tag{4.25}$$

$$\theta(2)^{-}_{b,R'} = \min\{\theta(1)^{-}_{b,R'} - q_8, -q_8\} \tag{4.26}$$

Similarly, if a third neighbor as shown in Figure 4.3e is evaluated, minimum and maximum loads for string b in route R' are computed as:

$$\theta(3)^{+}_{b,R'} = \max\{\theta(2)^{+}_{b,R'} - q_7, -q_7\} \tag{4.27}$$

$$\theta(3)^{-}_{b,R'} = \min\{\theta(2)^{-}_{b,R'} - q_7, -q_7\} \tag{4.28}$$

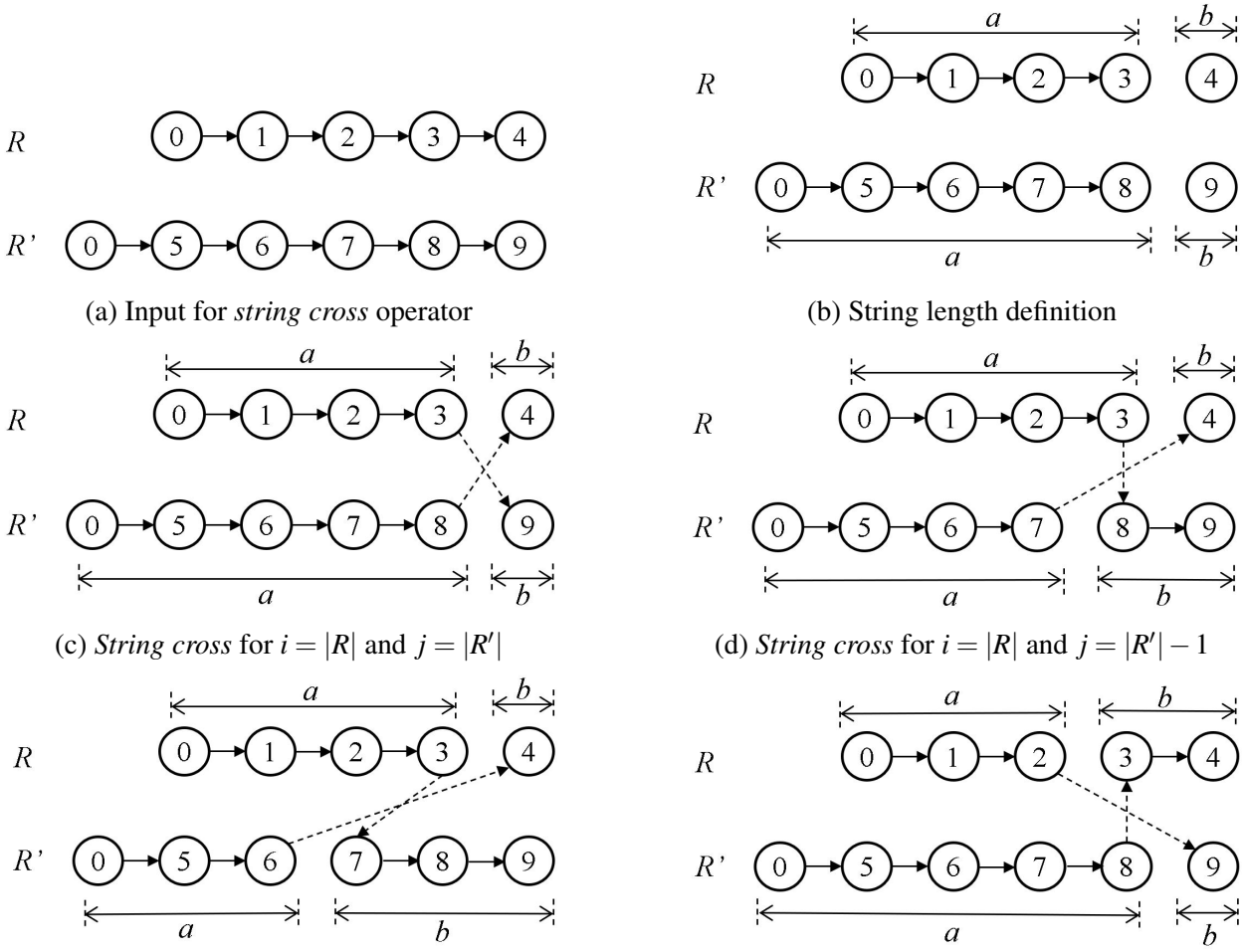
and neighbor in Figure 4.3e is feasible if

$$\max\{l^{+}_{R(4)}, l_{R(4)} + \theta(3)^{+}_{b,R'}\} - \min\{l^{-}_{R(4)}, l_{R(4)} + \theta(3)^{-}_{b,R'}\} \leq Q \tag{4.29}$$

$$\max\{l^{+}_{R'(3)}, l_{R'(3)} - q_4\} - \min\{l^{-}_{R'(3)}, l_{R'(3)} - q_4\} \leq Q \tag{4.30}$$

Needless to say, length of string b for R' increases as the maximum tour length is not violated. In that case, a new neighbor as the one in Figure 4.3f is evaluated computing maximum and minimum loads similarly to those described in expressions (4.23) and (4.24).

The string cross operator described in Figure 4.3 as an example, can be represented formally in Algorithm 4.6. This algorithm finds the best improvement on string cross searching within all



(a) Input for *string cross* operator

(b) String length definition

(c) *String cross* for $i = |R|$ and $j = |R'|$

(d) *String cross* for $i = |R|$ and $j = |R'| - 1$

(e) *String cross* for $i = |R|$ and $j = |R'| - 2$

(f) *String cross* for $i = |R| - 1$ and $j = |R'|$

Figure 4.3: *String cross* example

pairs of routes in solution s (Lines 3 and 5). Values for $\theta^{+}_{b,R}, \theta^{+}_{b,R'}, \theta^{-}_{b,R}$ and $\theta^{-}_{b,R'}$ are initialized in Lines 4 and 6. Similarly to these four values, it is possible to compute $\theta^{+}_{a,R}, \theta^{+}_{a,R'}, \theta^{-}_{a,R}$ and $\theta^{-}_{a,R'}$ (see Lines 8 and 10). Nevertheless, these values are not computed from scratch since they are inherently stored in each route information (as $l^{+}_{R(p)}, l^{+}_{R'(p)}, l^{-}_{R(p)}$ and, $l^{-}_{R'(p)}$, respectively, for each position p in routes R and R'). Variables $\gamma^{+}_{a,R,b,R'}$ and $\gamma^{-}_{a,R,b,R'}$ store values for maximum and minimum loads if string a in route R is crossed with string b in route R'. In a similar way, $\gamma^{+}_{a,R',b,R}$ and $\gamma^{-}_{a,R',b,R}$ denote values for maximum and minimum loads if string a in route R' is crossed with string b in route (Lines 11, 12, 14 and 15). If crossing these string is a feasible operation, then a cost change is computed with function `StringCrossCost`. Best improvement is stored while values for minimum and maximum loads are updated for b strings on both routes. Lastly, the algorithm updates the routes and returns solution s.

Algorithm 4.6: String cross for 1–PDVRPLC

function: String cross(s)
$\mathcal{R} \longleftarrow$ `ExtractRoutes(`s`)`
for $R = 1$ to $|\mathcal{R}| - 1$ **do**
  $\theta^-_{b,R} \longleftarrow -q^R_{|R|}$, $\theta^+_{b,R} \longleftarrow -q^R_{|R|}$
  for $R' = R+1$ to $|\mathcal{R}|$ **do**
    $\theta^-_{b,R'} \longleftarrow -q^{R'}_{|R'|}$, $\theta^+_{b,R'} \longleftarrow -q^{R'}_{|R'|}$
    for $i = |R|$ to 1, **step** -1 **do**
      $\theta^-_{a,R} \longleftarrow l^-_{R(i)}$, $\theta^+_{a,R} \longleftarrow l^+_{R(i)}$
      for $j = |R'|$ to 1, **step** -1 **do**
        $\theta^-_{a,R'} \longleftarrow l^-_{R'(j)}$, $\theta^+_{a,R'} \longleftarrow l^+_{R(j)}$
        $\gamma^-_{a,R,b,R'} \longleftarrow \min\{\theta^-_{a,R}, l_{R(i)} + \theta^-_{b,R'}\}$
        $\gamma^+_{a,R,b,R'} \longleftarrow \max\{\theta^+_{a,R}, l_{R(i)} + \theta^+_{b,R'}\}$
        if $\gamma^+_{a,R,b,R'} - \gamma^-_{a,R,b,R'} \leq Q$ **then**
          $\gamma^-_{a,R',b,R} \longleftarrow \min\{\theta^-_{a,R'}, l_{R'(j)} + \theta^-_{b,R}\}$
          $\gamma^+_{a,R',b,R} \longleftarrow \max\{\theta^+_{a,R'}, l_{R'(j)} + \theta^+_{b,R}\}$
          if $\gamma^+_{a,R',b,R} - \gamma^-_{a,R',b,R} \leq Q$ **then**
            $\delta \longleftarrow$ `StringCrossCost(`$R_{(i)}, R'_{(j)}$`)`
            if $\delta \leq \delta^*$ **then**
              $R^1 \longleftarrow R$, $R^2 \longleftarrow R'$
              $i^* \longleftarrow i$, $j^* \longleftarrow j$, $\delta^* \longleftarrow \delta$
            end if
          end if
        end if
        $\theta^-_{b,R'} \longleftarrow \min\{\theta^-_{b,R'} - q_j, -q_j\}$
        $\theta^+_{b,R'} \longleftarrow \max\{\theta^+_{b,R'} - q_j, -q_j\}$
      end for
      $\theta^-_{b,R} \longleftarrow \min\{\theta^-_{b,R} - q_i, -q_i\}$
      $\theta^+_{b,R} \longleftarrow \max\{\theta^+_{b,R} - q_i, -q_i\}$
    end for
  end for
end for
if $\delta^* \leq 0$ **then**
  $f(s) \longleftarrow f(s) + \delta^*$
  $\mathcal{R} \longleftarrow$ `UpdateCross(`R^1, R^2, i^*, j^*`)`
end if
$s \longleftarrow$ `UpdateSolution(`$\mathcal{R}, f(s)$`)`
return s

4.3.3 Split procedure for the 1–PDVRP

This *route-first cluster-second* heuristic is initially described in Prins (2004) as an strategy to find a solution for the VRP from an initial Hamiltonian tour. For a classical capacitated VRP, Split is designed as a two phase method where firstly a TSP solution is found by relaxing vehicle capacity constraints. Next, this TSP solution (i.e., Hamiltonian tour) is split in feasible routes for the VRP. The Split algorithm for the CVRP is based on an auxiliary graph with $|\mathcal{N}|$ nodes and the set of arcs is composed by all the pairs of locations (i, j) if a trip visiting nodes $i+1$ to j is feasible in terms of load (Prins, 2004). Based on the initial Hamiltonian tour, the shortest path from the depot to the node in the position $|\mathcal{N}|$ on the auxiliary graph, provides the optimal way to split the tour.

In the LNS based strategy to solve the 1–PDVRPLC, Split requires a Hamiltonian tour T to find a set of routes (a solution) that met vehicles capacity (Q) and the maximum route length (TL). Without loss of generality, TL can be a fixed value or may be expressed as a function based on number of available vehicles (as described in equation (4.15)). Since Prins (2004) describes Split for the CVRP, a variation on this procedure is presented in Algorithm 4.7 to include vehicle capacity constraints under pickup and delivery operations as well as tour length limitation.

The algorithm starts assigning an initial cost when reaching each node i (V_i) in lines 2 to 4. Then, starting from position i, Split checks whether is possible to reach node in each position j in the tour. Constraints based on the maximum route length (TL) ($length \leq TL$) are evaluated as in constraints (4.14). Similarly, computing maximum and minimum vehicle loads (l^+ and l^-, respectively), feasible loads for the vehicle ($l^+ - l^- \leq Q$) are also imposed (see lines 10 to 22). After cost is updated if position j can be reached from node in position i, the algorithm checks for an improvement in the total traveling cost of the partial route (Line 26). Similarly, the strategy updates the path (P) for the new route (line 28). Lastly, the function `extractSolution` delivers the complete path for each route by extracting nodes added to P.

4.3.4 MILP-based destroy and repair operator

As mentioned in Section 4.3.1, in the MS-ILNS the destroy and repair functions are replaced by an MILP. This section describes the mathematical model able to remove part of the solution (i.e., a subset of nodes), and insert them in different positions improving total traveling costs. The proposed MILP requires as parameters the traveling cost c_{ij} from i to j as well as demand at node i (q_i) and the vehicle capacity Q. A binary parameter w_{ij} denotes whether arc (i, j) exists in the solution to destroy and repair. Binary variable y^-_{ij} takes the value of one if arc (i, j) is removed (destroyed) from the solution and zero otherwise. Similarly, y^+_{ij} is also a binary variable and denotes whether

Algorithm 4.7: Split algorithm for PDVRPs. Adapted from Prins (2004)

function: Split (T)
for $i = 1$ to $|\mathcal{N}|$ **do**
    $V_i \leftarrow +\infty$
end for
$V_0 \leftarrow 0$
for $i = 1$ to $|\mathcal{N}|$ **do**
    $l \leftarrow 0, l^+ \leftarrow 0, l^- \leftarrow 0$
    $cost \leftarrow 0$, $length \leftarrow 0$
    $j \leftarrow i$
    while $j < |\mathcal{N}|$ and $l^+ - l^- \leq Q$ and $length \leq TL$ **do**
        $l \leftarrow q_{T_{[j]}}$
        if $l < l^-$ **then**
            $l^- \leftarrow l$
        else
            if $l > l^+$ **then**
                $l^+ \leftarrow l$
            end if
        end if
        if $j = i$ **then**
            $cost \leftarrow 2 \cdot c_{T[i],T[j]}$
        else
            $cost \leftarrow cost - c_{T[j-1],T[i]} + c_{T[j-1],T[j]} + c_{T[j],T[i]}$
        end if
        $length \leftarrow length + cost$
        if $l^+ - l^- \leq Q$ and $length \leq TL$ **then**
            if $V_{i-1} + cost < V_j$ **then**
                $V_j \leftarrow V_{i-1} + cost$
                $P_j \leftarrow i - 1$
            end if
        end if
        $j \leftarrow j + 1$
    end while
end for
$s^* \leftarrow$ `extractSolution(`P`)`
return s^*

arc (i, j) is added to the new solution. Varible l_{ij} represents the vehicle load when traversing arc (i, j) while z_{ij} saves the number of arcs traversed before node i is visited. Finally, ψ denotes the maximum number of arcs to destroy and consequently, to repair in the solution. The proposed mathematical model able to destroy and repair a 1–PDVRPLC solution follows.

$$\min f = \sum_{(i,j)\in\mathcal{A}} c_{ij}\cdot(y_{ij}^{+}-y_{ij}^{-}) \tag{4.31}$$

subject to,

$$\sum_{(i,j)\in\mathcal{A}} y_{ij}^{+} \leq \psi \tag{4.32}$$

$$y_{ij}^{+}+y_{ij}^{-} \leq 1, \qquad \forall\,(i,j)\in\mathcal{A} \tag{4.33}$$

$$\sum_{\substack{j\in\mathcal{N}\\ i\neq j}} (w_{ij}+y_{ij}^{+}-y_{ij}^{-}) = 1, \qquad \forall\, i\in\mathcal{N} \tag{4.34}$$

$$\sum_{i\in\mathcal{N}} (w_{ij}+y_{ij}^{+}-y_{ij}^{-}) = \sum_{i\in\mathcal{N}} (w_{ji}+y_{ji}^{+}-y_{ji}^{-}), \qquad \forall\, j\in\mathcal{N} \tag{4.35}$$

$$y_{ij}^{-} \leq w_{ij}, \qquad \forall\,(i,j)\in\mathcal{A} \tag{4.36}$$

$$l_{ij} \leq Q\cdot(w_{ij}+y_{ij}^{+}-y_{ij}^{-}), \qquad \forall\,(i,j)\in\mathcal{A} \tag{4.37}$$

$$\sum_{j\in\mathcal{N}} l_{ji} - \sum_{j\in\mathcal{N}} l_{ij} = q_i, \qquad \forall\, i\in\mathcal{N} \tag{4.38}$$

$$\sum_{j\in\mathcal{N}} z_{ji} - \sum_{j\in\mathcal{N}} z_{ij} = 1, \qquad \forall\, i\in\mathcal{N}\backslash\{0\} \tag{4.39}$$

$$z_{ij} \leq |\mathcal{N}|\cdot(w_{ij}+y_{ij}^{+}-y_{ij}^{-}), \qquad \forall\,(i,j)\in\mathcal{A} \tag{4.40}$$

$$y_{ij}^{+}, y_{ij}^{-} \in \{0,1\}, \qquad \forall\,(i,j)\in\mathcal{A} \tag{4.41}$$

$$z_{ij}, l_{ij} \geq 0, \qquad \forall\,(i,j)\in\mathcal{A} \tag{4.42}$$

$$\psi \geq 0 \tag{4.43}$$

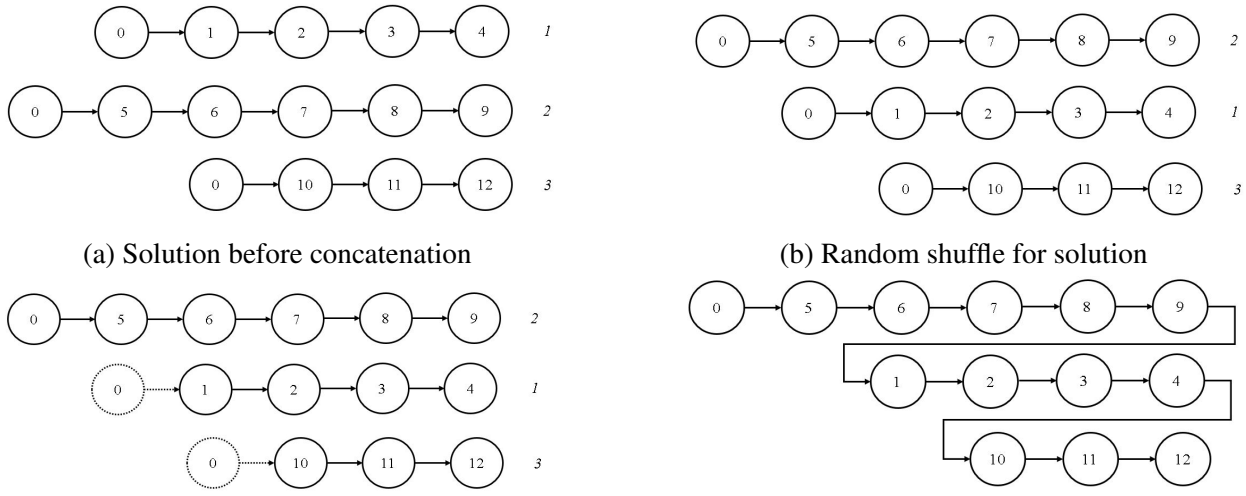
Objective function in (4.31) aims to minimize the traveling cost for destroyed and repaired arcs in a solution. Since traveling cost is computed through variables y_{ij}^{+} and y_{ij}^{-} values, only negative values for f denotes an improved solution. Constraint in (4.32) imposes that no more than ψ arcs must be repaired (and destroyed) for the new solution. Expressions in (4.33) force the model to include an arc (i,j) in the new route only if (i,j) does not exist in the original solution. Constraints (4.34) and (4.35) ensure that each node i in the route is visited only once even if at least one of its arcs is removed from the solution. Inequalities in (4.36) guarantee that only existing arcs can be destroyed. Vehicle capacity constraints are described in (4.37) while expressions in (4.38) force the model to meet demand at each node. Constraints (4.39) and (4.40) avoid subtours in the final solution. Finally, constraints in (4.41)–(4.43) describe the nature of decision variables.

4.3.5 Concatenation and perturbation functions

As mentioned in Section 4.3.1, Algorithm 4.2 calls `Concatenate` and `Perturbation` functions as part of diversification component for the MS–ILNS. Firstly, `Concatenate` function links the set of routes in a solution s as a Hamiltonian tour. Then, this tour is modified (i.e., a small number of nodes are moved to a different position) via `Perturbation` function. Since as a later step, Split algorithm is applied to perturbed tours, a different 1–PDVRPLC solution is obtained.

Function `Concatenate` simply shuffles randomly routes in solution s and then required arcs are linked to find a Hamiltonian tour. Figure 4.4 describes the complete process. A solution s composed of 13 nodes and three routes is depicted in Figure 4.4a. Figure 4.4b shows a random shuffle of s. In Figure 4.4c depot is removed from second and third route (and consequently, arcs connecting depot to nodes 1 and 10). Finally, required arcs as $(9,1)$ and $(4,10)$ are added to conform the Hamiltonian tour. It is worth to mention that this concatenation function may return an unfeasible Hamiltonian tour (i.e., vehicle load exceeds its capacity). Nevertheless, after perturbation is applied, Split algorithm is able to provide a new multi-vehicle solution with feasible loads as described in Section 4.3.3.

Once the concatenation is completed, function `Perturbation` slightly modify the Hamiltonian tour delivered by `Concatenate`. For the MS-ILNS algorithm, the perturbation function is based on the one described in Section 3.3.4 for the MS–ELS to solve the 1–PDTSP. For the MS–ILNS proposed to solve 1–PDVRPLC, the perturbation function allows unfeasible Hamiltonian tours (i.e., `Concatenate` may return unfeasible vehicle loads). Thus, this perturbation procedure does



(a) Solution before concatenation

(b) Random shuffle for solution

(c) Arcs to remove after shuffle

(d) New Hamiltonian tour after concatenation

Figure 4.4: Concatenate operation example

not check for feasible movements or improvements in total traveling cost. $2-opt$ operator for perturbation function only removes randomly selected arcs and links the nodes in the other possible way but no feasibility check is performed.

4.3.6 Set partitioning based post-optimization procedure

The MS–ILNS described in Section 4.3.1 starts with an empty set $\mathcal{S}$ which stores 1–PDVRPLC solutions after Split and VNDs are called. With the aim to improve the best solution found via MS–ILNS, a final procedure is added to the solution strategy: a set partitioning model. Let $\mathcal{R}$ be the set of all routes that conform solutions in $\mathcal{S}$. A SPP model can be solved for set $\mathcal{R}$ and a solution with a lower total traveling cost may be found along the routes of different solutions in $\mathcal{S}$. The parameters for the SPP IP include g_r as the traveling cost of route r $(r \in \mathcal{R})$. Parameter a_{ir} takes the value of one if node i is visited in route r and zero otherwise. Moreover, an upper bound for the number of vehicles required may also be imposed to the model. In CVRPs, a straightforward way to compute this upper bound consists on finding any integer greater than the optimal solution of a bin packing problem in which the weight of items is equal to demands and the bin capacity is the vehicle capacity. For large values of TL this upper bound is also valid for 1–PDVRPLC.

Finally, decision variable x_i takes the value of one if route r is selected in the SPP solution and zero otherwise. The SPP IP for the 1–PDVRPLC follows:

$$\min f = \sum_{r \in \mathcal{R}} g_r \cdot x_r \tag{4.44}$$

subject to,

$$\sum_{r \in \mathcal{R}} a_{ir} \cdot x_r = 1, \qquad \forall\, i \in \mathcal{N} \setminus \{0\} \tag{4.45}$$

$$\sum_{r \in \mathcal{R}} x_r \leq \bar{K}, \tag{4.46}$$

$$x_r \in \{0,1\}, \qquad \forall\, r \in \mathcal{R} \tag{4.47}$$

Objective function in (4.44) minimizes the total traveling cost. Constraints in (4.45) ensure that each node is visited once while expression (4.46) allows to select up to $\bar{K}$ routes. Note that this mathematical model is suitable to solve fixed fleet size problems by replacing $\bar{K}$ with the number of available vehicles. Lastly, expressions in (4.47) describe the domain of variables x_i.

4.4 Adaptive large neighborhood search algorithm

This section describes an extension of LNS algorithm presented in Section 4.3 which is also able to solve the 1–PDVRPLC. This extension is inspired on adaptive features added to LNS metaheuristic. Adaptive LNS (ALNS) was firstly proposed in Ropke and Pisinger (2006). Precisely, Ropke and Pisinger (2006) define ALNS as a LNS that allows several destroy and repair procedures with adaptive operator selection. These sets of destroy and repair procedures are available to be used in a single search. For each destroy/repair procedure an initial weight is assigned and then, those weights are dynamically adjusted during the search. Thus the metaheuristic behavior is adapted to the particular instance being solved. Algorithm 4.8 depicts a pseudocode for ALNS metaheuristic. Firstly, destroy and repair weights (values between 0 and 1) are initialized assigning the same weight for every destroy method and also for each repair method. Then, one destroy method (d) and one repair method (r) are selected randomly using a *roulette wheel principle* and weights computed for each method. Once these methods are selected, solution s is destroyed and then repaired. The algorithm checks whether best solution found must be updated and weights for destroy and repair operators are updated. This process is repeated until a stop criterion is met.

ALNS algorithms are widely used to solve vehicle routing problem and its extensions. For example, Li et al. (2016) and Liu et al. (2019) solve a VRP with time windows via ALNS. ALNSs are also used to solve VRPs with truck and trailers (Parragh and Cordeau, 2017), multi-depot VRP (Alinaghian and Shokouhi, 2018; Mancini, 2016), two-echelon VRP (Grangier et al., 2016; Hemmelmayr et al., 2012; Li et al., 2020), selective VRP (Aksen et al., 2014) and, periodic routing problems (Koç, 2016). Pickup and delivery problems are also addressed via ALNS: Masson et al. (2013) introduce transfer points between pickup and delivery nodes; Grimault et al. (2017) include temporal precedence and synchronization constraints. Lastly, time dependent extensions are also included in pickup and delivery problems solved via ALNS (Sun et al., 2020).

The ALNS presented in this section slightly differs from the classical algorithm described in Ropke and Pisinger (2006) and Algorithm 4.8. Firstly, the proposed ALNS aims to improve 1–PDVRPLC solutions through two different operations (*goals*): remove nodes from routes (and insert them in different positions) or, remove a complete route (and insert their nodes in other paths). The proposed ALNS is adaptive in two ways: (i) weights to select a *goal* are computed dynamically; (ii) since several destroy operators for removing nodes from routes are available, their weights are also updated dynamically. After a route is removed from the solution, its nodes are inserted in the other paths via MILP. Similarly, if some nodes are removed from routes, an MILP finds a new position for them in the solution.

Several components of the proposed ALNS based matheuristic are already described in previous sections. Constructive algorithm, VNDs, Split and, concatenate and perturbation for ALNS were previously presented in Sections 3.3.2, 4.3.2, 4.3.3 and, 4.3.5, respectively. Thus, following sections describe the general structure of the ALNS based matheuristic and its elements not yet defined.

Algorithm 4.8: ALNS framework

```
1:  input: initial solution (s)
2:  f* ← ∞, s* ← ∅
3:  InitializeDestroyOperatorWeights()
4:  InitializeRepairOperatorWeights()
5:  repeat
6:      d ←SelectDestroyOperator()
7:      r ←SelectRepairOperator()
8:      s' ← Destroy(s,d)
9:      s ← Repair(s',r)
10:     if f(s) < f* then
11:         s* ← s
12:         f* ← f(s)
13:     end if
14:     UpdateDestroyOperatorWeights()
15:     UpdateRepairOperatorWeights()
16: until stop criterion
17: return s*
```

4.4.1 General structure

The proposed ALNS algorithm for the 1–PDVRPLC is a matheuristic strategy since it combines (meta)heuristic procedures (Split, VND, greedy algorithm) with mathematical models to repair solutions. Algorithm 4.9 shows the ALNS matheuristic framework.

The adaptive nature of the algorithm is determined by *goals* and destroy strategies for routes that are selected dynamically. Therefore, two functions initialize these two sets of weights: `InitializeDestroyOperatorWeights` and `InitializeGoalWeights` for destroy operators and *goals*, respectively. After lines 3 and 4 are executed, vectors $\vec{\pi}$ and $\vec{\lambda}$ contain the same weight for each method or goal. Similar to the MS-ILNS described in Section 4.3.1, an initial Hamiltonian tour is constructed and improved via greedy randomized algorithm and $\texttt{VND}_T$ functions. Then, an initial 1–PDVRPLC solution is obtained and improved via Split algorithm and $\texttt{VND}_s$ (Lines 5 to 8).

The function `SelectGoal` determines randomly one out of two actions: destroy routes partially or remove a route completely. If a partial destruction of nr routes is selected ($\lambda = 1$), then a

Algorithm 4.9: Adaptive LNS matheuristic

```
1: function ALNS( MaxIterations)
2:     z* ⟵ ∞, s* ⟵ ∅
3:     π⃗ ⟵InitializeDestroyOperatorWeights()
4:     λ⃗ ⟵InitializeGoalWeights()
5:     T ⟵ GreedyRandomizedAlgorithm(seed)
6:     T ⟵ VND_T(T)
7:     s ⟵ Split(T)
8:     s ⟵ VND_S(s)
9:     if f(s) < z* then
10:        s* ⟵ s
11:        z* ⟵ f(s)
12:    end if
13:    for j = 1 to MaxIterations do
14:        repeat
15:            λ ⟵ SelectGoal()
16:            if λ = 1 then
17:                π ⟵ SelectDestroyOperator(π⃗)
18:                s ⟵ Destroy&RepairRoutes(s, π)
19:                π⃗ ⟵UpdateDestroyOperatorWeights()
20:            else
21:                s ⟵ RemoveRoute(s)
22:            end if
23:            λ⃗ ⟵UpdateGoalWeights()
24:            if f(s) < z* then
25:                s* ⟵ s
26:                z* ⟵ f(s)
27:            end if
28:        until stop criterion
29:        T ⟵ Concatenate(s)
30:        T ⟵ Perturbation(T)
31:        s ⟵ Split(T)
32:        if f(s) < z* then
33:            s* ⟵ s
34:            z* ⟵ f(s)
35:        end if
36:    end for
37:    return s*
38: end function
```

destroy operator must be selected (following *roulette wheel principle* based on computed weights for destroy methods); secondly, solution s is indeed destroyed and repaired (line 18). Weights

for destroy methods are updated as in line 19. On the other hand, if the algorithm attemps to destroy a whole route ($\lambda = 2$), then an MILP in function `RemoveRoute` is called (line 21). Weights for goals are updated in line 23. This process repeats until a stopping criterion is met. As a diversification component of ALNS matheuristic, `Concatenate`, `Perturbation`, and `Split` provide a new solution s for `MaxIterations` times. The algorithm returns the best solution found s^*.

In following sections main components of the ALNS matheuristic are discussed. Weights updates, methods for partially destroy routes and MILP for repair them. an MILP for completely remove a path is also described.

4.4.2 Adaptive control of the algorithm

As described in previous sections, the proposed ALNS matheuristic differs from the adaptive algorithm firstly described in Ropke and Pisinger (2006). Apart from the hybrid nature of this procedure to solve the 1–PDVRPLC, the adaptive control of the algorithm relies also in a different set of parameters called *goals*. This ALNS may improve solutions by relocating nodes through the original number of routes (first goal) or may also attempt to remove completely a route and assign its nodes to different paths (second goal). Thus, function `InitializeGoalWeights` defines a two position vector $\vec{\lambda}$, where $\vec{\lambda} = [0.5; 0.5]$. These values allows the algorithm to initially select with the same probability goals 1 or 2. In Line 15 of Algorithm 4.9, function `SelectGoal` chooses randomly via *roulette wheel principle* one of the goals.

In order to dynamically update values in $\vec{\lambda}$, a vector $\vec{\Lambda} = [\Lambda_1; \Lambda_2]$ keeps track of the number of times that the objective function is improved via each one of the goals (i.e., after a goal is selected, the solution is improved in lines 18 or 21 in Algorithm 4.9). Each time a better solution is found when routes are destroyed and repaired or a single route is removed, $\vec{\Lambda}$ is updated in function `UpdateGoalWeights`. In a similar way, and periodically (i.e., each ρ `SelectGoal` calls), $\vec{\lambda}$ values are computed within the same function, as follows:

$$\lambda_g = \frac{\Lambda_g}{\sum_{j=1}^{2} \Lambda_j}, \qquad \forall g = 1, 2 \tag{4.48}$$

Similarly to $\vec{\lambda}$, vector $\vec{\pi}$ is periodically updated redefining the probabilities of choosing a destroy method if a partial destroy and repair of routes is selected as goal. Vector $\vec{\Pi}$ manages the number of times a solution is improved after each destroy method is selected. Vector $\vec{\Pi}$ is updated each

time function `UpateDestroyOperatorWeights` is called in Line 19 of Algorithm 4.9. Let m be the number of strategies to partially destroy routes. Thus, values for vector $\vec{\pi}$ are computed as:

$$\pi_d = \frac{\Pi_d}{\sum_{j=1}^{m} \Pi_j}, \qquad \forall d = 1,..,m \tag{4.49}$$

Needless to say, from expressions in equations (4.48) and (4.49) it follows that $0 \leq \lambda_g \leq 1$, $\forall g = 1,2$ and $0 \leq \pi_d \leq 1$, $\forall d = 1,..,m$. Similarly, $\sum_{g=1}^{2} \lambda_g = 1$ and $\sum_{d=1}^{m} \pi_d = 1$.

4.4.3 Destroy operators

As Algorithm 4.9 shows, if selected goal is to partially destroy routes, a method to remove nodes from solution is called. In line 17 of Algorithm 4.9, the function `SelectDestroyOperator` determines a destroy method. The proposed ALNS matheuristic uses six different methods to partially destroy routes. These methods requires initially an empty set $\mathcal{D}$ to store removed nodes from solution and a fixed number of nodes to remove d. A brief description of destroy methods follows:

(*i*) *Nodes in the largest arc and their nearest neighbors (LANN):* given a solution s, LANN randomly picks up a route in s and seeks for the arc a ($a = (h,t)$) with largest traveling cost within the route. Then, h and t are removed from the route and stored in $\mathcal{D}$. To complete the number of nodes to destroy, the $\lceil \frac{d-2}{2} \rceil$ nearest neighbors to h and are also stored in $\mathcal{D}$ as well as the $\lfloor \frac{d-2}{2} \rfloor$ nearest neighbors to t.

(*ii*) *Nodes in random arc and their nearest neighbors (RANN):* this method works in a similar way to LANN. However, RANN randomly chooses arc $a = (h,t)$. Then, h, t and their nearest nodes are removed from the route and stored in $\mathcal{D}$ as in LANN.

(*iii*) *Nodes in random arc and their nearest neighbors with large demand (RANNLD):* RANNLD picks up an arc $a = (h,t)$ in a random way. Then, h, t and their nearest nodes are removed from solution and stored in $\mathcal{D}$ as in LANN and RANN. Nonetheless, an additional condition is imposed to choose a near neighbor: pickup or delivery quantities must be large and similar to those in nodes h or t. Therefore, the i-th nearest neighbor to t (t') is selected if $|q_t + q_{t'}| > Q$. The sorted list of nearest nodes to t is explored until $\lfloor \frac{d-2}{2} \rfloor$ nodes met the described condition. In a similar way, the j-th nearest neighbor to h (h') is selected only if $|q_h + q_{h'}| > Q$ and $\lceil \frac{d-2}{2} \rceil$ nodes are stored in $\mathcal{D}$.

The intuition behind the idea of RANNLD relies on the fact that demand at nodes stored in $\mathcal{D}$ is large. For large values on demands, local search operators in intensification procedures for ALNS matheuristic, as *string relocation* and *string exchange* barely finds different and promising positions for these nodes. As described in equation (4.22) and in properties in Section 4.3.2 nodes selected as candidates for LS operations depend on their demand. Therefore, with this destroy method, it is expected to find new positions for nodes with large demand even when local search procedures are not able to provide one.

(*iv*) *Nodes in largest arcs (LA):* this method selects a route from solution s at random. From the selected route, arcs are sorted in non-increasing order of their length. Next, preserving such order, head and tail nodes from each arc are added to $\mathcal{D}$ until $|\mathcal{D}| = d$.

(*v*) *Adjacent nodes to random arc (ANRA):* ANRA randomly chooses an arc $a = (h,t)$ from solution s. Firstly, nodes h and t are added to $\mathcal{D}$. Next, nodes connected to elements in $\mathcal{D}$ are also removed from s until $|\mathcal{D}| = d$. Figure 4.5 depicts the order in which nodes are added to $\mathcal{D}$. If $a = (4,5)$, then nodes 3 and 6 are removed from s and added to $\mathcal{D}$. Next candidates are nodes 2 and 6. As example, if $d = 5$, a node from these two candidates is selected randomly to fill $\mathcal{D}$.

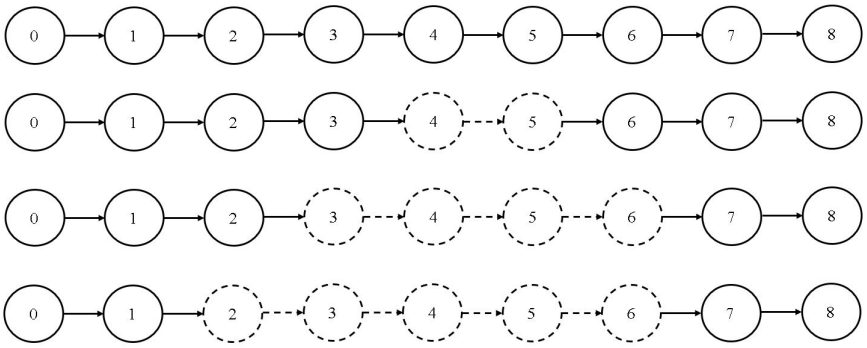


Figure 4.5: Removing adjacent nodes to a random arc

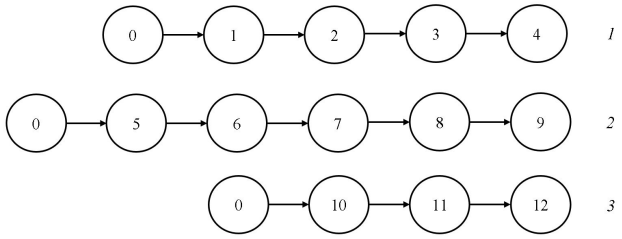
(*vi*) *Random nodes (RND):* the last proposed method selects d nodes at random to remove from s and added to $\mathcal{D}$.

4.4.4 A MILP as repair operator

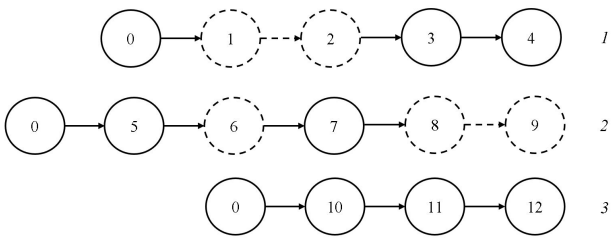
After a destroy method is selected and removed nodes from a solution are stored in $\mathcal{D}$, these nodes must be added again to solution via repair method. As mentioned before, an MILP is used as repair

method by adding removed nodes to solution minimizing total traveling costs. The mathematical model is solved for a subset $\mathcal{K}'$ of routes. $\mathcal{K}'$ is the set of routes where at least one node was previously removed with any of the six available destroy methods. Moreover, $\mathcal{F}_k$ is defined as the set of not removed nodes from route k ($k \in \mathcal{K}'$). Figure 4.6 depicts an example on sets definitions for repair MILP. If a three-route solution as the one shown in Figure 4.6a is destroyed via any of the previously described methods, nodes in dotted lines are stored in $\mathcal{D}$ (see Figure 4.6b). Finally, following Figure 4.6c, sets $\mathcal{F}_1 = \{0,3,4\}$ and $\mathcal{F}_2 = \{0,5,7\}$ where $\mathcal{K}' = \{1,2\}$.

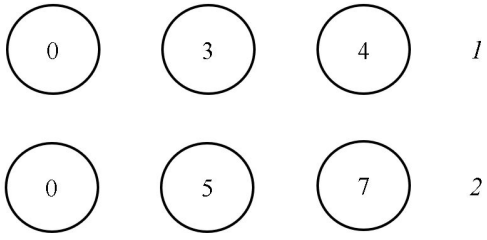
Some parameters (c_{ij}, q_i and Q) and decision variables (y_{ij}, l_{ij} and z_{ij}) required in MILP as repair method were defined for 1–PDVRP MILP (equations (4.1) to (4.12)). New parameters for the proposed formulation include $\bar{f}$ and TL as the total traveling cost of the original solution to repair and the maximum route length, respectively. Moreover, additional decision variables (p^k_{ij}) are required. Variable p^k_{ij} takes the value of one if node i is visited in any position before node j with vehicle k. With the example in 4.6c, variables $p^1_{0,3}$, $p^1_{0,4}$, $p^2_{0,5}$ and $p^2_{0,7}$ are equal to one. The mathematical model to repair partially destroyed routes, follows:



(a) Solution before destroy procedure



(b) Solution with selected nodes to destroy



(c) $\mathcal{F}_k$ ($k = 1,2$) after destroy procedure

Figure 4.6: Example on repair MILP sets

$$\min f = \sum_{k\in\mathcal{K}'}\sum_{i\in\mathcal{F}_k\cup\mathcal{D}}\sum_{j\in\mathcal{F}_k\cup\mathcal{D}} c_{ij}\cdot y_{ij}^k \tag{4.50}$$

subject to,

$$\sum_{k\in\mathcal{K}'}\sum_{i\in\mathcal{F}_k\cup\mathcal{D}}\sum_{j\in\mathcal{F}_k\cup\mathcal{D}} c_{ij}\cdot y_{ij}^k \le \bar{f}, \tag{4.51}$$

$$\sum_{i\in\mathcal{F}_k\cup\mathcal{D}}\sum_{j\in\mathcal{F}_k\cup\mathcal{D}} c_{ij}\cdot y_{ij}^k \le TL, \qquad \forall\, k\in\mathcal{K}' \tag{4.52}$$

$$\sum_{j\in\mathcal{F}_k\cup\mathcal{D}} l_{ji}^k - \sum_{j\in\mathcal{F}_k\cup\mathcal{D}} l_{ij}^k = q_i \sum_{j\in\mathcal{F}_k\cup\mathcal{D}} y_{ij}^k, \qquad \forall\, k\in\mathcal{K}', i\in\mathcal{F}_k\cup\mathcal{D}\backslash\{0\} \tag{4.53}$$

$$l_{ij}^k \le Q\cdot y_{ij}, \qquad \forall\, k\in\mathcal{K}', i,j\in\mathcal{F}_k\cup\mathcal{D} \tag{4.54}$$

$$p_{0j}^k = 1, \qquad \forall\, k\in\mathcal{K}', j\in\mathcal{F}_k\cup\mathcal{D}\backslash\{0\} \tag{4.55}$$

$$p_{ij}^k + p_{ji}^k \le 1, \qquad \forall\, k\in\mathcal{K}', i\in\mathcal{F}_k, j\in\mathcal{D} \tag{4.56}$$

$$y_{ij}^k \le p_{ij}^k, \qquad \forall\, k\in\mathcal{K}', i,j\in\mathcal{F}_k\cup\mathcal{D} \tag{4.57}$$

$$p_{ij}^k = 1, \qquad \forall\, k\in\mathcal{K}', i,j\in\mathcal{F}_k : i<j \tag{4.58}$$

$$\sum_{j\in\mathcal{F}_k\cup\mathcal{D}} y_{ij}^k = 1, \qquad \forall\, k\in\mathcal{K}', i\in\mathcal{F}_k \tag{4.59}$$

$$\sum_{k\in\mathcal{K}'}\sum_{j\in\mathcal{F}_k\cup\mathcal{D}} y_{ij}^k = 1, \qquad \forall\, i\in\mathcal{D} \tag{4.60}$$

$$\sum_{k\in\mathcal{K}'}\sum_{j\in\mathcal{F}_k\cup\mathcal{D}} y_{ji}^k = 1, \qquad \forall\, i\in\mathcal{D} \tag{4.61}$$

$$\sum_{j\in\mathcal{F}_k\cup\mathcal{D}} y_{ij}^k = \sum_{j\in\mathcal{F}_k\cup\mathcal{D}} y_{ji}^k, \qquad \forall\, k\in\mathcal{K}', i\in\mathcal{F}_k \tag{4.62}$$

$$y_{i,i+1}^k + \sum_{j\in\mathcal{D}} y_{ij}^k = 1, \qquad \forall\, k\in\mathcal{K}', i\in\mathcal{F}_k\backslash\{|\mathcal{F}_k|\} \tag{4.63}$$

$$y_{|\mathcal{F}_k|,0}^k + \sum_{j\in\mathcal{D}} y_{|\mathcal{F}_k|,j}^k = 1, \qquad \forall\, k\in\mathcal{K}' \tag{4.64}$$

$$\sum_{j\in\mathcal{F}_k\cup\mathcal{D}} z_{ji}^k - \sum_{j\in\mathcal{F}_k\cup\mathcal{D}} z_{ij}^k = \sum_{j\in\mathcal{F}_k\cup\mathcal{D}} y_{ij}^k, \qquad \forall\, k\in\mathcal{K}', i\in\mathcal{F}_k\cup\mathcal{D}\backslash\{0\} \tag{4.65}$$

$$z_{ij}^k \le (|\mathcal{F}_k| + |\mathcal{D}|)\cdot y_{ij}^k, \qquad \forall\, k\in\mathcal{K}', i,j\in\mathcal{F}_k\cup\mathcal{D} \tag{4.66}$$

$$y_{ij}^k,\ p_{ij}^k \in \{0,1\}, \qquad \forall\, k\in\mathcal{K}', i,j\in\mathcal{F}_k\cup\mathcal{D} \tag{4.67}$$

$$l_{ij}^k,\ z_{ij}^k \ge 0, \qquad \forall\, k\in\mathcal{K}', i,j\in\mathcal{F}_k\cup\mathcal{D} \tag{4.68}$$

The objective function in (4.50) minimizes the total traveling cost for routes in $\mathcal{K}'$. Constraint in (4.51) avoids a repaired solution with a large value for total traveling cost. With (4.52), repaired

routes must not exceed maximum route length. Expression in (4.53) and (4.54) force the model to met demand at each node and vehicle capacity, respectively. Constraints (4.55) to (4.58) set precedence relation between nodes within routes in $\mathcal{K}'$. Note that expressions in (4.56) fix a position for each node in $\mathcal{D}$ before or after each node in $\mathcal{F}_k$. In (4.58) all precedence relations between nodes in $\mathcal{F}_k$ are fixed. Therefore, the order in which nodes in $\mathcal{F}_k$ are visited does not change in solution delivered by MILP. Expressions in (4.59) to (4.62) are routing constraints for nodes in $\mathcal{F}_k$ and $\mathcal{D}$. Equations in (4.63) and (4.64) ensure that arcs in the new solution starting in a node from $\mathcal{F}_k$ end in its contiguous node in $\mathcal{F}_k$ or in any node from $\mathcal{D}$. With constraints in (4.65) and (4.66) subtours are forbidden in final solution. Lastly, (4.67) and (4.68) describe the decision variables domain.

4.4.5 Removing a route: MILP approach

Following Algorithm 4.9, if selected goal aims to improve solution removing a complete route then function `RemoveRoute` is called. This function searches for the route with the lowest cost (r') and attempts to eliminate it from the solution by assigning its nodes to the others paths in the solution. To do so, an MILP is solved to check whether the goal can be met. The intuition behind this MILP is similar to the one presented in Section 4.4.4; however, set $\mathcal{D}$ is composed by all nodes in r'. To determine if it is feasible to remove a route, set $\mathcal{K}'$ contains now routes in $\mathcal{K}$ except for r' ($\mathcal{K}' = \mathcal{K} - \{r'\}$). Therefore, the objective of MILP is to determine whether adding new nodes in routes $\mathcal{K}'$ do not exceed vehicle capacity. To do so, variables s^k describe the amount of load that exceed vehicle capacity for route k. Variables l^{-k} and l^{+k} denote the minimum and maximum load of vehicle k ($k \in \mathcal{K}'$). Since the proposed MILP to destroy a route completely requires most of the constraints described in Section 4.4.4, let Ω be set of constraints (4.51) to (4.68) except for expression in (4.54) which are slightly modified in order to check feasibility once r' is removed. The proposed MILP can be described as:

$$\min f = \sum_{k \in \mathcal{K}'} s^k \tag{4.69}$$

subject to,

$$l^{-k} \leq l^k_{ij} \leq l^{+k}, \qquad \forall\, k \in \mathcal{K}', i, j \in \mathcal{F}_k \cup \mathcal{D} \tag{4.70}$$

$$l^{+k} - l^{-k} \leq Q + s^k, \qquad \forall\, k \in \mathcal{K}' \tag{4.71}$$

$$\Omega, \tag{4.72}$$

$$l^{-k}, l^{+k}, s^k \geq 0, \qquad \forall\, k \in \mathcal{K}' \tag{4.73}$$

Expression (4.69) minimizes vehicle capacity violation. If $f = 0$, then it is feasible to remove route r' from solution. Otherwise, r' must be preserved to keep feasibility of solution. Constraints in (4.70) compute values for minimum and maximum loads. In (4.71) vehicle capacity is checked for set $\mathcal{K}'$. Lastly, expressions (4.73) denote the decision variables domain.

4.5 An enumeration algorithm for solution repairing

The matheuristic algorithm presented in Section 4.4 is based on two MILPs for repairing solutions. One of these MILPs aims to relocate nodes through a subset of routes ($\mathcal{K}'$) while the second one attempts to reduce the total traveling costs by removing a route completely. This section describes an *enumeration algorithm* able to replace MILPs as repairing operations. The algorithm is based on a recursive function that checks for each position in the incumbent solution, feasible insertions of nodes in $\mathcal{D}$ preserving precedence relations between nodes in $\mathcal{F}_k$ ($k \in \mathcal{K}$). A general structure of the algorithm as well as a description of some of the key components as lower bounds computation follows.

4.5.1 General structure

The repairing process can be summarized as in Algorithm 4.10. This algorithm requires three inputs: solution s, a set of nodes to insert $\mathcal{D}$ (i.e., removed nodes from s) and, set of fixed (i.e., not removed) nodes from each route k, $\mathcal{F}_k$. In line 2, a list of nodes T is defined as an empty list. T stores the list of nodes to visit in current best order after the repair solution recursive algorithm is executed. The recursive function also requires a list of candidates $\mathcal{C}$. This list is updated at each iteration p and stores nodes in $\mathcal{D}$ not yet relocated in T as well as the candidate node in $\mathcal{F}_k$. It is worth to recall that similarly to MILPs for repairing, this algorithm ensures that all arcs starting in a node from $\mathcal{F}_k$ end up in its contiguous node in $\mathcal{F}_k$ or in any node from $\mathcal{D}$. In Line 3, a list of nodes F is retrieved

Algorithm 4.10: Enumeration algorithm for solution repairing

1: **Enumeration algorithm** $(s, \mathcal{D}, \mathcal{F}_1, .., \mathcal{F}_{|\mathcal{K}|})$
2: $T \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset$
3: $F \leftarrow$`Concatenate`$(\mathcal{F}_1, .., \mathcal{F}_{|\mathcal{K}|})$
4: $T^* \leftarrow$`Concatenate`(s)
5: $T \leftarrow$`RepairSolution`$(0, T, F, \mathcal{C})$
6: $s \leftarrow$`getSolution`(T)
7: **return** s

from a function `Concatenate` where new arcs are created from the node in the last position of $\mathcal{F}_k$ to the first node in $\mathcal{F}_{k+1}$ for all $k = 1,..,|\mathcal{K}|-1$. Similarly, in line 4, function `Concatenate` creates new arcs from last node in each route in s to the first one (depot) in the next path in s and stores the ordered list of nodes in T^*. To add nodes to T, `RepairSolution` function is called in line 5. This function requires a position p where a new node is added and the lists T^*, T, F and, $\mathcal{C}$. Initially, $p = 0$ to set the first node in the solution. Finally, once the solution is repaired (i.e., all nodes were added to T), `getSolution` turns list T in a 1–PDVRPLC solution.

The repair solution algorithm (`RepairSolution` function) is depicted in Algorithm 4.11. In lines 2 to 4, node 0 (depot) is assigned to the new solution. Since the solution only contains depot, traveling cost (*cost*) is zero and demand at depot is assigned as vehicle load and as the minimum and maximum partial load for the route. From the depot it is possible to create an arc to any of the nodes in $\mathcal{D}$ or to node in first position of F ($F_{[1]}$). The algorithm counts the number of nodes in F already added to T with variable j. Once information for first position is updated, function `RepairSolution` is called in a recursive way.

For second position onward ($p \geq 1$), the algorithm evaluates each one of the candidates for that position within the loop for starting in line 7. The algorithm computes the cost of traveling from node in position $p-1$ to the candidate node (lines 8 and 9) and, in line 10, the algorithm determines whether the partial solution $T \cup \mathcal{C}_{[i]}$ *dominates* the best solution found (T^*). To do so, a dominance rule is proposed and it is described in next section. If partial solution may end up with and improvement on best solution found, the algorithm updates loads, checks for load feasibility and assigns a new node to partial solution T as follows: if the candidate to add to T is not the depot, then vehicle load takes into account $\mathcal{C}_{[i]}$ demand as in line 12. Similarly, minimum and maximum loads for partial solution are updated in lines 13 and 14, respectively. If load does not exceed the vehicle capacity, then the candidate node is added to partial solution (line 16) and it is also removed from the list of candidates. When selected node is not in $\mathcal{D}$ but in F, then the number of fixed nodes already added to T increases by one and a new candidate is added to $\mathcal{C}$ (i.e., next node in F) (see lines 18 to 21). The `RepairSolution` function is called while the candidate list stores at least one node. If no candidate is available, thus a complete solution has been constructed. Given the dominance rule in the algorithm, each time a complete solution is found, an improvement on best solution found is also made. For any evaluated position, if selected candidate is the depot (line 27), that means that a new route is created, therefore, depot is assigned to T, and values for load, minimum and maximum loads are computed using demand at depot (see lines 28 and 2). Finally, once all candidates for each position are evaluated, the algorithm returns T^*.

Algorithm 4.11: Repair solution algorithm

function REPAIRSOLUTION($p, T^*, T, F, \mathcal{C}$)
  if $p = 0$ **then**
    $\mathcal{C} \leftarrow \mathcal{D} \cup F_{[1]}, T_{[p]} \leftarrow 0, cost_{[p]} \leftarrow 0,$
    $l_{[p]} \leftarrow -q_0, l^+_{[p]} \leftarrow -q_0, l^-_{[p]} \leftarrow -q_0, j \leftarrow 1$
    `RepairSolution`$(1, T^*, T, F, \mathcal{C})$
  else
    for $i = 1$ to $|\mathcal{C}|$ **do**
      $m \leftarrow T_{[p-1]}, n \leftarrow \mathcal{C}_{[i]}$
      $cost_{[p]} \leftarrow cost_{[p-1]} + c_{m,n}$
      if $T \cup \mathcal{C}_{[i]} \succeq T^*$ **then**
        if $\mathcal{C}_{[i]} \neq 0$ **then**
          $l_{[p]} \leftarrow L_{[p-1]} - q_{\mathcal{C}_{[i]}}$
          $l^+_{[p]} \leftarrow \max\{l^+_{[p-1]}, l_{[p-1]} + q_{\mathcal{C}_{[i]}}\}$
          $l^-_{[p]} \leftarrow \min\{l^-_{[p-1]}, l_{[p-1]} + q_{\mathcal{C}_{[i]}}\}$
          if $l^+_{[p]} - l^-_{[p]} \leq Q$ **then**
            $T_{[p]} \leftarrow \mathcal{C}_{[i]}$
            $\mathcal{C} \leftarrow \mathcal{C} \backslash \{\mathcal{C}_{[i]}\}$
            if $\mathcal{C}_{[i]} = F_{[j]}$ **then**
              $j \leftarrow j + 1$
              $\mathcal{C} \leftarrow \mathcal{C} \cup F_{[j]}$
            end if
            if $|\mathcal{C}| > 0$ **then**
              `RepairSolution`$(p + 1, T^*, T, F, \mathcal{C})$
            else
              $T^* \leftarrow T$
            end if
          else
            $T_{[p]} \leftarrow 0$
            $l_{[p]} \leftarrow -q_0, l^+_{[p]} \leftarrow -q_0, l^-_{[p]} \leftarrow -q_0$
            `RepairSolution`$(p + 1, T^*, T, F, \mathcal{C})$
          end if
        end if
      end if
    end for
  end if
  return T^*
end function

4.5.2 Dominance rules for partial solutions

Line 10 in Algorithm 4.11 evaluates whether the partial solution $T \cup \mathcal{C}_{[i]}$ *dominates* the best found solution (T^*). For the enumeration algorithm, *dominance* is based on two criteria: maximum route length feasibility and total traveling cost. Thus, instructions in lines 11 to 32 are executed only if adding candidate node in $\mathcal{C}_{[i]}$ to T (i.e., $T \cup \mathcal{C}_{[i]}$), the allowed maximum route length can be achieved and also, objective function value does not increase.

To estimate whether route length constraints may be violated or total traveling cost for a partial solution may end up in a greater value than objective function for T^*, information in lists F and T is used. Firstly, a lower bound for length of route k is computed over nodes in $\mathcal{F}_k$ and not yet added to T as follows:

$$RLB_p = \sum_{i=p+1}^{|\mathcal{F}_k|-1} c_{[F_i],[F_{i+1}]}, \qquad \forall\ p = 1,..,|T^*| \tag{4.74}$$

RLB_p denotes the fixed cost not yet added to traveling cost for route k in a partial solution constructed up to position p. Note that this cost is computed over nodes not previously removed (i.e., destroyed arcs) from original solution s. Therefore, if $cost[p] + RLB_p > TL$ then partial solution violates maximum duration constraints.

Secondly, and in a similar way to equation in (4.74), a lower bound is computed for total traveling cost. SLB_p denotes the fixed traveling cost not yet added to partial solution constructed up to position p. Values for SLB_p are obtained as the sum of traveling costs for all arcs in F and not yet added to T as follows:

$$SLB_p = \sum_{i=p+1}^{|F|-1} c_{[F_i],[F_{i+1}]}, \qquad \forall\ p = 1,..,|T^*| \tag{4.75}$$

Since SLB_p denotes the total traveling cost for fixed arcs (i.e., not destroyed arcs) in a complete solution, if $cost[p] + SLB_p \geq f(T^*)$ then, partial solution does not improve best found solution T^*.

It is worth to mention that for any position p there may be also some nodes in $\mathcal{D}$ not yet added to solution T (i.e., $\mathcal{D} \cap \mathcal{C}$). Therefore, total traveling cost for a partial solution will increase at least $\sum_{i \in \mathcal{D} \cap \mathcal{C}} \min_{j \in \mathcal{N}}\{c_{ij}\}$ units and SLB_p can be computed in a more accurate way as:

$$SLB_p = \sum_{i=p+1}^{|F|-1} c_{[F_i],[F_{i+1}]} + \sum_{i \in \mathcal{D} \cap \mathcal{C}} \min_{j \in \mathcal{N}}\{c_{ij}\}, \qquad \forall p = 1,..,|T^*| \tag{4.76}$$

Following equations (4.74) and (4.76), partial solution $T \cup \mathcal{C}_{[i]}$ in line 10 of Algorithm 4.11 *dominates* best found solution T^* if and only if $cost[p] + RLB_p \leq TL$ and $cost[p] + SLB_p < f(T^*)$.

4.6 Computational experiments

This section presents the main results on the proposed MILP and matheuristic algorithms for the 1–PDVRPLC. Firstly, experiments based on MILP provide some insights about how the number of nodes, number of vehicles and vehicle capacity are key factors to determine whether it is possible to reduce imbalance throughout the route costs. Next, experiments with matheuristic strategies are described and results for small and large 1–PDVRPLC instances are outlined. Moreover, results on matheuristic algorithms are compared with those reported in Shi et al. (2009). Before general results on MILP and matheuristic algorithms are presented, a brief description on experiments and instances used to test both approaches, follows.

4.6.1 Instances and experiments configuration

To test the performance of the mathematical model described in Section 4.2.2 for the 1–PDVRPLC, some of the benchmark instances described in 3.4.1 are solved. MILP experiments for the 1–PDVRPLC include instances with $|\mathcal{K}| = \{2,3\}$ and $|\mathcal{N}| = \{20,30,40\}$. Additionally, three different values for the vehicle capacity are defined: $Q = \{10,20,40\}$ and three values for α: 0.5, 0.8 and $|\mathcal{K}| - 1$. For each size of the problem, number of vehicles and each value of Q and α, 10 instances were solved (i.e, a complete set of 540 instances). To code the objective function described in (4.16), M is fixed to $c_{max} \cdot |\mathcal{N}| \cdot |\mathcal{K}|$ where $c_{max} = \max_{(i,j)\in\mathcal{A}}\{c_{ij}\}$. Recall that T requires an upper bound for the cost of a single route. As mentioned in Section 4.2.2, this can be computed as the optimal solution of the corresponding 1–PDTSP (f^*_{PDTSP}). The optimal solution for 1–PDTSP instances with up to 60 nodes are currently reported in Hernández-Pérez et al. (2009) and Palacio and Rivera (2022). To conduct the computational experiments on MILP, Gurobi optimizer 8.1.1 runs in an Intel Core i7 with 64 gigabytes of RAM running under debian 8 (x86-64). For each running of the models, a maximum computational time of one hour (3600 s) is set.

For experiments based on matheuristic algorithms, and due to the large number of runs, only a small subset of instances described in Section 3.4.1 are solved. Following the ideas of Shi et al. (2009), only one instance for each size of the problem is solved. Nonetheless, four different values are tested for route length on these instances. Indeed, one of these values allows to compare results on matheuristic algorithms with those reported in Shi et al. (2009). Computational experiments on

matheuristic algorithms were conducted on a on an Intel Core i7 at 1.80GHz with 16.00 gigabytes of RAM. To solve MILPs, Gurobi 9.1 was used. Mathematical models and algorithms were coded in C++ within Visual studio 2019 interface. Each running of the hybrid algorithms, a maximum computational time of one hour (3600 s) is set. After several tests, values for parameters `MaxStarts` and `MaxIterations` were set to 5 and 20, respectively. In a similar way, for ALNS algorithms the number of nodes to remove in destroy methods (d) was set to 5.

4.6.2 Results on mixed integer lineal model

Table 4.1 presents the results on 1–PDVRPLC instances. For each subset of instances based on different values for $|\mathcal{K}|$, $|\mathcal{N}|$, and Q, the table summarizes the MILP performance in three different categories. Columns Opt. presents, for each set of corresponding instances, the percentage of instances in which optimal solution is found within the predefined computational time. For these subset of instances decision variable λ is equal to zero, which means that objective function in (4.16) is not penalized (i.e. it is feasible). In columns Best known, we report the percentage of instances without an optimality certificate and the average gap when the time limit condition is met. Finally, for instances in columns Best unfeas., MILP is able to find an optimal solution but it does not satisfy maximum tour length for the given α. In these cases, decision variable λ takes a value greater than zero and consequently, the objective function in (4.16) is penalized. It is worth to mention that final lower bounds reported for instances in columns Best known do not include penalization values. Thus, it is possible to conclude that all instances in which variable λ would take a value greater than zero are grouped in columns Best unfeas. Precisely, from the whole set of 540 instances, only five of them are not feasible when tour length constraints are imposed. Those instances come from the subset of problems with three vehicles, 20 nodes and vehicle capacity equal to 20 and 40. For example, if vehicle capacity is 20, there exists one instance in which the optimal solution found violates the tour length constraint by exceeding its limit by 11 units if $\alpha = 0.5$.

As expected, in columns Opt. the overall number of optimal solutions decreases as the number of nodes increases. In a similar way, the overall gap for instances in columns Best known, increases as the number of vehicles also increases. For instances with 30 and 40 nodes, vehicle capacity impacts on the average gap and the number of optimal solutions found. For each size of the problem, the average gap for best known solutions decreases and the number of optimal solution become larger as the vehicle capacity increases. Particularly, for all the instances with 40 nodes and the tightest vehicle capacity, it was not possible to find the optimal solution within the maximum computational time. Nevertheless, number of optimal solutions begins to increase as the vehicle

Table 4.1: Results on 1–PDVRPLC instances

$\|\mathcal{K}\|$	$\|\mathcal{N}\|$	Q	$\alpha = 0.5$			$\alpha = 0.8$			$\alpha = \|\mathcal{K}\| - 1$		
			Opt.	Best known	Best unfeas.	Opt.	Best known	Best unfeas.	Opt.	Best known	Best unfeas.
		10	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%
	20	20	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%
		40	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%
		10	90%	10% (0.30%)	0%	80%	20% (0.83%)	0%	90%	10% (0.24%)	0%
2	30	20	90%	10% (0.37%)	0%	90%	10% (0.82%)	0%	90%	10% (0.36%)	0%
		40	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%
		10	0%	100% (5.50%)	0%	0%	100% (6.30%)	0%	0%	100% (5.02%)	0%
	40	20	80%	20% (0.55%)	0%	80%	20% (0.82%)	0%	90%	10% (0.41%)	0%
		40	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%
		10	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%
	20	20	80%	10% (0.81%)	10%[a]	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%
		40	60%	0% (0.00%)	40%[b]	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%
		10	30%	70% (4.03%)	0%	50%	50% (2.38%)	0%	60%	40% (1.45%)	0%
3	30	20	50%	50% (3.30%)	0%	90%	10% (0.82%)	0%	90%	10% (1.02%)	0%
		40	50%	50% (4.57%)	0%	90%	10% (0.08%)	0%	100%	0% (0.00%)	0%
		10	0%	100% (13.51%)	0%	0%	100% (11.60%)	0%	0%	100% (9.47%)	0%
	40	20	0%	100% (9.62%)	0%	50%	50% (1.89%)	0%	50%	50% (2.27%)	0%
		40	0%	100% (8.51%)	0%	100%	0% (0.00%)	0%	100%	0% (0.00%)	0%

[a] One instance with $\lambda = 11$

[b] Four instances with $\lambda = 65, 73, 41$ and 22

capacity is larger. Values for α determine whether tour length constraints can be met. As mentioned before, the tightest tested value for α causes unfeasible solutions if three vehicles perform the pickup and delivery operations. As values for α increase, less computational effort is required to solve instances and average optimality gaps become to decrease.

Table 4.2 presents the average cost deterioration when α value is less than $|\mathcal{K}| - 1$. Since $\alpha = |\mathcal{K}| - 1$ allows the model to find solution with no maximum tour length constraints, a variation on cost (Δf_α) when those constraints are imposed (i.e. for $\alpha = \{0.5, 0.8\}$) is computed as follows.

$$\Delta f_\alpha = \frac{f_\alpha - f_{|\mathcal{K}|-1}}{f_\alpha}$$

In general, the average cost variation increases for each size of the problem as long as the vehicle capacity also increases. However, in a worst-case scenario, for instances with 40 nodes and the largest vehicle capacity, the total cost does not increase more than 9.16% on average.

Finally, average computational time required to solve the 1–PDVRPLC are reported in Table

Table 4.2: Average cost increase based on $\alpha = |\mathcal{K}| - 1$

		$\|\mathcal{K}\| = 2$		$\|\mathcal{K}\| = 3$	
		α		α	
$\|\mathcal{N}\|$	Q	0.5	0.8	0.5	0.8
20	10	0.37%	0.21%	1.83%	0.68%
	20	1.75%	1.59%	7.63%	2.75%
	40	3.40%	2.32%	9.60%	3.94%
30	10	0.24%	0.25%	2.22%	0.44%
	20	0.96%	1.33%	5.07%	0.89%
	30	1.79%	1.76%	8.96%	1.69%
40	10	0.32%	1.00%	4.11%	2.11%
	20	0.37%	0.27%	6.33%	0.00%
	40	1.71%	0.96%	9.16%	1.84%

4.3. Needless to say, instances with 40 nodes and three vehicles are the hardest subset of problems to solve. In this subset, for those instances with the tightest vehicle capacity or $\alpha = 0.5$ no optimal solution was found. Not only number of nodes but vehicle capacity are key factors on computational time behavior; the tightest vehicle capacity, the highest computational effort.

Table 4.3: Average CPU time (s) for the 1-PDVRPRL MILP

		$\|\mathcal{K}\| = 2$			$\|\mathcal{K}\| = 3$		
		α			α		
$\|\mathcal{N}\|$	Q	0.5	0.8	$\|\mathcal{K}\| - 1$	0.5	0.8	$\|\mathcal{K}\| - 1$
20	10	69.56	48.74	33.48	519.45	347.42	203.04
	20	20.83	17.61	9.54	1001.21	430.05	94.63
	40	7.98	7.65	4.43	691.07	259.26	35.88
30	10	1255.80	1279.41	1099.82	3226.23	2785.46	2281.30
	20	432.55	458.64	476.38	2631.94	1094.63	710.54
	30	40.72	46.52	31.42	2450.60	651.64	269.74
40	10	3600.00	3600.00	3600.00	3600.00	3600.00	3600.00
	20	958.32	966.69	913.44	3600.00	2633.21	2343.10
	40	152.79	103.64	73.16	3600.00	1830.58	493.72

4.6.3 Matheuristic algorithms: comparative results

This section summarizes results on proposed matheuristic algorithms for the 1–PDVRPLC. Firstly, for small instances with up to 40 nodes, the algorithms performance is compared with MILP proposed in Section 4.2.2. For larger instances, metaheuristic algorithms are compared to each other

when several values for route length are tested. Finally, this section also compares the proposed algorithms with a genetic algorithm for the 1–PDVRPLC proposed in Shi et al. (2009).

Tables 4.4, 4.5 and, 4.6 summarize the main results for the three LNS-based algorithms when instances with 20, 30 and 40 are solved, respectively. For each size of the problem, results on MILP, MS–ILNS and ALNS algorithms are reported. For the sake of clarity, hereafter adaptive strategies based on MILPs and enumeration algorithm as repairing methods are called IALNS-Math and IALNS-EA, respectively. For each one of these algorithms, rows Best, # Best and Avg., reports best solution found, number of times each algorithm is able to find solution in column Best, and the average value for objective function also over ten runs, respectively. In a similar way, each algorithm is compared with solution obtained solving MILP via commercial solver. Therefore, four gaps are computed as follows:

$$Min.\ gap_{UB}(\%) = \frac{Best - UB}{Best} \cdot 100 \tag{4.77}$$

$$Avg.\ gap_{UB}(\%) = \frac{Avg. - UB}{Avg.} \cdot 100 \tag{4.78}$$

$$Min.\ gap_{LB}(\%) = \frac{Best - LB}{Best} \cdot 100 \tag{4.79}$$

$$Avg.\ gap_{LB}(\%) = \frac{Avg. - LB}{Avg.} \cdot 100 \tag{4.80}$$

For instances with 20 nodes, Table 4.4 shows that it is possible to find the optimal solution via any of the three proposed algorithms. For each fleet size (i.e., $|\mathcal{K}| = \{2,3\}$) the number of times that the optimal solution is found via any of three algorithms, increases as α value also increases. Despite the three algorithms find the optimal solution in at least 60% of runs, IALNS-EA is able to retrieve optimal solutions more frequently than MS-ILNS and IALNS-Math, on average. Table 4.5 shows that for instances with three vehicles, MILP is not able to retrieve optimal solutions. Therefore, the optimizer reports gaps up to 4.76%. MS–ILNS algorithm finds up to five times the solution reported by MILP and lower bound average gaps ($Avg.gap_{LB}$) vary up to 5.71%. Nonetheless, For those instances, IALNS-Math and IALNS-EA are able to improve up to six times out of ten runs, the solution delivered via commercial solver. Improvements up to 2.00%, 0.32% and 1.00% are reported solving instances via IALNS-Math with $\alpha = 0.5, 0.8$ and 1, respectively. Similar results are obtained when $\alpha = 0.5$ for instances with two and three vehicles and 40 nodes (see Table 4.6). All algorithms provides at least once, the solution reported by MILP as the best solution found. Once again, for instances with $\alpha = 0.5$, IALNS-Math and IALNS-EA retrieve improved UB when they are compared with those reported vian MILP. Thus improvements on UBs ($Min.gap_{UB}$ up to 1.99% are reported with both adaptive LNS strategies.

Table 4.4: Comparative results on LNS-based algorithms for instances with $|\mathcal{N}| = 20$

Solution strategy	$\|\mathcal{K}\|$ / α	2			3		
		0.5	0.8	1	0.5	0.8	1
MILP	UB	4866	4866	4866	4988	4966	4883
	LB	4866	4866	4866	4988	4966	4883
	gap	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
MS–ILNS	Best	4866	4866	4866	4988	4966	4883
	# Best	7	8	10	6	7	7
	Avg.	4903.82	4900.12	4866.00	5101.45	5111.14	5087.16
	$Min.gap_{UB}$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	$Avg.gap_{UB}$	0.77%	0.70%	0.00%	2.22%	2.84%	4.01%
	$Min.gap_{LB}$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	$Avg.gap_{LB}$	0.77%	0.70%	0.00%	2.22%	2.84%	4.01%
IALNS-Math	Best	4866	4866	4866	4988	4966	4883
	# Best	8	8	9	6	6	8
	Avg.	4952.80	4901.75	4896.52	5013.13	5008.43	5051.22
	$Min.gap_{UB}$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	$Avg.gap_{UB}$	1.75%	0.73%	0.62%	0.50%	0.85%	3.33%
	$Min.gap_{LB}$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	$Avg.gap_{LB}$	1.75%	0.73%	0.62%	0.50%	0.85%	3.33%
IALNS-EA	Best	4866	4866	4866	4988	4966	4883
	# Best	7	9	10	6	8	8
	Avg.	4914.71	4898.36	4866	5023.47	5019.98	5046.72
	$Min.gap_{UB}$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	$Avg.gap_{UB}$	0.99%	0.66%	0.00%	0.71%	1.08%	3.24%
	$Min.gap_{LB}$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	$Avg.gap_{LB}$	0.99%	0.66%	0.00%	0.71%	1.08%	3.24%

Table 4.7 depicts results for larger instances (i.e., $|\mathcal{N}| \geq 50$). Since these instances were not solved via commercial solver, results are based on LNS algorithms comparisons. For each problem size, Table 4.7 reports defined route lengths computed following equation (4.15) with $|\mathcal{K}|$ vehicles and an α as route length factor. For all the LNS algorithms, best solution found after ten runs is reported in columns Best. The number of vehicles required to reach total traveling cost in Best is also depicted in columns $|\mathcal{K}|^*$. Since results for small instances in Tables 4.4, 4.5 and, 4.6 show that ALNS-EA outperforms solution quality for MS–ILNS and ALNS-Math, an improvement percentage for the other two algorithms is computed as:

Table 4.5: Comparative results on LNS-based algorithms for instances with $|\mathcal{N}| = 30$

Solution strategy	$\|\mathcal{K}\|$	2			3		
	α	0.5	0.8	1	0.5	0.8	1
MILP	UB	6452	6447	6503	6633	6576	6568
	LB	6452	6447	6503	6317	6424	6465
	gap	0.00%	0.00%	0.00%	4.76%	2.31%	1.57%
MS-ILNS	Best	6452	6447	6503	6633	6576	6568
	# Best	3	3	5	2	3	4
	Avg.	6551.00	6599.42	6574.63	6699.83	6712.33	6590.67
	$Min.gap_{UB}$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	$Avg.gap_{UB}$	1.51%	2.31%	1.09%	1.00%	2.03%	0.34%
	$Min.gap_{LB}$	0.00%	0.00%	0.00%	4.76%	2.31%	1.57%
	$Avg.gap_{LB}$	1.51%	2.31%	1.09%	5.71%	4.30%	1.91%
IALNS-Math	Best	6466	6447	6503	6503	6555	6503
	# Best	5	2	4	5	6	5
	Avg.	6542.70	6600.47	6654.10	6657.37	6631.69	6578.93
	$Min.gap_{UB}$	0.22%	0.00%	0.00%	-2.00%	-0.32%	-1.00%
	$Avg.gap_{UB}$	1.39%	2.33%	2.27%	0.37%	-0.31%	0.17%
	$Min.gap_{LB}$	0.22%	0.00%	0.00%	2.86%	2.00%	0.58%
	$Avg.gap_{LB}$	1.39%	2.33%	2.27%	5.11%	2.01%	1.73%
IALNS-EA	Best	6452	6447	6503	6503	6555	6511
	# Best	1	3	3	4	3	4
	Avg.	6601.12	6584.92	6679.19	6690.01	6708.33	6704.49
	$Min.gap_{UB}$	0.00%	0.00%	0.00%	-2.00%	-0.32%	-0.88%
	$Avg.gap_{UB}$	2.26%	2.09%	2.64%	0.85%	1.97%	2.04%
	$Min.gap_{LB}$	0.00%	0.00%	0.00%	2.86%	2.00%	0.71%
	$Avg.gap_{LB}$	2.26%	2.09%	2.64%	5.58%	4.24%	3.57%

$$ALNS-EAImp._{\cdot MS-ILNS}(\%) = \frac{Best_{MS-ILNS} - Best_{ALNS-EA}}{Best_{ALNS-EA}} \cdot 100 \tag{4.81}$$

$$ALNS-EAImp._{\cdot ALNS-Math}(\%) = \frac{Best_{ALNS-Math} - Best_{ALNS-EA}}{Best_{ALNS-EA}} \cdot 100 \tag{4.82}$$

Average values for best solution found over each size of the problem are also computed. These averages show that, except for $|\mathcal{N}|$=300, ALNS-EA outperforms MS–ILNS and ALNS-Math. This result keeps consistency with algorithms performance for small instances with up to 40 nodes. For some instances (e.g., instance with 50 nodes and $TL = 1498$), more vehicles than those set in $|\mathcal{K}|$ are required to meet total traveling cost reported in Best. It is worth to recall that $|\mathcal{K}|$ is not a constraint

Table 4.6: Comparative results on LNS-based algorithms for instances with $|\mathcal{N}| = 40$

Solution strategy	$\|\mathcal{K}\|$	2			3		
	α	0.5	0.8	1	0.5	0.8	1
MILP	UB	7325	7246	7246	7541	7489	7539
	LB	7030	7006	7006	7113	7115	7117
	gap	4.03%	3.31%	3.31%	5.68%	4.99%	5.60%
MS-ILNS	Best	7325	7246	7246	7541	7489	7539
	# Best	1	3	2	1	1	2
	Avg.	7397.00	7420.47	7382.44	7667.78	7612.25	7623.81
	$Min.gap_{UB}$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
	$Avg.gap_{UB}$	0.97%	2.35%	1.85%	1.65%	1.62%	0.00%
	$Min.gap_{LB}$	4.03%	3.31%	3.31%	5.68%	4.99%	5.60%
	$Avg.gap_{LB}$	4.96%	5.59%	5.10%	7.24%	6.53%	5.60%
IALNS-Math	Best	7285	7314	7470	7394	7489	7539
	# Best	2	2	3	1	1	2
	Avg.	7488.20	7553.84	7616.74	7667.20	7596.27	7648.03
	$Min.gap_{UB}$	-0.55%	0.93%	3.00%	-1.99%	0.00%	0.00%
	$Avg.gap_{UB}$	2.18%	4.08%	4.87%	1.65%	1.41%	1.43%
	$Min.gap_{LB}$	3.50%	4.21%	6.21%	3.80%	4.99%	5.60%
	$Avg.gap_{LB}$	6.12%	7.25%	8.02%	7.23%	6.34%	6.94%
IALNS-EA	Best	7285	7301	7246	7394	7489	7539
	# Best	2	1	2	1	3	3
	Avg.	7573.15	7548.84	7597.48	7614.20	7624.29	7671.56
	$Min.gap_{UB}$	-0.55%	0.75%	0.00%	-1.99%	0.00%	0.00%
	$Avg.gap_{UB}$	3.28%	4.01%	4.63%	0.96%	1.77%	1.73%
	$Min.gap_{LB}$	3.50%	4.04%	3.31%	3.80%	4.99%	5.60%
	$Avg.gap_{LB}$	7.17%	7.19%	7.79%	6.58%	6.68%	7.23%

on this problem. This value only works as a parameter to determine TL and Split algorithm, VNDs procedures and mathematical models to destroy and repair solution do not use $|\mathcal{K}|$ as a limitation to create or improve solutions.

On the other hand, for each size of the instances, six different values for TL were tested. However, for 200 nodes instances and greater some of the results are not reported in Table 4.7. For those instances, non-reported values for TL turn infeasible the problem (i.e., traveling cost from depot to at least one node is greater than TL).

In 2009, Shi et al. (2009) designed a genetic algorithm (GA) for the 1–PDVRPLC. Shi et al. (2009) tested instances with up 500 nodes and $TL = 3000$. Since experiments summarized in Tables

Table 4.7: Comparative results on LNS-based algorithms for instances with $|\mathcal{N}| \geq 50$

$\|\mathcal{N}\|$	α	TL	$\|\mathcal{K}\|$	MS–ILNS		ALNS-Math		ALNS-EA		ALNS-EA Imp. (%)	
				Best	$\|\mathcal{K}\|^*$	Best	$\|\mathcal{K}\|^*$	Best	$\|\mathcal{K}\|^*$	MS-ILNS	ALNS-Math
50	0.5	1498	7	9889	8	9930	8	9826	8	0.64%	1.1%
		2096	5	8245	5	7769	4	7769	4	6.13%	0.0%
	0.8	1797	7	8706	6	8601	5	8588	4	1.37%	0.2%
		2515	5	7700	4	7488	3	7488	3	2.83%	0.0%
	1	1997	7	8245	5	8106	5	8201	5	0.54%	-1.2%
		2795	5	7783	4	7319	3	7272	3	7.03%	0.6%
	Average			8428.00	5.33	8202.17	4.67	8190.67	4.50	2.90%	0.1%
60	0.5	1613	8	11636	8	11833	9	11912	8	-2.32%	-0.7%
		2150	6	9703	6	9756	5	9756	5	-0.54%	0.0%
	0.8	1936	8	10424	7	10061	6	10061	6	3.61%	0.0%
		2580	6	9754	5	9299	4	9299	4	4.89%	0.0%
	1	2151	8	9774	6	10003	6	9774	6	0.00%	2.3%
		2867	6	9494	5	9290	4	9290	4	2.20%	0.0%
	Average			10130.83	6.17	10040.33	5.67	10015.33	5.50	1.15%	0.2%
100	0.5	1765	10	16558	11	16991	11	16695	11	-0.82%	1.8%
		1960	9	15748	10	15482	9	15296	8	2.96%	1.2%
	0.8	2117	10	15247	10	14973	8	14973	8	1.83%	0.0%
		2351	9	14660	8	14726	7	14726	7	-0.45%	0.0%
	1	2353	10	14341	7	14017	7	14017	7	2.31%	0.0%
		2613	9	13950	8	13771	6	13771	6	1.30%	0.0%
	Average			15084.00	9.00	14993.33	8.00	14913.00	7.83	1.15%	0.5%
200	0.5	1556	17	31559	25	31096	21	31096	21	1.49%	0.0%
	0.8	1588	20	31086	24	32047	22	31086	24	0.00%	3.1%
		1867	17	27348	18	28262	18	27915	18	-2.03%	1.2%
	1	1765	20	28467	19	29223	18	28601	19	-0.47%	2.2%
		2075	17	25074	14	26058	14	26058	14	-3.78%	0.0%
	Average			28706.80	20.00	29337.20	18.60	28951.20	19.20	-0.84%	1.3%
300	0.8	1653	25	40165	31	39096	25	39886	26	0.70%	-2.0%
	1	1531	30	41925	35	41795	29	41553	29	0.90%	0.6%
		1837	25	35102	22	35926	21	35926	21	-2.29%	0.0%
	Average			39064.00	29.33	38939.00	25.00	39121.67	25.33	-0.15%	-0.5%
400	0.8	1615	34	53099	38	53135	35	53135	35	-0.07%	0.0%
	1	1527	40	56359	44	59543	42	55429	41	1.68%	7.4%
		1795	34	46607	29	49499	30	45434	28	2.58%	8.9%
	Average			52021.67	37.00	54059.00	35.67	51332.67	34.67	1.34%	5.3%

4.4 to 4.7 do not include that value for route length, Table 4.8 shows results obtained when TL is set to 3000. For each size of instances, average total traveling cost reported in Shi et al. (2009) is shown (see column Average above GA group) as well as the average number of vehicles required. Shi et al. (2009) do not report best found solutions; therefore, Table 4.8 also show the average traveling cost over ten runs for each one of the proposed LNS-based algorithms. For each size of the problem, at least one the proposed strategies outperforms solutions in Shi et al. (2009), on average (except for $|\mathcal{N}| = \{40, 400\}$). Lastly, improvement percentages are computed as in Table 4.7 for ALNS-EA over GA, MS–ILNS and ALNS-Math algorithms. Despite results obtained with different values for TL as in Table 4.7, when $TL = 300$ and $|\mathcal{N}| \geq 60$, MS–ILNS outperforms ALNS-EA.

Regarding computational times for LNS-based algorithms, Table 4.9 summarizes the performance of proposed strategies. As expected, required CPU time increases as the size of instances also increases. Since maximum computation time for algorithm was set to one hour, MS-ILNS and ALNS-EA reach that limit for instances with more than 200 nodes. For instances up to 100 nodes MS-ILNS requires less computation time than ALNS-Math and ALNS-EA. Differences between MS-ILNS and ALNS algorithms for CPU times rely mainly on MILPs as destroy and/or repair methods. Differences between ALNS-Math and ALNS-EA, are based on the definition of set $\mathcal{K}'$ for ALNS-Math. Recall that ALNS-Math attempts to repair a solution adding nodes over the subset of partially destroyed routes while ALNS-EA adds removed nodes over the whole set of routes $\mathcal{K}$.

4.6.4 Comments on multi-start iterative LNS matheuristic performance

As described in Algorithm 4.2, the proposed strategy firstly finds a 1–PDVRPLC solution within a multi-start iterative LNS. Then, a SPP IP is solved after a large pool of routes is created. For the sake of clarity, Table 4.10 reports main results after the LNS procedure is applied and also, how obtained solution is improved once the SPP IP is solved. For each size of instances, Table 4.10 shows the average traveling cost and CPU times required (D&R MILP time and Avg.CPU time) for MS-ILNS before SPP is solved. In a similar way, the average value for objective function, time required to solve SPP (Avg. IP CPU time) are also reported. Firstly, a vast percentage of CPU time required for the solution strategy is devoted to LNS procedure (before SPP is applied). For most of the cases, no more than 10 seconds are required to solve the SPP model. On the other hand and regarding times for LNS procedure, more than 60% of computation effort is devoted to solve destroy and repair MILP. Finally, LNS component is able to find good quality solutions since SPP IP improves best found solution within LNS procedure only up to 4.16%.

Table 4.8: Comparative results on LNS-based algorithms for instances with $TL = 3000$

$\|\mathcal{N}\|$	GA[a]		MS–ILNS			ALNS-Math			ALNS-EA			ALNS-EA Improvement		
	Average	$\|\bar{\mathcal{K}}\|$	Best	Average	$\|\bar{\mathcal{K}}\|$	Best	Average	$\|\bar{\mathcal{K}}\|$	Best	Average	$\|\bar{\mathcal{K}}\|$	GA[a]	MS-ILNS	ALNS-M
20	5515.40	2.30	5001	5058.07	2.25	5001	5044.50	2.00	5001	5037.60	2.00	9.48%	0.41%	0.14%
30	6906.00	3.10	6587	6808.14	3.53	6503	6622.62	3.00	6503	6603.15	3.00	4.59%	3.10%	0.29%
40	7476.40	3.00	7652	7849.72	3.97	7407	7709.11	3.11	7407	7659.14	3.08	-2.39%	2.49%	0.65%
50	9263.50	3.80	7288	7828.63	3.83	7283	7763.30	3.30	7283	7795.67	3.39	18.83%	0.42%	-0.42%
60	9931.60	4.10	9007	9574.17	4.50	9212	9877.00	4.11	8988	9802.63	4.23	1.32%	-2.33%	0.76%
100	14379.30	5.60	13214	13758.39	6.25	13523	14498.17	5.67	13687	14065.17	5.96	2.23%	-2.18%	3.08%
200	23331.70	9.60	20918	21982.14	9.45	22671	24111.50	10.00	21712	23526.70	9.80	-0.83%	-6.57%	2.49%
300	29805.30	12.20	28184	28374.20	12.20	29025	30159.00	12.00	28760	29117.22	12.00	2.36%	-2.55%	3.58%
400	34574.40	14.50	37395	38011.75	15.92	38889	39858.33	15.83	37919	38553.08	15.69	-10.32%	-1.40%	3.39%
500	39872.50	16.50	34691	35063.33	14.33	36032	36913.89	15.00	34823	35411.11	14.13	12.60%	-0.98%	4.24%

[a] Results taken from Shi et al. (2009)

Table 4.9: Comparative results on CPU times (s) for LNS-based algorithms

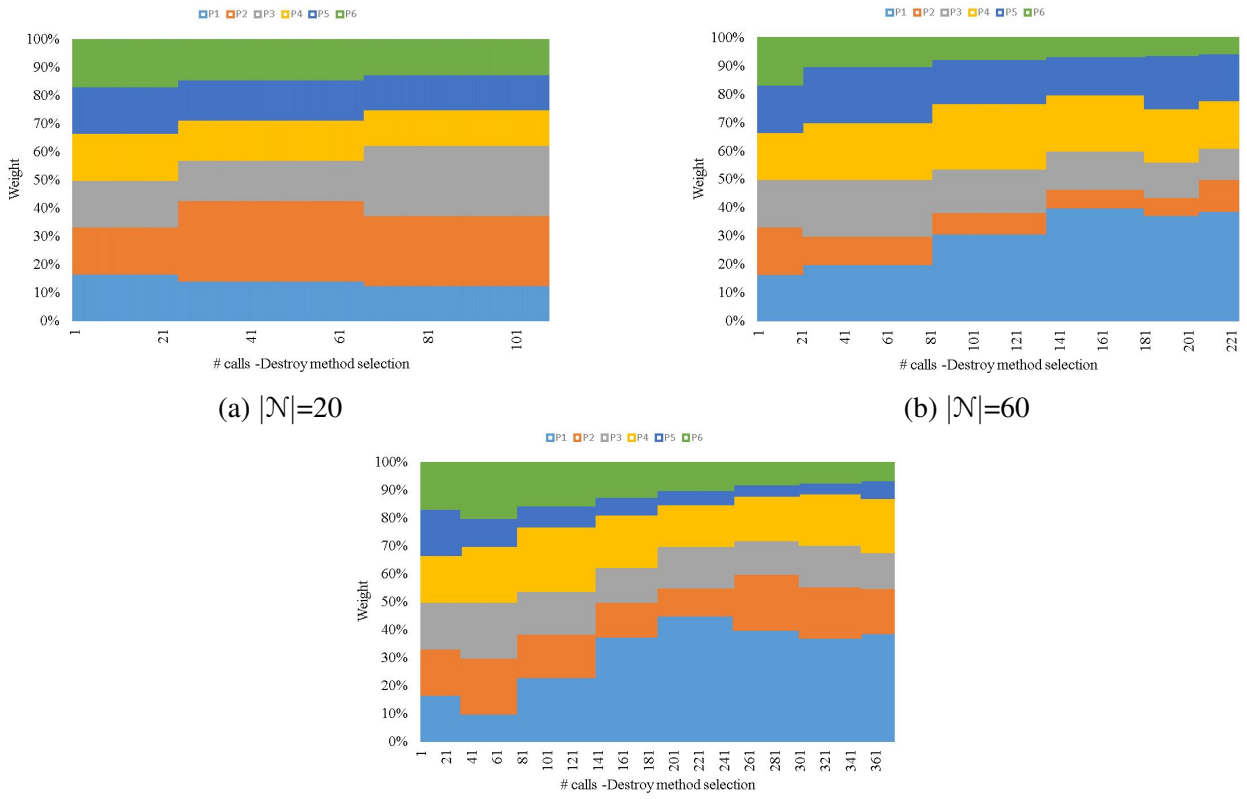
$\|\mathcal{N}\|$	MS–ILNS	ALNS-Math	ALNS-EA
20	49.97	93.06	78.88
30	133.20	226.08	197.13
40	377.23	378.34	352.98
50	153.57	307.91	408.11
60	241.10	384.09	569.77
100	764.72	890.73	1274.34
200	3417.23	1055.87	3600.00
300	3600.00	1714.77	3600.00
400	3600.00	2335.59	3600.00
500	3600.00	3164.29	3600.00

Table 4.10: Results on LNS procedure and SPP IP for the MS-ILNS

$\|\mathcal{N}\|$	MS-ILNS			SPP IP		
	Avg. f	D&R MILP time (s)	Avg. CPU time (s)	Avg. f	Avg. IP CPU time (s)	Imp. SPP
20	5090.53	32.39	49.97	5027.74	0.04	1.25%
30	6784.26	107.30	133.20	6719.54	0.04	0.96%
40	7730.72	332.38	377.23	7661.99	0.05	0.90%
50	8557.68	101.85	153.57	8215.53	0.07	4.16%
60	10319.46	177.68	241.10	9998.31	0.10	3.21%
100	14831.79	609.22	764.72	14593.10	0.46	1.64%
200	26547.87	2687.11	3417.23	26089.21	9.04	1.76%
300	37115.75	2894.56	3600.00	36450.80	52.47	1.82%
400	44249.90	3019.18	3600.00	43748.90	120.49	1.15%
500	35251.23	3245.40	3600.00	35063.33	0.44	0.54%

4.6.5 Comments on adaptive features for LNS-based algorithms

As described in Section 4.4.1 for ALNS algorithm, two sets of parameters are dynamically adjusted through optimization process: destroy methods for solutions and a *goal* which attempts to reduce fleet or improve traveling costs with the available number of vehicles. Figure 4.7 depicts the evolution of weights for set of destroy methods described in Section 4.4.3 for three different sizes of the problem. At the beginning of the optimization process, each destroy method can be selected with the same probability (i.e., 1/6) and weights are updated each 20 calls. As shown in Figure 4.7a, for small instances ($|\mathcal{N}| = 20$) destroy methods RANN and RANNLD conducted to a solution improvement with higher probability. For larger instances with 60 and 100 nodes as depicted in Figures 4.7b and 4.7c, respectively, method LANN leads to solution improvements more often than



(a) $|\mathcal{N}|$=20

(b) $|\mathcal{N}|$=60

(c) $|\mathcal{N}|$=100

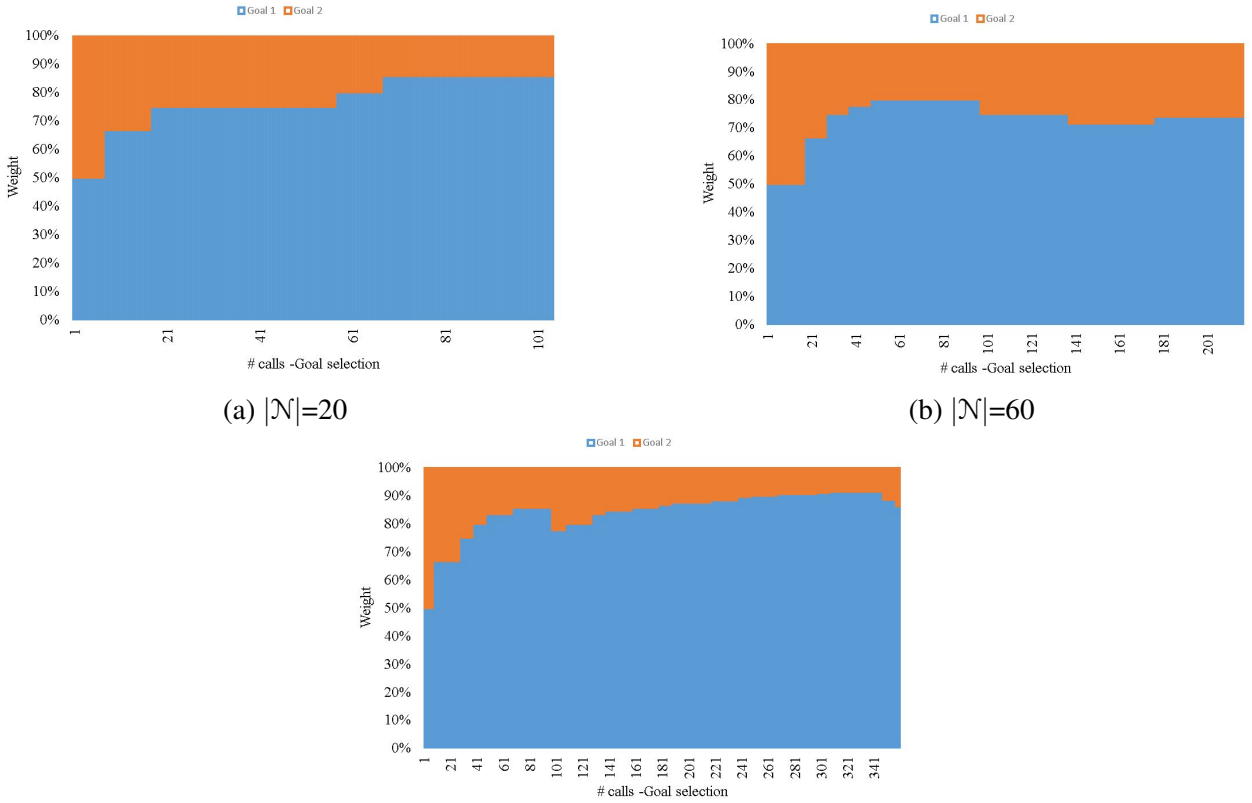
Figure 4.7: Destroy methods weight for different sizes of 1–PDVRPLC

other destroy methods. For large instances ($|\mathcal{N}| = 100$), ANRA seems to have a poor performance within the search process. Finally, the second set of parameters under adaptive control is related to goals. As expected, to partially destroy routes helps with a higher probability to improve solutions as depicted in Figure 4.8 for different instance size.

4.6.6 Comments on enumeration algorithm for ALNS-based strategies

This section briefly summarizes main differences on proposed ALNS algorithms performance. Recall that in ALNS-EA, repair operations are made via enumeration algorithm described in Section 4.5 while ALNS-Math repairs solutions vian MILPs (Sections 4.4.4 and 4.4.5). Moreover, to repair a solution after a subset of routes is partially destroyed, MILP in ALNS-Math repairs on $\mathcal{K}'$ (i.e., subset of partially destroyed routes), while EA in ALNS-EA searches for an improved solution adding removed nodes in the whole set of routes, $\mathcal{K}$.

To compare ALNS-Math and ALNS-EA, three simple experiments were conducted. For the three experiments, seven initial solutions for instance with 200 nodes and $TL = 3000$ were chosen,



(a) $|\mathcal{N}|$=20

(b) $|\mathcal{N}|$=60

(c) $|\mathcal{N}|$=100

Figure 4.8: Goals weight for different sizes of 1–PDVRPLC

saving seed values in order to compare fairly all the results. First experiment consists on applying repair MILP and EA to initial solution and setting $|\mathcal{K}'|$ as search space for EA, as in MILP. Table 4.11 shows results on this small experiment. For each destroyed solution, Repair MILP and EA retrieve the same value for traveling cost, as expected (see columns f). Columns time report computational time required to solve MILP and EA if dominance rules (Section 4.5.2) are not applied. Only for one case (seed = 2994), EA outperforms MILP time. Next, if dominance rules are applied to EA, computational times decrease for all cases (see column +DRs) and also, all times are less than those reported for MILP.

Similarly to Table 4.11, Table 4.12 reports results on seven initial solution after repair MILP and EA are applied as repair method. For this second experiment, EA runs on $\mathcal{K}$. Therefore, computational times increase significantly even if dominance rules are applied. However, total traveling costs (f) are improved for all cases.

In a third experiment, repair MILP and EA (running on $\mathcal{K}$) are compared setting an order for routes in $\mathcal{K}$ for EA. Nodes visited with vehicles in $\mathcal{K}'$ are added first to list T. Once all nodes in $\mathcal{K}'$

Table 4.11: Repair MILP in ALNS-Math vs. EA on $\mathcal{K}'$

Seed	Initial f	Repair MILP		EA on $\mathcal{K}'$		
		f	time (s)	f	time (s)	+DRs
837	62479	61893	1.08	61893	4.80	0.66
1001	62378	62105	0.71	62105	52.25	0.35
2	62913	62320	0.55	62320	0.55	0.08
2994	61631	61525	1.51	61525	1.25	0.05
399349	60304	59089	1.05	59089	10.52	0.48
2200	59180	58243	1.17	58243	104.55	0.13
156	60635	59845	1.13	59845	5.64	0.44

are in T, nodes in remaining routes ($\mathcal{K}\backslash\mathcal{K}'$) are added to T. The idea behind this order for routes in EA, relies on the fact that non-partially destroyed routes provide strong lower bounds (see equations (4.74) and (4.76)) to check dominance rules. Table 4.13 presents comparative results when routes in EA are sorted as described before. In spite of larger CPU times when compared with repair MILP, EA with dominance rules and sorted routes provides lower values for traveling costs and computational times decrease significantly after routes are sorted if they are compared with those reported in Table 4.12.

Table 4.12: Repair MILP in ALNS-Math vs. EA on $\mathcal{K}$

Seed	Initial f	Repair MILP		EA on $\mathcal{K}$	
		f	time (s)	f	+DRs
837	62479	61893	1.08	60878	54.69
1001	62378	62105	0.71	62074	4.85
2	62913	62320	0.55	61577	7.28
2994	61631	61525	1.51	61395	0.94
399349	60304	59089	1.05	58947	54.24
2200	59180	58243	1.17	58106	17.20
156	60635	59845	1.13	59156	9.61

4.7 Concluding remarks

A mathematical model based on mixed-integer linear programming is proposed to solve the 1–PDVRPLC. Due to imbalance on routes length for multi-vehicle problems if maximum duration are not imposed, a target based on the number of available vehicles and a upper bound for the 1–PDVRPLC is also proposed. Experiments performed solving MILP via commercial solver provide

Table 4.13: Repair MILP in ALNS-Math vs. EA on $\mathcal{K}$ and sorted routes

Seed	Initial f	Repair MILP		EA on $\mathcal{K}$	
		f	time (s)	f	+DRs
837	62479	61893	1.08	60878	2.04
1001	62378	62105	0.71	62074	1.12
2	62913	62320	0.55	61577	0.46
2994	61631	61525	1.51	61395	0.79
399349	60304	59089	1.05	58947	0.79
2200	59180	58243	1.17	58106	1.68
156	60635	59845	1.13	59156	6.73

insights about solution quality and computation effort when vehicle capacity and maximum route length vary.

Due to the $\mathcal{NP}$–Hardness of 1–PDVRPLC, instances up to 40 nodes were solved via commercial solver. Nevertheless, three matheuristic algorithms are proposed to solve medium and large size instances. These matheuristic strategies are based on LNS algorithms and its adaptive version. A multi-start iterative LNS (MS–ILNS) able to destroy and repair solutions vian MILP is the first proposed solution strategy. Next, as second solution strategy, an adaptive LNS able to repair solutions vian MILP after a destroy method is selected dynamically is also presented (ALNS-Math). Lastly, in ALNS-Math, MILP is replaced with a enumeration algorithm within a recursive structure (ALNS-EA). These three strategies are able to solve instances with up to 500 nodes within a maximum computation time of one hour. In spite of the required computational time, ALNS-EA outperforms MS-ILNS and ALNS-Math, on average. Lastly, ALNS-EA is also able to find better solutions than those reported in Shi et al. (2009), on average.

Conferences and publications

Results and analysis on MILP for 1–PDVRPLC was presented in the International Conference on Computational Logistics (ICCL) 2019:

- Palacio J.D., Rivera J.C. The One-Commodity Pickup and Delivery Vehicle Routing Problem: A mixed-Integer Linear Programming Approach. 10th International Conference on Computational Logistics (ICCL). Barranquilla, Colombia. 2019.

Results and analysis on MILP for 1–PDVRP was presented in III Congreso Colombiano de Investigación Operativa (ASOCIO), 2019:

- Palacio J.D., Rivera J.C. Modelos de programación lineal entera mixta para el problema de reposicionamiento de bicicletas. III Congreso Colombiano de Investigación de Operaciones (ASOCIO). Bucaramanga, Colombia. 2019.

Chapter 5

The two-echelon bicycle repositioning problem with split delivery

5.1 Introduction and motivation

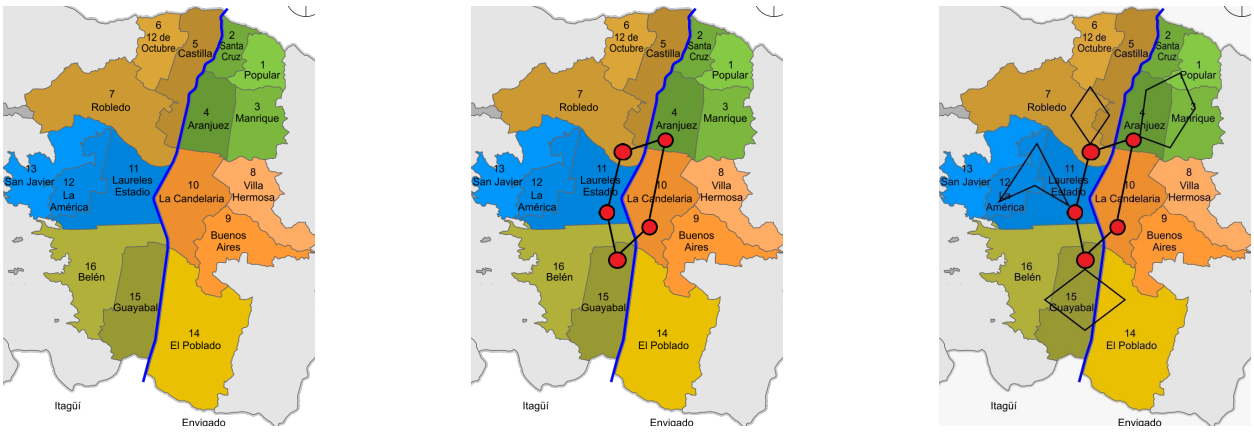
This chapter describes a new static BRP that, apart from the pickup and delivery structure of the problem, combines two key features on VRPs: two-echelon configuration and split delivery. This problem, hereafter is called the two-echelon bicycle repositioning problem with split delivery (2E–BRPSD).

In the literature, several authors have proposed variants for the BRP in order to provide efficient alternatives to repositioning operations in different BSSs around the world. Lee et al. (2020) describe a selective-based vehicle routing optimization problem for repositioning process for a BSS in Gangnam-district in Seoul, South Korea. Duan and Wu (2022) propose a BRP with worker recruitment decisions in which BSS users are rewarded if they ride bikes from a station with bike excess to a station with a shortage indicated by the system operator in New York, USA. Wang and Szeto (2021b) define the green bike repositioning problem with broken bikes; in this problem broken bikes are collected by the vehicles and the classical distance-based objective function is replaced by the minimization of CO_2 emissions during the repositioning operation (based on a linear function for fuel consumption as described in Xiao et al. (2012)). In this case, authors test their solution strategy with instances based on CityBike BSS in Vienna, Austria.

In Latin America there also exist many BSSs in which decisions for rebalancing operations must be made. One of them is EnCicla, the public BSS in Medellín, Colombia. Medellín is the second largest city in Colombia and the capital of Antioquia Department. The city is located in the

Aburrá Valley with a population of around 2.533.424 people. For administrative purposes, Medellín is divided into six zones. Each one of these zone is also subdivided into communes. Consequently, the city is made up of 16 communes as shown in Figure 5.1a. The city BSS, EnCicla, is currently operating with 101 stations throughout the six zones and one operation center (i.e., depot) located in commune 10 (downtown). Inspired on the administrative division of the city and future expansion plans of EnCicla, one suitable way to perform repositioning process may consist on designing a *central* route able to support (but not to complete) the operation, starting from the operation center and visiting at least one station in each zone (henceforth *satellite depots*) as depicted in Figure 5.1b. Additionally, the remaining vehicles in the fleet may complete the repositioning operation at non-visited stations in each zone (*secondary* routes) not only using bikes available in the visited communes but with bicycles provided by the vehicle serving the central route. Since satellite depots are visited twice (within the central route and one secondary route), a lack of bikes in each zone may be covered via split demand process at satellite depots (see Figure 5.1c). If the total traveling cost of central and secondary routes is minimized, the two-echelon bicycle repositioning problem with split deliveries (2E–BRPSD) arises. In the proposed two-echelon configuration for the BRP, satellite depots are not previously defined as in classical two-echelon vehicle routing problems.

The 2E–BRPSD differs from the multi-vehicle BRP because not all vehicles must leave (and return) to the operation center and also, demand at some stations (i.e., satellite depots) is split. Indeed, split demand in pickup and delivery problems as the BRP may ends up in a traveling cost reduction as locations (i.e., stations) can be used to storage an amount of units that are picked up later or locations can be used to lend an amount of units that are replaced later (Palacio and



(a) Administrative division in Medellín (Colombia).

(b) Example of central route for the 2E–BRPSD.

(c) Example of solution for the 2E–BRPSD.

Figure 5.1: An example of a 2E–BRPSD solution for Encicla in Medellín (Colombia).

Rivera, 2019). Moreover, if a location is served by more than one vehicle, there exists an inherent synchronized operation. Particularly, synchronization is not explicitly included as a set constraints within a temporal aspect for our proposed 2E–BRPSD, but a load synchronization (Drexl, 2012) is imposed to satellite depots in the problem.

This chapter initially proposes a mixed-integer linear programming model (MILP) for the 2E–BRPSD. Nonetheless, since the original BRP is an $\mathcal{NP}$-hard problem (Ho and Szeto, 2014), the 2E–BRPSD is also $\mathcal{NP}$-hard. Thus, this chapter also presents a metaheuristic algorithm and three hybrid solution strategies based on MILP and metaheuristic algorithms (i.e., matheuristic) to solve medium and large-size problem instances. The first strategy heuristically finds secondary routes via greedy randomized procedure (GRP) and split algorithm. For each set of secondary routes, a greedy procedure is also used to construct a central route. In a second solution strategy a pool of central routes as well as a pool of secondary routes are created and then, an enhanced partitioning problem (SPP) formulation determines the best combination for central and secondary routes. A third solution strategy firstly generates heuristically a large set secondary routes. Then, in an iterative way, a SPP IP selects the best secondary routes and next, a *generalized traveling salesman problem* (GTSP) MILP retrieves a central route for the 2E–BRPSD. Finally, as fourth solution strategy a variation on third matheuristic is proposed by modifying criterion to select secondary routes at each iteration of the algorithm.

The remainder of this chapter presents a brief review on two-echelon routing problems. Next, a mathematical formulation for the 2E–BRPSD is described as well as four solution strategies to solve the problem. Lastly, computational results on solution strategies are summarized.

5.2 A brief review on two-echelon routing problems

This section presents a brief literature review on two-echelon VRPs in different applications and contexts. The purpose of this review is not to provide a complete state of the art for two-echelon routing problems but to show that there is no evidence of prior work on this configuration for static BRPs.

In order to briefly summarize only the latest literature for two-echelon routing problems, this short review starts with the survey presented in Cuda et al. (2015). In that paper, the authors provide a classification for two-echelon routing problems by reviewing three classes of problems: the two-echelon location routing problem (2E–LRP), the two-echelon vehicle routing problem (2E–VRP) and the truck and trailer routing problem (TTRP). In the 2E–LRP, strategic and tactical decisions are made since location for intermediate facilities (or *satellite depots*) must be defined as

well as vehicle routes. The 2E–LRP aims to minimize the total traveling cost and the opening costs for each satellite depot. On the other hand, the 2E–VRP only minimize routing costs since location for satellite depots is previously defined. Finally, in the TTRP some customers are served with a truck pulling a trailer or just with a truck. The TTRP also aims to minimize the routing cost since satellites are given. Nevertheless, in this problem an additional constraint is imposed: a complete vehicle must visit all customers from the first level route while customers within a second level are served via the truck alone. Cuda et al. (2015) report a wide number of solution strategies for 2E-LRP, 2E-VRP and TTRP based on exact approaches and heuristic algorithms. Matheuristics strategies are also developed for two-echelon problems. (e.g., Caramia and Guerriero (2010); Perboli et al. (2011); Villegas et al. (2013))

After the review in Cuda et al. (2015), several authors deal with two-echelon routing problems. Vidović et al. (2016) address the 2E–LRP for designing a non-hazardous recyclables collection network. The authors propose an MILP and a heuristic procedure to determine the location of collection and transfer points. The heuristic decomposes the problem in two steps: firstly, the number and location of collection points is determined. Then, in a second phase, the optimal route for each collection vehicle and the transfer points is computed.

Breunig et al. (2016) propose a hybrid metaheuristic based on D&R algorithm and local search procedures to solve the 2E–VRP and the 2E–LRP. The authors test the algorithm solving six different sets of instances with up to 200 customers. This hybrid strategy is able to improve best known solutions for 18 out of 49 2E–VRP instances. Grangier et al. (2016) also develop a LNS algorithm to solve a variant of the 2E–VRP: the two-echelon multiple-trip vehicle routing problem with satellite synchronization (2E-MTVRP-SS). Apart from the classical configuration of the 2E–VRP, the 2E-MTVRP-SS also includes time windows, multiple trips for second level vehicles, and synchronization constraints for the operation at satellite depots. To test the algorithm, instances up to 200 customers and ten satellite depots were solved.

Similar to Grangier et al. (2016), Wang et al. (2017) describe a new 2E–VRP variant in which capacitated vehicles perform first and second level routes minimizing drivers wage, fuel cost and handling cost at satellite depots. This problem is called the capacitated 2E–VRP with environmental considerations (2E–CVRP–E) and it is solved via matheuristic algorithm. This strategy is based on a VNS and an IP model as a post-optimization technique. To validate the performance of this matheuristic approach, the authors solve a set of 2E–VRP instances finding new best known solutions for 13 out of 234 instances. In the literature, 2E–VRPs with pick up and delivery operations are also reported as in Belgin et al. (2018). The authors propose an MILP and a hybrid metaheuristic algorithm based on VND and LS. Despite the linear relaxation is

strengthened with three valid inequalities, the hybrid VND and local search strategy outperforms the mathematical model performance not only on computational times but in solutions quality for well-known 2E–VRP instances with up to 50 customers in second level routes. Belgin et al. (2018) also report results on VND and LS algorithm for a supermarket distribution problem in Turkey. The authors prove that a two-echelon configuration improves total cost when it is compared with a single-echelon distribution operation.

Two echelon configurations are also reported for vehicle routing problems with an electric fleet as in Breunig et al. (2019) and Jie et al. (2019). Firstly, Breunig et al. (2019) introduce the electric two-echelon vehicle routing problem (E2E–VRP) in the context of city logistics. The authors solve the problem via an exact mathematical programming algorithm that uses decomposition techniques and develop a LNS based metaheuristic. These algorithms are tested on new instances that simulate characteristics of urban areas by including different density factors for charging stations. The instances are designed with up to 200 customers and ten satellite depots. Then, Jie et al. (2019) tackle a problem similar to the E2E–VRP in which vehicles must visit battery swapping stations in order to change the almost depleted battery by a completely charged one. This problem is called the two-echelon electric vehicle routing problem with battery swapping stations (2E–EVRP–BSS). As solution strategies for the 2E–EVRP–BSS, a hybrid column generation and adaptive LNS (CG-ALNS) are developed. This algorithm is tested on instances with up to 200 customers, ten satellite depots and 20 battery swapping stations. The authors also present an economic analysis and managerial implications of battery driving ranges and efficiency of vehicle emission reduction.

Recently, Fallahtafti et al. (2021) describe a multi-objective perspective for the 2E–LRP within a cash logistics context. In this case, several vehicles must leave the central bank (depot), then some intermediate facilities (satellite depots) are visited. In the second level routes, ATMs (customers) are attended. This 2E–LRP supports decision making process on opening and closing intermediate facilities while objective functions minimize the risk of robbery and the total traveling cost. To tackle this bi-objective problem, ε–constraint and five different metaheuristic algorithms are implemented. For large instances, authors prove that archived multi-objective simulated annealing (AMOSA) outperforms the other four strategies including non-dominated sorting genetic algorithms (NSGA–II and NSGA–III).

5.3 Mathematical model for the 2E–BRPSD

In this section, a mathematical formulation for the 2E–BRPSD is presented. This formulation is an MILP that describes the conditions of the problem where the total cost (i.e., routes total distance) is

minimized.

Similar to the 1–PDTSP and the 1–PDVRP described in sections 3.2 and 4.2.1, the 2E–BRPSD is defined on a complete and directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$. For the 2E–BRPSD, $\mathcal{N}$ is the set of BSS stations ($\mathcal{N} = \{0, 1, 2, .., n\}$) and $\mathcal{A}$ is the set of arcs connecting each pair of stations. Without loss of generality, station 0 refers to the operation center (i.e., depot) for the BSS. In the 2E–BRPSD a set of capacitated vehicles $\mathcal{K}$ is available to serve the demand at stations. Following the notation presented in sections 3.2 and 4.2.1, parameter c_{ij} denotes the cost of traveling from station i to station j. Q describes the vehicles capacity. If $q_i > 0$, then q_i bikes must be delivered at station i. On the other hand, if $q_i < 0$, then q_i bikes must be picked up at station i. For the MILP, the binary decision variable y_{ij}^k, takes the value of one only if vehicle k crosses arc (i, j). Variable w_i^k also can take the value of one if vehicle k visits station i and zero otherwise. d_i^k is equal to one only if station i is selected as a satellite depot for vehicle k. d_i^k takes the value of zero in any other case. Variable l_{ij}^k denotes the number of bikes in vehicle k when traversing arc (i, j). Finally, z_{ij}^k denotes a label to avoid arc (i, j) to be part of subtours in final solution. The proposed MILP for the 2E–BRPSD follows:

$$\min f = \sum_{(i,j)\in\mathcal{A}} c_{ij} \cdot \sum_{k\in\mathcal{K}} y_{ij}^k \tag{5.1}$$

subject to,

$$\sum_{i\in\mathcal{N}} d_i^k = 1, \qquad \forall\, k \in \mathcal{K} \tag{5.2}$$

$$\sum_{k\in\mathcal{K}} d_i^k \leq 1, \qquad \forall\, i \in \mathcal{N} \tag{5.3}$$

$$d_0^0 = 1 \tag{5.4}$$

$$d_i^k \leq w_i^k, \qquad \forall\, i \in \mathcal{N},\ k \in \mathcal{K} \tag{5.5}$$

$$d_i^k \leq w_i^0, \qquad \forall\, i \in \mathcal{N} \setminus \{0\},\ k \in \mathcal{K} \setminus \{0\} \tag{5.6}$$

$$\sum_{i\in\mathcal{N}} w_i^0 = |\mathcal{K}|, \tag{5.7}$$

$$w_0^0 = 1 \tag{5.8}$$

$$\sum_{i\in\mathcal{N}} w_i^k \leq \alpha \cdot |\mathcal{N}|, \qquad \forall\, k \in \mathcal{K} \setminus \{0\} \tag{5.9}$$

$$\sum_{j\in\mathcal{N}} y_{ij}^k = w_i^k, \qquad \forall\, i \in \mathcal{N},\ k \in \mathcal{K} \tag{5.10}$$

$$\sum_{\substack{j\in\mathcal{N}\\ j\neq i}}\sum_{k\in\mathcal{K}} y_{ij}^{k} \geq 1, \qquad \forall\, i \in \mathcal{N}\setminus\{0\} \tag{5.11}$$

$$\sum_{\substack{j\in\mathcal{N}\\ j\neq 0}} y_{0j}^{0} = 1, \tag{5.12}$$

$$\sum_{\substack{j\in\mathcal{N}\\ j\neq i}} y_{ij}^{k} = \sum_{\substack{j\in\mathcal{N}\\ j\neq i}} y_{ji}^{k}, \qquad \forall\, k \in \mathcal{K},\ i \in \mathcal{N} \tag{5.13}$$

$$l_{ij}^{k} \leq Q \cdot y_{ij}^{k}, \qquad \forall\, k \in \mathcal{K},\ (i,j) \in \mathcal{A} \tag{5.14}$$

$$\sum_{\substack{j\in\mathcal{N}\\ j\neq i}}\sum_{k\in\mathcal{K}} l_{ji}^{k} - \sum_{\substack{j\in\mathcal{N}\\ j\neq i}}\sum_{k\in\mathcal{K}} l_{ij}^{k} = q_i, \qquad \forall\, i \in \mathcal{N} \tag{5.15}$$

$$\sum_{j\in\mathcal{N}} z_{ji}^{k} - \sum_{j\in\mathcal{N}} z_{ij}^{k} \geq \frac{\sum_{j\in\mathcal{N}} y_{ij}^{k}}{|\mathcal{N}|} - M \cdot d_i^{k}, \qquad \forall\, k \in \mathcal{K},\ i \in \mathcal{N} \tag{5.16}$$

$$\sum_{j\in\mathcal{N}} z_{ji}^{k} - \sum_{j\in\mathcal{N}} z_{ij}^{k} \leq 1 + M \cdot d_i^{k}, \qquad \forall\, k \in \mathcal{K},\ i \in \mathcal{N} \tag{5.17}$$

$$z_{ij}^{k} \leq 2 \cdot |\mathcal{N}| \cdot y_{ij}^{k}, \qquad \forall\, k \in \mathcal{K},\ (i,j) \in \mathcal{A} \tag{5.18}$$

$$d_i^{k}, w_i^{k} \in \{0,1\}, \qquad \forall\, k \in \mathcal{K},\ i \in \mathcal{N} \tag{5.19}$$

$$y_{ij}^{k} \in \{0,1\}, \qquad \forall\, k \in \mathcal{K},\ (i,j) \in \mathcal{A} \tag{5.20}$$

$$l_{ij}^{k}, z_{ij}^{k} \geq 0, \qquad \forall\, k \in \mathcal{K},\ (i,j) \in \mathcal{A} \tag{5.21}$$

The objective function in (5.1) aims to minimize the total traveling cost for the repositioning operation. Equations in (5.2) state that only one station i is assigned as depot for each vehicle k. Constraints (5.3) assign a station i as a depot for at most one vehicle k. In equation (5.4), station 0 is fixed as the depot for the central route. Inequalities in (5.5) force the model to assign a station i as depot for vehicle k only if i is visited by the vehicle while (5.6) ensure that all depots are visited in the central route. The size of the central route is defined by the number of available vehicles (i.e., one vehicle for the central route and one vehicle for each secondary route) as described in (5.7). Equation (5.8) forces the model to assign the operation center ($i = 0$) to the central route. The length of each secondary route does not exceed a percentage of the BSS size as described in (5.9). Equations in (5.10) describe a relation between variables y and w by including an arc (i, j) for route k only if the vehicle visits station i. Constraints in (5.11) and (5.12) state that each station i ($i > 0$) is visited in a secondary route while an arc starting in the operation center ($i = 0$) is added to the central route. In (5.13) if a vehicle goes into a station i, then the vehicle must leave the station. With constraints (5.14), the load of the vehicle does not exceed its capacity while equations in (5.15) assure that demand in each station is satisfied. As described in MILPs for 1–PDTSP and 1–PDVRP

in sections 3.2 and 4.2.1, constraints proposed in Miller et al. (1960) are adapted to fix a coefficient for each arc with expressions (5.16) and (5.17). Inequalities in (5.18) limit arc coefficients. Values for variables z_{ij}^k may vary from zero to $2 \cdot |\mathcal{N}|$. Since several stations (i.e., satellite depots) are visited twice, $2 \cdot |\mathcal{N}|$ denotes an upper bound for z_{ij}^k. Finally, expressions (5.19) to (5.21) describe the nature of decision variables. For BSS contexts, load of vehicles (l_{ij}^k) must be integer, nevertheless, if $q_i \in \mathbb{Z}$ in equations (5.15), a continuous domain for l_{ij}^k leads to integer values for these variables.

Figure 5.2 depicts the optimal solution for a 2E–BRPSD instance with 20 stations, three secondary routes, and $Q = 10$ for each vehicle. Numbers above each node are station demands (positive and negative numbers represent delivery and pick up operations, respectively). In the optimal solution, stations 19, 16 and 5 are selected as depots for secondary routes, thus, these stations are included in the central route for the instance which also includes node 0 as origin (see dotted arcs). Demand at stations in central route (except for station 0) is fulfilled using not only the central route vehicle but vehicles serving secondary routes. In point of fact, bikes picked up or delivered at satellite depots allow to meet demand in other stations visited in secondary routes. For instance, the route $5 \rightarrow 17 \rightarrow 3 \rightarrow 11 \rightarrow \ldots \rightarrow 5$ is not feasible to be served with a single vehicle with capacity $Q = 10$ (i.e., 14 bikes must be delivered in stations 5, 17 and 11). However, if the

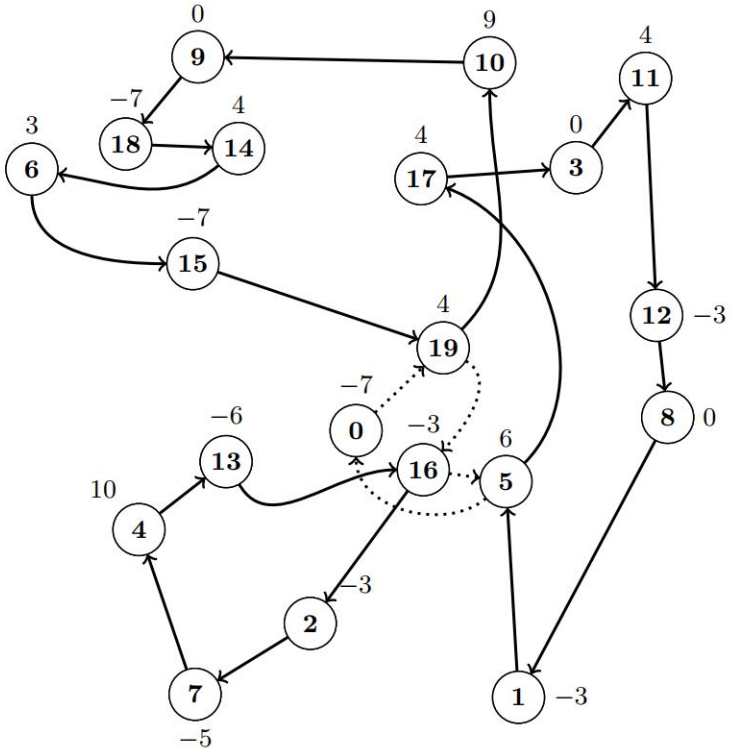


Figure 5.2: Optimal solution for a 2E–BRPSD instance with 20 stations and four vehicles

vehicle assigned to central route delivers eight bikes in station 5, these bikes can be delivered at stations 17 and 11 fulfilling their demands. Similarly, a single vehicle is not able to serve route $16 \rightarrow 2 \rightarrow 7 \rightarrow ... \rightarrow 16$ since 11 bikes are picked up (with no delivery operations in between). Nevertheless, if station 16 is used as a temporal storage point, the vehicle serving secondary route can traverse the arc $(16, 5)$ with two bikes (not three as demand in station 16 indicates) and complete the route with feasible loads. Request at station 16 is completely satisfied after secondary route is completed. Table 5.1 summarizes the load of vehicles serving central and secondary routes before and after visiting each depot (stations in central route). As modeled in equations (5.15), the difference between outgoing and ongoing flows for each station ensure demand requirements.

Table 5.1: Values for vehicles load when visiting central route stations

Station	0		19		16		5	
	In	Out	In	Out	In	Out	In	Out
Central	(5,0) 2	(0,19) 9	(0,19) 9	(19,16) 3	(19,16) 3	(16,5) 10	(16,5) 10	(5,0) 2
Secondary			(15,19) 7	(19,10) 9	(13,16) 6	(16,2) 2	(1,5) 6	(5,17) 8
Flow	2	9	16	12	9	12	16	10
Demand	-7		4		-3		6	

5.4 Hybrid constructive algorithm for the 2E–BRPSD

This section presents a first solution strategy for the 2E–BRPSD. The strategy can be described in three phases. Firstly, a greedy algorithm generates a set of partial solutions (i.e. secondary routes). To do so, the algorithm finds a Hamiltonian tour which is improved and then split in several secondary routes. In second phase, a naive constructive algorithm finds a central route checking whether is feasible to visit a station in each secondary route. Finally, a set of linear constraints determines the global feasibility of the 2E–BRPSD solution in terms of vehicle loads. These three steps are executed repeatedly until a stopping criterion is met. The algorithm returns the best solution found.

Algorithm 5.1 describes a hybrid constructive algorithm as a first procedure to solve the 2E–BRPSD. This algorithm requires a parameter maxIterations (i.e. maximum number of solutions to generate). The parameter maxIterations also determines a stop criterion for the algorithm. Since a secondary route is not necessarily balanced (i.e., the sum over station demands in the route may be different from zero), parameter β denotes the maximum percentage of unbalanced Hamiltonian tours. The function `generateHamiltonianTour` in line 5 finds an (un)feasible tour via greedy randomized algorithm and then, functions `VND` and `split` are called to improve the tour and to split

it in $|\mathcal{K}|-1$ routes, respectively (see lines 6 and 7). Each secondary route is improved by checking if moving a station to any other position in the route is feasible and reduce the total traveling cost (i.e. local search procedure in line 8). In line 9, the function `feasibleSatelliteDepots` returns the subset of candidates stations for satellite depot (i.e., starting and ending a secondary route in a given station may end up in an unfeasible load for the vehicle. The function `buildCentralRoute` finds a suitable subset of stations over *depots* conforming a central route (i.e., at least one candidate for satellite depot in each secondary route is visited). If the total traveling cost for central and secondary routes is less than the best cost found and also, if the new solution is globally feasible, then the best solution is updated (lines 12 to 14). Subsequent sections describe in detail each function within the hybrid constructive algorithm.

Algorithm 5.1: Hybrid constructive algorithm for the 2E–BRPSD

```
1:  function: hybridConstructive (maxIterations, β, |𝒦|)
2:  f* ← +∞, s* ← ∅
3:  for iteration= 1 to maxIterations do
4:      s ← ∅
5:      T ← generateHamiltonianTour(iteration,β)
6:      T ← VND(T)
7:      s ← split(T, |𝒦|)
8:      s ← localSearch(s)
9:      depots ← feasibleSatelliteDepots(s)
10:     r ← buildCentralRoute(s, depots)
11:     s ← s ⋃ r
12:     if f(s) < f* and isFeasible(s) then
13:         f* ← f(s)
14:         s* ← s
15:     end if
16: end for
17: return s*
```

5.4.1 Greedy randomized construction

Algorithm 5.2 shows the framework for an iterative tour construction. This process requires a greedy function which is based on *nearest neighbor* algorithm. For β percent of iterations, the function `generateHamiltonianTour` allows to return an unfeasible Hamiltonian tour. A tour for the BRP is feasible if the maximum load of the vehicle is less or equal than the capacity Q. Without loss of generality, feasibility for a Hamiltonian tour within the BRP context may be proven if

$$\max_{i\in\{1,..,|\mathcal{N}|\}}\{l_i\} - \min_{i\in\{1,..,|\mathcal{N}|\}}\{l_i\} \leq Q \tag{5.22}$$

where l_i is the vehicle load after visiting the i-th station in the tour. Thus, l_i may be computed as: $l_i = l_{i-1} - q_{T[i]}$. Similarly, an unfeasible tour met that

$$\max_{i\in\{1,..,|\mathcal{N}|\}}\{l_i\} - \min_{i\in\{1,..,|\mathcal{N}|\}}\{l_i\} = Q + \delta \tag{5.23}$$

where δ can be defined as a feasibility violation degree (e.g., number of loaded bikes that exceed the vehicle capacity. Without loss of generality, a non-positive feasibility violation degree (i.e., $\delta \leq 0$), represents a feasible Hamiltonian tour. In lines 3 and 5 of Algorithm 5.2, the function `greedyRandomizedAlgorithm` finds an unfeasible and feasible tour, respectively.

Algorithm 5.2: Hamiltonian tour construction

1: **function:** generateHamiltonianTour (iteration, β)
2: **if** iteration $\leq \beta \cdot$maxIterations **then**
3: $T \leftarrow$ `greedyRandomizedAlgorithm`(**False**, seed)
4: **else**
5: $T \leftarrow$ `greedyRandomizedAlgorithm`(**True**, seed)
6: **end if**
7: **return** T

To find feasible Hamiltonian tours, Palacio and Rivera (2022) present a *nearest neighbor* algorithm for the 1–PDTSP (described in Section 3.3.2). Based on the ideas in Palacio and Rivera (2022), the `greedyRandomizedAlgorithm` function to generate (un)feasible tours for the 2E–BRPSD may be described as follows:

(i) Select a station i as the first visited location at random. Let the number of visited stations be equal to one ($p \longleftarrow 1$) and $T[p] \longleftarrow i$.

(ii) Let $\mathcal{C}$ be the set of closest and feasible (i.e. the vehicle capacity is not violated) non-visited stations after the station i is served. The size of $\mathcal{C}$ (i.e., $|\mathcal{C}|$) is the minimum value between a restricted candidate list size (φ) and the number of feasible non-visited stations. Note that if $\mathcal{C} = \emptyset$, the constructed solution so far leads to an unfeasible path. If the tour is required to be feasible, then go to step (i). Otherwise, $\mathcal{C}$ is completed including the closest but unfeasible non-visited stations.

This step requires to check whether is feasible to add a station j to a partial tour with size $p-1$. To do so, it is possible to adapt condition in (5.22) as in Hernández-Pérez and Salazar-González (2004b):

$$\max_{i=1,..,p-1}\{l_i, l_{p-1}-q_j\} - \min_{i=1,..,p-1}\{l_i, l_{p-1}-q_j\} \leq Q \tag{5.24}$$

(*iii*) Choose a station j from $\mathcal{C}$ at random.

(*iv*) Let $p \longleftarrow p+1$ and $T[p] \longleftarrow j$. If $p < |\mathcal{N}|$ then define $i \longleftarrow j$ and go to step (ii), else stop.

5.4.2 Variable neighborhood descent

The proposed VND to solve the 2E–BRPSD is based on three local search operators: move vertex, $2-opt$, and $3-opt$. A brief description of these operators follows:

(*i*) Move vertex: the first local search operator ($v = 1$), attempts to improve the total traveling cost of a tour by moving a station from its original position p to a different one in the path. The new position p' may vary from 1 to $|\mathcal{N}|-1$ and $p' \neq p$. It is worth to recall that a tour passed to this LS operator may be unfeasible. Thus, *move vertex* implementation ensures that if tour T' is unfeasible, then feasibility violation degree (i.e., δ in equation (5.23)) does not increase its value. Similarly, if tour T' is feasible, *move vertex* aims to improve the solution preserving its feasibility (see condition in (5.22)).

The complexity of *move vertex* operator is $\mathcal{O}(|\mathcal{N}|^2)$. Nonetheless, this implementation follows the ideas described in Lin (1965) for $2-opt$ and $3-opt$ operators. Firstly, a list of m nearest locations for each station is stored. Then, the operator only scans a possible move for station i if it can be located one position before one of its m closest stations. Following this strategy, the complexity of *move vertex* operator is reduced to $\mathcal{O}(|\mathcal{N}| \cdot m)$.

(*ii*) $2-opt$: similarly to *move vertex*, $2-opt$ as second local search operator ($v = 2$) in VND aims to improve total traveling cost for T' without increasing δ value in equation (5.23). This neighborhood deletes two edges and therefore the tour is divided in two paths. Then, those paths are connected again in the other possible way. The computational complexity of $2-opt$ is $\mathcal{O}(|\mathcal{N}|^2)$ but as in *move vertex*, the number of potential edge exchanges to the m closest stations to location i is reduced. Thus, $2-opt$ also runs in $\mathcal{O}(|\mathcal{N}| \cdot m)$.

(*iii*) $3-opt$: the third and final neighborhood within VND ($v = 3$) deletes three edges of the original tour and then, four different solutions may be found. If all possible combinations when deleting three edges are explored, $3-opt$ complexity is $\mathcal{O}(|\mathcal{N}|^3)$. In this implementation, only m possible improvements are explored for each one of the m nearest neighbors for each station i. This strategy leads to a complexity of $\mathcal{O}(|\mathcal{N}| \cdot m^2)$ instead of $\mathcal{O}(|\mathcal{N}|^3)$.

5.4.3 Split

In the hybrid constructive algorithm as solution strategy for the 2E–BRPSD, Split finds a set of secondary routes. Thus, all the stations (except the operation center or depot) conform the solution after tour T is split. To find the set of secondary routes, the Split algorithm that consider limited number of vehicles (Prins, 2004) is adapted. Moreover, this version of Split algorithm allows pickups and deliveries as required in BRPs (see Section 4.3.3). Details of the implementation are depicted in Algorithm 5.3.

The algorithm starts assigning an initial cost when reaching each station i with vehicle k (V_{ki}) in lines 1 to 6. Then, starting from the second position (i) in the Hamiltonian tour (i.e, operations center is not added to any secondary route), Split checks whether is possible to reach station in each position j in the tour. Constraints based on the maximum number of stations added to the route ($length \leq \alpha \cdot |\mathcal{N}|$) are evaluated as in constraints (5.9). Similarly, computing maximum and minimum vehicle loads (l^+ and l^-, respectively), feasible loads for the vehicle ($l^+ - l^- \leq Q$) are also imposed (see lines 9 to 20). After cost is updated if position j can be reached from station in position i, the algorithm checks for improves in the total traveling cost of the partial route. Similarly, the strategy updates the path (P) for the evaluated vehicle (lines 28 to 32). Lastly, the function `extractSolution` delivers the complete path for each route by extracting stations added to P.

5.4.4 Local search

After Split algorithm delivers a set of secondary routes, each one of these paths are improved via local search. Particularly, possible improvements on partial solution s are explored by applying *move vertex* operator described in Section 5.4.2. The main motivation to explore secondary routes via local search after a VND and Split are applied relies precisely on a different set of feasible neighbors that may be found over each separate route in s and not on a single Hamiltonian tour (T). Moreover, since each one of the secondary routes is feasible in terms of load after Split finds s, function `localSearch` always deals with a δ value equal to zero. Thus, this neighborhood at

Algorithm 5.3: Split algorithm for PDVRPs. Adapted from Prins (2004)

function: Split (T)
for $k = 1$ to $|\mathcal{K}|$ **do**
    for $i = 1$ to $|\mathcal{N}| - 1$ **do**
        $V_{k,i} \leftarrow +\infty$
    end for
end for
$V_{1,0} \leftarrow 0$
for $k = 1$ to $|\mathcal{K}|$ **do**
    for $i = 2$ to $|\mathcal{N}|$ **do**
        $l \leftarrow 0, l^+ \leftarrow 0, l^- \leftarrow 0$
        $cost \leftarrow 0$, $length \leftarrow 0$
        $j \leftarrow i$
        while $j < |\mathcal{N}|$ and $l^+ - l^- \leq Q$ and $length \leq \alpha \cdot |\mathcal{N}|$ **do**
            $l \leftarrow q_{T_{[j]}}$
            if $l < l^-$ **then**
                $l^- \leftarrow l$
            else
                if $l > l^+$ **then**
                    $l^+ \leftarrow l$
                end if
            end if
            if $j = i$ **then**
                $cost \leftarrow 2 \cdot c_{T[i],T[j]}$
            else
                $cost \leftarrow cost - c_{T[j-1],T[i]} + c_{T[j-1],T[j]} + c_{T[j],T[i]}$
            end if
            $length \leftarrow length + 1$
            if $l^+ - l^- \leq Q$ and $length \leq \alpha \cdot |\mathcal{N}|$ **then**
                if $V_{k,i-1} + cost < V_{k+1,j}$ **then**
                    $V_{k+1,j} \leftarrow V_{k,i-1} + cost$
                    $P_{k+1,j} \leftarrow i - 1$
                end if
            end if
            $j \leftarrow j + 1$
        end while
    end for
end for
$s^* \leftarrow$ `extractSolution(`P`)`
return s^*

this step of the solution strategy does not aim to reduce feasibility violations but to improve total traveling cost.

5.4.5 Finding feasible satellite depots

With the aim to find a central route able to connect and support the rebalancing operation through stations in secondary routes, the solution strategy must check for feasible satellite depots (i.e., stations belonging to central route and to one secondary route). To show the proposed procedure, a small example is described. In Figure 5.2, one of the secondary routes within the optimal solution is 16→2→7→4→13→16, where 16 is the satellite depot. Table 5.2 shows vehicle load l, maximum load l^+, minimum load l^-, and the feasibility violation degree δ for this secondary route. With an initial load equal to zero, vehicle traverses each arc and values for l^+, l^- are updated at each step as described in equation (5.24). Values for δ are also computed after each station is visited. The path followed in this route is feasible since final δ value is zero. If station 7 is selected as satellite depot, Table 5.3 shows that the path leads to an unfeasible route because values for l^+ and l^- end up with a feasibility violation degree, $\delta = 2$. Without loss of generality, initial vehicle load may be different from zero and values for δ do not vary. It is worth to recall that such initial load depends on the number of picked (or delivered) bikes with any of the two vehicles visiting the satellite depot and the split demand operation at that location.

	16	**2**	**7**	**4**	**13**	**16**
q	-3	-3	-5	10	-6	-3
l	0	3	8	-2	4	
l^+		3	8	8	8	
l^-		3	3	-2	-2	
δ		0	0	0	0	

Table 5.2: Feasible path for a secondary route

	7	**4**	**13**	**16**	**2**	**7**
q	-5	10	-6	-3	-3	-5
l	0	-10	-4	-1	2	
l^+		-10	-4	-1	2	
l^-		-10	-10	-10	-10	
δ		0	0	0	2	

Table 5.3: Unfeasible path for a secondary route

Algorithm 5.4 describes in detail the procedure to determine the subset of stations able to conform the satellite depots. For each route r in solution s, it is assumed that all stations may be satellite depots as shown in lines 5 and 7. Then, the algorithm copy the path of stations for each route and add the locations at the end of the route preserving the order in which each station is visited (line 8). At this point, a route r in solution s' has a length of $2 \cdot n$ stations where n is the real number of locations to visit. Particularly, for the route presented in the example in Table 5.2, s'_r is $16 \rightarrow 2 \rightarrow 7 \rightarrow 4 \rightarrow 13 \rightarrow 16 \rightarrow 2 \rightarrow 7 \rightarrow 4 \rightarrow 13$. Starting from the station in the first position of route r, up to n values for l, l^+ and l^- are computed in order to check whether vehicle loads do not exceed the maximum capacity (see lines 10 to 24). If $l^+ - l^- > Q$ for any station b as satellite depot, then array $depots$ reports b as an unfeasible location for intermediate depot.

Algorithm 5.4: Checking feasibility for satellite depots

```
1:  function: feasibleSatelliteDepots(s)
2:  for r = 1 to |K| − 1 do
3:      s'_r ← s_r
4:      n ←length(s_r) − 1
5:      depots_[s'_r,[1]] ←True
6:      for p = 2 to n do
7:          depots_[s'_r,[p]] ←True
8:          s'_r,[n+p−1] ← s_r,[p]
9:      end for
10:     for i = 1 to n do
11:         l ← q_s'r,[i+1]
12:         l+ ← l, l− ← l
13:         for j = 2 to n do
14:             l ← l + q_s'r,[i+j]
15:             if l < l− then
16:                 l− ← l
17:             else
18:                 if l > l+ then
19:                     l+ ← l
20:                 end if
21:             end if
22:             if l+ − l− > Q then
23:                 b ← s'_r,[i]
24:                 depots_[b] ←False
25:                 break
26:             end if
27:         end for
28:     end for
29: end for
30: return depots
```

5.4.6 Central route construction

The function `buildCentralRoute` in Algorithm 5.1 attempts to build a central route to integrate secondary routes in s by visiting at least one of the candidates (i.e., array $depots$) per route. To do so, the central route construction follows some of the ideas in the greedy randomized construction algorithm presented in Section 5.4.1. Nonetheless, the strategy to find a central route is not based on a greedy selection of stations. At this step, we skip the use of restricted candidate lists, then a randomized construction arises. A brief description for a central route construction follows.

(*i*) Select the operation center or depot (i.e., location 0) as the first visited location. Let the number of visited stations be equal to one ($p \longleftarrow 1$). Let r be the path for central route. Thus, $r_{[p]} \longleftarrow 0$ and let also $i \leftarrow 0$

(*ii*) Let $\mathcal{C}$ be the set of all feasible stations to visit in non-visited secondary routes after station i is served within the central route. Note that to add a station j as the p^{th} location in central route, constraint (5.24) must be met and also j must be a candidate for satellite depot (i.e., $depots_{[j]} =$**True**).

(*iii*) Choose station j from $\mathcal{C}$ at random. Check j as visited as well as the secondary route containing j. Let $i \leftarrow j$

(*iv*) Let $p \longleftarrow p+1$ and $r[p] \longleftarrow j$. If $p < |\mathcal{K}|$ then define $i \longleftarrow j$ and go to step (*ii*), else stop and return r as central route.

5.4.7 Global feasibility verification

Procedures described in Sections 5.4.3 and 5.4.6 ensure vehicle capacity feasibility for each secondary route and central route, respectively. Nonetheless, pickups and deliveries quantities at satellite depots are not fixed yet. Thus, connections between central and secondary routes via satellite depots may end up in demand requirement violations for a global solution. As mentioned before, at satellite depots split demand operations are allowed: some extra bikes can be temporally delivered at satellite depots using the central route vehicle, and then, the vehicle devoted to secondary route must pickup those extra bikes in order to ensure demand, as in equation (5.15). Therefore, once central and secondary routes are connected, a straightforward method to check whether a solution s is feasible for demand requirements is to find values for variables l in (5.15) for all known arcs ($\mathcal{A}'$) within s. To do so, a set of linear constraints are stated as follows:

$$\sum_{j:(j,i)\in\mathcal{A}'} l_{ji} - \sum_{j:(i,j)\in\mathcal{A}'} l_{ij} = q_i, \qquad \forall\, i \in \mathcal{N} \tag{5.25}$$

$$0 \leq l_{ij} \leq Q, \qquad \forall\, (i,j) \in \mathcal{A}' \tag{5.26}$$

Equations in (5.25) aim to meet stations requirement assigning loads for each used arc (i, j) in s. Expressions in (5.26) check for feasible loads in all routes. For the hybrid constructive solution strategy, if conditions in (5.25) and (5.26) are not met, then the solution is discarded and function `isFeasible` for solution s in Line 12 of Algorithm 5.1 returns a **False** value.

5.5 A set partitioning problem based matheuristic

This section proposes a second solution strategy for the 2E–BRPSD based on a two-phase matheuristic algorithm. The first phase creates a large number of central and secondary routes. Secondly, an enhanced SPP is solved to find a solution for the 2E–BRPSD. Algorithm 5.5 describes the procedure. Phase I is outlined in lines 1 to 15 and uses some of the procedures described in Section 5.4. The SPP mathematical model is solved in line 17 as Phase II. Details on this matheuristic follow.

5.5.1 Sets of central and secondary routes for the 2E–BRPSD

Phase I starts defining sets $\mathcal{C}$ and $\mathcal{S}$ to store central and secondary routes, respectively. For a number of predefined iterations (`maxIterations`), the algorithm builds (un)feasible central and secondary routes. Firstly, for secondary routes, the greedy randomized procedure described in Section 5.4.1 finds feasible and unfeasible Hamiltonian tours (see lines 4 and 7, respectively). Each one of these tours is improved via VND (Section 5.4.2) and then, several secondary routes (s) are obtained with split algorithm (Section 5.4.3). These routes obtained from T are added to set $\mathcal{S}$ as in Line 13. Moreover, with the aim to have a larger number of routes in $\mathcal{S}$, the algorithm copies each path and remove the last visited station in each route within the function `removeLastNode`. This strategy

Algorithm 5.5: SPP based algorithm for the 2E–BRPSD

```
1:  C, S ← ∅
2:  for i = 1 to maxIterations do
3:      if i <maxIterations·β then
4:          T ← greedyRandomizedAlgorithm(True, seed)
5:          c ← buildCentralRoute(True, |K|, seed)
6:      else
7:          T ← greedyRandomizedAlgorithm(False, seed)
8:          c ← buildCentralRoute(False, |K|, seed)
9:      end if
10:     C ← C ⋃ c
11:     T ← VND(T)
12:     s ← split(T)
13:     S ← S ⋃ s
14:     s ← removeLastNode(s)
15:     S ← S ⋃ s
16: end for
17: s* ← SPP_{2E−BRP}(C, S)
18: return s*
```

may end up in a higher probability of finding better solutions. Needless to say, removing the last visited station from a route does not increase its feasibility violation degree (δ).

Secondly, as shown in Algorithm 5.5, central routes are created before secondary routes are split. Thus, in this solution strategy, the pure randomized procedure described in Section 5.4.6 cannot be considered. Since the general purpose of Phase I is to generate a large set of central routes ($\mathcal{C}$), it is possible to adapt the function `greedyRandomizedAlgorithm` initially described in Section 5.4.1 for secondary routes generation. For the central routes case, the length for array T is $|\mathcal{K}|$ and the first visited location is the operation center. Then, $T[1] \leftarrow 0$ in step (i) from Section 5.4.1. The greedy construction remains as described in steps (ii) and (iii) and the algorithm stops if $p = |\mathcal{K}|$ (see step (iv) in Section 5.4.1). Each central route generated in lines 5 and 8 in Algorithm 5.5 is added to set $\mathcal{C}$.

5.5.2 SPP mathematical formulation for the 2E–BRPSD

Once sets $\mathcal{C}$ and $\mathcal{S}$ are completed in Phase I, one central route and $|\mathcal{K}|-1$ secondary routes, must be selected. To do so, a SPP based model able to provide a solution for the 2E–BRPSD is proposed. This model is an MILP in which apart from sets $\mathcal{N}$, $\mathcal{S}$ and $\mathcal{C}$, a set $\mathcal{A}_s$ is defined as the used arcs in secondary route s ($s \in \mathcal{S}$). Similarly, $\mathcal{A}_c$ is the set of arcs used in the central route c ($c \in \mathcal{C}$). Parameters h_c and g_s denote the cost of central route c and secondary route s, respectively. a_{ic} is a binary parameter that takes the value of one if station i belongs to central route c and zero otherwise. Similarly, parameter b_{is} is equal to one if i is included in secondary route s. Decision variable y_c takes the value of one if central route c is selected in the solution, and zero otherwise. In a similar way, binary decision variable x_s works for secondary route s. d_{is} is also a binary variable that is equal to one if station i is selected as a satellite depot for s and takes the value of zero in other case. Finally, variables l^c_{ij} and l'^s_{ij} determine the number of bikes in the vehicle when traversing the arc (i,j) in the central route c and secondary route s, respectively. The proposed MILP (SPP_{2E-BRP} in Algorithm 5.5) for the SPP based matheuristic follows.

$$\min f = \sum_{c\in\mathcal{C}} h_c \cdot y_c + \sum_{s\in\mathcal{S}} g_s \cdot x_s \tag{5.27}$$

subject to,

$$\sum_{s\in\mathcal{S}} b_{is} x_s = 1, \qquad \forall\, i \in \mathcal{N}\setminus\{0\} \tag{5.28}$$

$$\sum_{s\in\mathcal{S}} x_s = |\mathcal{K}| - 1, \tag{5.29}$$

$$\sum_{c\in\mathcal{C}} y_c = 1, \tag{5.30}$$

$$\sum_{s\in\mathcal{S}}\left(\sum_{j:(j,i)\in\mathcal{A}_s} l'^{s}_{ji} - \sum_{j:(i,j)\in\mathcal{A}_s} l'^{s}_{ij}\right) + \sum_{c\in\mathcal{C}}\left(\sum_{j:(j,i)\in\mathcal{A}_c} l^{c}_{ji} - \sum_{j:(i,j)\in\mathcal{A}_c} l^{c}_{ij}\right) = q_i, \qquad \forall\, i\in\mathcal{N} \tag{5.31}$$

$$l'^{s}_{ij} \le Q\cdot x_s, \qquad \forall\, s\in\mathcal{S},\ (i,j)\in\mathcal{A}_s \tag{5.32}$$

$$l^{c}_{ij} \le Q\cdot y_c, \qquad \forall\, c\in\mathcal{C},\ (i,j)\in\mathcal{A}_c \tag{5.33}$$

$$\sum_{i\in\mathcal{N}} b_{is}\cdot d_{is} = x_s \qquad \forall\, s\in\mathcal{S} \tag{5.34}$$

$$\sum_{s\in\mathcal{S}} d_{is} = \sum_{c\in\mathcal{C}} a_{ic}\cdot y_c, \qquad \forall\, i\in\mathcal{N}\setminus\{0\} \tag{5.35}$$

$$d_{is}\in\{0,1\}, \qquad \forall\, s\in\mathcal{S},\ i\in\mathcal{N} \tag{5.36}$$

$$y_c\in\{0,1\}, \qquad \forall\, c\in\mathcal{C} \tag{5.37}$$

$$x_s\in\{0,1\}, \qquad \forall\, s\in\mathcal{S} \tag{5.38}$$

$$l'^{s}_{ij},\ge 0, \qquad \forall\, s\in\mathcal{S},\ (i,j)\in\mathcal{A}_s \tag{5.39}$$

$$l^{c}_{ij}\ge 0, \qquad \forall\, c\in\mathcal{C},\ (i,j)\in\mathcal{A}_c \tag{5.40}$$

This enhanced SPP formulation aims to minimize the total traveling cost with the objective function in (5.27). Expressions in (5.28) force the model to include each station in one of the secondary routes. The number of these secondary routes is limited to the available vehicles as shown in constraint (5.29). Equation in (5.30) guarantees that a single central route is selected from set $\mathcal{C}$. Equations in (5.31) model the demand fulfilment at each station. Since a station demand can be meet using up to two vehicles (i.e, stations selected as satellite depots),it is necessary to consider loads from vehicles in central and secondary routes. Inequalities in (5.32) and (5.33) forbid vehicle loads that exceed the capacity Q in secondary and central routes, respectively. Equations in (5.34) force the model to select a station i as satellite depot for each secondary route s only if i belongs to s. Expressions in (5.35) forbid to select a station i as satellite depot for any secondary route s if i does not belong to a selected central route c. Lastly, expressions (5.36) to (5.40) model the domain of decision variables in the mathematical model.

Finally, as shown in Algorithm 5.5 (line 17), SPP_{2E-BRP} MILP returns a 2E–BRPSD solution (s^*). This solution s^* is also reported as the overall solution found via SPP based matheuristic strategy. Despite the large number of central and secondary routes heuristically generated and stored in $\mathcal{C}$ and $\mathcal{S}$, solving SPP_{2E-BRP} MILP may end up with no feasible solution. Since central

and secondary routes are generated independently, it is possible that none of the routes in $\mathcal{C}$ is able to complete demand requirements for any subset of secondary routes from $\mathcal{S}$.

5.6 Generalized traveling salesman problem based matheuristic for the 2E–BRPSD

A third solution strategy for the 2E–BRPSD is presented in this section. This strategy can be also described in two phases. The first phase aims to find a large set of candidates for secondary routes. Next, in the second phase, a classical SPP IP model finds the *best* combination of these routes for the 2E–BRPSD. This combination is then used to determine a central route via a GTSP MILP. These two mathematical models are solved repeatedly until a stop criterion is met.

Algorithm 5.6: GTSP-based algorithm for the 2E–BRPSD

$\mathcal{S} \leftarrow \emptyset$, $f^* \leftarrow +\infty$, $\lambda \leftarrow 0$
for $i = 1$ to maxIterations **do**
    if $i <$ maxIterations$\cdot\beta$ **then**
        $T \leftarrow$ `greedyRandomizedAlgorithm`(**True**, seed)
    else
        $T \leftarrow$ `greedyRandomizedAlgorithm`(**False**, seed)
    end if
    $T \leftarrow \texttt{VND}(T)$
    $s \leftarrow \texttt{split}(T)$
    $\mathcal{S} \leftarrow \mathcal{S} \bigcup s$
    $s \leftarrow \texttt{removeLastNode}(s)$
    $\mathcal{S} \leftarrow \mathcal{S} \bigcup s$
end for
while true **do**
    $\mathcal{P} \leftarrow \texttt{SPP}(\mathcal{S}, \lambda)$
    $s' \leftarrow \texttt{GTSP}_{2E-BRP}(\mathcal{P}) \bigcup \mathcal{P}$
    if $f(s') < f^*$ **then**
        $f^* \leftarrow f(s')$
        $s^* \leftarrow s'$
    end if
    $\lambda \leftarrow f(\mathcal{P}) + \Delta$
    if $\lambda > f^*$ **then**
        break
    end if
end while
return s^*

Algorithm 5.6 depicts the complete strategy. Starting with an empty set of secondary routes ($\mathcal{S}$) and an incumbent for the best known solution (f^*), the algorithm generates `maxIterations` Hamiltonian tours as described in Section 5.4.1. Similar to Phase I procedure described in Section 5.5.1 for secondary routes, each tour T is improved and split in order to add obtained paths to $\mathcal{S}$ (lines 2 to 13 in Algorithm 5.6). Then, Phase II starts and it is shown in lines 14 to 25 in Algorithm 5.6. In Phase II, $|\mathcal{K}|-1$ secondary routes ($\mathcal{P}$) are obtained solving a SPP IP in which a minimum value λ for total traveling cost over the secondary routes is imposed. Value for λ changes iteratively with the aim to find different configurations for secondary routes. The motivation to solve the SPP IP in an iterative way (and so, the GTSP MILP in line 16 of Algorithm 5.6) relies on the fact that the subset of secondary routes with minimum traveling cost does not ensure a minimum total cost for the complete problem (once a central route is added to the solution). After secondary routes are stored in $\mathcal{P}$, GTSP MILP finds a central route to provide a complete 2E–BRPSD solution, s'. If s' improves the incumbent, then the best solution found is updated (lines 17 to 20). Finally, the algorithm solves again SPP IP and GTSP MILP until a value for λ (total traveling cost over secondary routes) exceeds the best solution found for the 2E–BRPSD. New values for λ are computed as a small increase on traveling cost for secondary routes. Details on the mathematical formulations for this second phase follow.

5.6.1 Set partitioning problem for secondary routes

In line 15 of Algorithm 5.6, $|\mathcal{K}|-1$ secondary routes in $\mathcal{P}$ are stored. To find these routes, the algorithm calls a classical SPP IP forcing the model to find an objective function value greater or equal than λ. Using the aforementioned notation from Section 5.5.2, the SPP IP for secondary routes selection is described as follows:

$$\min f = \sum_{s\in\mathcal{S}} g_s \cdot x_s \tag{5.41}$$

subject to,

$$\sum_{s\in\mathcal{S}} b_{is} \cdot x_s = 1, \qquad \forall\, i \in \mathcal{N}\setminus\{0\} \tag{5.42}$$

$$\sum_{s\in\mathcal{S}} x_s = |\mathcal{K}|-1, \tag{5.43}$$

$$\sum_{s\in\mathcal{S}} g_s \cdot x_s \geq \lambda, \tag{5.44}$$

$$x_s \in \{0,1\}, \qquad \forall\, s \in \mathcal{S} \tag{5.45}$$

The objective function in (5.41) minimizes the total traveling cost for secondary routes. Equations in (5.42) force the model to include each station in a secondary route while (5.43) guarantees $|\mathcal{K}| - 1$ secondary routes in the solution. Constraint in (5.44) imposes a lower bound for the total traveling cost. Finally, expressions (5.45) describe the domain of decision variables. Based on obtained values for variables x, set $\mathcal{P}$ is updated and then, used as an input for the GTSP MILP.

5.6.2 An enhanced GTSP mathematical formulation

The GTSP is a well-known extension of the TSP in which a classification of each element in the set of nodes (e.g., cities, BSS stations) into groups is given. The GTSP aims to find a minimum length tour by visiting exactly one location for each group (Noon, 1988). Inspired in this problem, a GTSP formulation is developed. This formulation aims to find central routes using secondary paths in $\mathcal{P}$ as the groups required to solve the model.

Similarly to model described in equations (5.1) – (5.21) for the 2E–BRPSD, $\mathcal{N}$ and $\mathcal{A}$ denote the set of stations and arcs, respectively. Additionally, $\mathcal{N}_p$ and $\mathcal{A}_p$ denote the set of stations and arcs predefined for secondary route p where $p \in \mathcal{P}$. The binary parameter ϕ_i denotes whether station i can be a feasible satellite depot for secondary route as explained in Section 5.4.5. Again, Q and q_i denote the vehicle capacity and the demand at each station, respectively. Since the adapted GTSP aims to determine a unique central route for the 2E–BRPSD, decision variables do not depend on set $\mathcal{K}$. Then, y_{ij} takes the value of one if arc (i, j) is traversed within the central route. Decision variables l_{ij} and l'_{ij} represent the number of bikes to transport from i to j in the central and secondary routes, respectively. It is worth to mention that each station i is included in just one of the routes in $\mathcal{P}$. Finally, and as described in model (5.1) – (5.21), variables z_{ij} keep track of the order in which arc (i, j) is used in the solution. The proposed MILP to determine central routes for the 2E–BRPSD is as follows:

$$\min f = \sum_{(i,j)\in\mathcal{A}} c_{ij} \cdot y_{ij} \tag{5.46}$$

subject to,

$$\sum_{j\in\mathcal{N}} y_{0j} = 1, \tag{5.47}$$

$$\sum_{j\in\mathcal{N}} y_{j0} = 1, \tag{5.48}$$

$$\sum_{\substack{j\in\mathcal{N}\\ j\neq i}} y_{ij} = \sum_{\substack{j\in\mathcal{N}\\ j\neq i}} y_{ji}, \qquad \forall\, i \in \mathcal{N} \tag{5.49}$$

$$\sum_{i\in\mathcal{N}_p} \sum_{j\notin\mathcal{N}_p} y_{ij} = 1, \qquad \forall\, p \in \mathcal{P} \tag{5.50}$$

$$\sum_{j\in\mathcal{N}} y_{ij} \leq \phi_i, \qquad \forall\, i \in \mathcal{N}\setminus\{0\} \tag{5.51}$$

$$\sum_{j\in\mathcal{N}} l_{ji} + \sum_{j\in\mathcal{N}} l'_{ji} - \sum_{j\in\mathcal{N}} l_{ij} - \sum_{j\in\mathcal{N}} l'_{ij} = q_i, \qquad \forall\, i \in \mathcal{N} \tag{5.52}$$

$$\sum_{j\in\mathcal{N}} l_{j0} - \sum_{j\in\mathcal{N}} l_{0j} = q_0, \tag{5.53}$$

$$l_{ij} \leq Q \cdot y_{ij}, \qquad \forall\, (i,j) \in \mathcal{A} \tag{5.54}$$

$$\sum_{j\in\mathcal{N}} l_{ij} + \sum_{j\in\mathcal{N}} l'_{ij} \leq Q, \qquad \forall\, i \in \mathcal{N} \tag{5.55}$$

$$l'_{ij} \leq Q, \qquad \forall\, (i,j) \in \mathcal{A}_p,\ p \in \mathcal{P} \tag{5.56}$$

$$l'_{ij} = 0, \qquad \forall\, (i,j) \notin \mathcal{A}_p,\ p \in \mathcal{P} \tag{5.57}$$

$$\sum_{j\in\mathcal{N}} z_{ji} - \sum_{j\in\mathcal{N}} z_{ij} = \sum_{j\in\mathcal{N}} y_{ij}, \qquad \forall\, i \in \mathcal{N} \tag{5.58}$$

$$z_{ij} \leq |\mathcal{K}| \cdot y_{ij}, \qquad \forall\, (i,j) \in \mathcal{A} \tag{5.59}$$

$$y_{ij} \in \{0,1\}, \qquad \forall\, (i,j) \in \mathcal{A} \tag{5.60}$$

$$l_{ij}, l'_{ij}, z_{ij} \geq 0, \qquad \forall\, (i,j) \in \mathcal{A} \tag{5.61}$$

The objective function in (5.46) seeks to minimize the traveling cost for central route. Equations (5.47) and (5.48) ensure that the vehicle leaves and returns to the BSS operations center (station 0). Expressions in (5.49) force the vehicle to leave a station if it is visited. With equations in (5.50), each group (i.e., each secondary route) is connected to central route visiting one station in the group. Constraints in (5.51) guarantee that the vehicle can go into a station only if the station is a feasible depot for its associated secondary route. Expressions in (5.52) and (5.53) fix suitable values for vehicle loads in order to meet the demand at each station and the operations center, respectively. Constraints (5.54) to (5.57) force the model to satisfy the maximum capacity of vehicles for central and secondary routes. Constraints (5.58) and (5.59), as a modified version of the ideas in Miller et al. (1960), avoid subtours in feasible solutions for the 2E–BRPSD. Finally, (5.60) and (5.61) describe the nature of decision variables.

5.6.3 An alternative procedure for secondary route selection

As shown in Algorithm 5.6, Phase II of the solution strategy aims to find sets of secondary routes by increasing λ value in SPP formulation (equations (5.41) – (5.45)). This procedure allows to improve objective function value by varying the selected secondary routes, and consequently the obtained central route via GTSP MILP. As an alternative way to generate different sets of secondary routes, it is possible to design a procedure based on the nodes in a route and their position within the path. As an example, Figure 5.3 shows two routes; the number above each node represents the position of the node through the route path. An *identifier I* as a way to differentiate one route from another, can be computed as:

$$I = \sum_{i \in \mathcal{N}} i \cdot \rho_i \tag{5.62}$$

where i is the node number and ρ_i denotes the position of node i within the path. Thus, *identifier I* for routes 1 and 2 in Figure 5.3 is:

$$I_1 = (1 \cdot 5) + (2 \cdot 6) + (3 \cdot 7) + (4 \cdot 8) + (5 \cdot 9) = 115 \tag{5.63}$$

$$I_2 = (10 \cdot 1) + (11 \cdot 2) + (12 \cdot 3) = 68 \tag{5.64}$$

Since Phase II for the GTSP-based solution strategy is solved in an iterative way, for each iteration t, an identifier for the set of selected secondary routes is calculated as:

$$Iv(t) = \sum_{p \in \mathcal{P}} I_p \cdot x_p \tag{5.65}$$

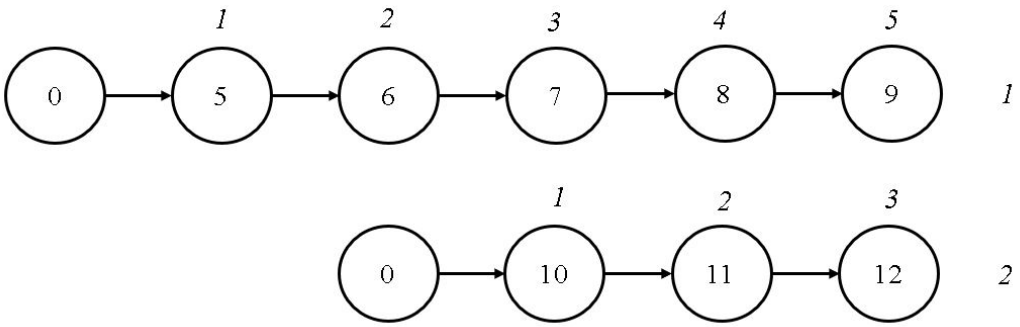


Figure 5.3: Example for route index computation

If different sets of secondary routes are expected at each iteration t of Phase II of solution strategy, then $Iv(t) \neq Iv(t')$ for $t \neq t'$ must be met. Nonetheless a linear version of this relation is required to replace equation in (5.44). A linear approach to reach different identifiers through t iterations can be stated as:

$$\sum_{p \in \mathcal{P}} I_p \cdot x_p \leq Iv(t') - 1 + M \cdot (1 - y_t), \qquad \forall\, t' < t \tag{5.66}$$

$$\sum_{p \in \mathcal{P}} I_p \cdot x_p \geq Iv(t') + 1 - M \cdot y_t, \qquad \forall\, t' < t \tag{5.67}$$

where y_t is a binary variable that takes the value of one if the secondary routes identifier (i.e., $\sum_{p \in \mathcal{P}} I_p \cdot x_p$) is less than the identifier of secondary routes at iteration t'. Note that the size of SPP model adding equations (5.66) and (5.67) increases significantly as the number of iterations also increases. However, given the definition of variable y_t and the structure of equations (5.66) and (5.67), it is possible to add valid inequalities for each pair of iterations. Therefore for a pair of iterations t' and t, if $Iv_t < Iv'_t$ then

$$y_t \leq y'_t \tag{5.68}$$

and also, in a similar way if $Iv_t > Iv'_t$ then,

$$y_t \geq y'_t \tag{5.69}$$

Regarding computational performance it is worth to mention that constraints in (5.66) to (5.69) are added at each iteration but SPP mathematical model is defined once at first iteration. Thus, solver does not build the mathematical formulation from scratch each time the SPP is called.

5.7 Computational experiments

This section presents and discusses the main results on MILP outlined in Section 5.3 and on solution strategies for the 2E–BRPSD described in Sections 5.4, 5.5 and 5.6. Firstly, a description on instances to test the solution strategies is presented. Then, this section summarizes results on MILP for the 2E–BRPSD solving the model via commercial solver and setting a maximum computation time of 3600 seconds. Thirdly, results on hybrid strategies are discussed. All mathematical models were solved using Gurobi 9.1. The algorithms were coded in Visual C++ for Windows 10 running

on an Intel Core i7 at 1.80GHz with 16.00 gigabytes of RAM. For each run on hybrid strategies, maximum computation time is set to 3600 seconds. After several tests, values for parameter `MaxIterations` were set to 250 for all solution strategies. Similarly, for GTSP-based algorithm, λ was set to 0.02.

5.7.1 Data sets

Mathematical model and solution strategies were tested on two different sets of instances. The first one is the set of well-known instances previously used to solve the 1–PDTSP in Chapter 3 (Hernández-Pérez and Salazar-González, 2004a). The 2E–BRPSD is also solved with a set of instances adapted from the operation of EnCicla in Medellín, Colombia (Palacio and Rivera, 2022).

1–PDTSP benchmark instances

The first set of instances are available at `http://hhperez.webs.ull.es/PDsite/`. These instances were previously used to solve the 1–PDTSP as in Hernández-Pérez et al. (2009); Hernández-Pérez and Salazar-González (2004a); Palacio and Rivera (2019). For these instances, $|\mathcal{N}| \in \{20, 30, 40, 50, 60, 100, 200, 300\}$ and demand on stations (q) may vary from -10 to 10. In 2E–BRPSD experiments, vehicle capacity is ten ($Q = 10$). As in Chapters 3 and 4, this value for Q allows to set up the hardest configuration for the instances since it coincides with the largest demand on a station (Hernández-Pérez et al., 2009). Since the number of stations in these instances vary significantly, fleet size is computed as function of the number of stations as follows:

$$|\mathcal{K}| = \left\lceil \frac{|\mathcal{N}|}{\gamma} \right\rceil + 1 \tag{5.70}$$

where γ is the number of expected stations to visit in a secondary route. Two different scenarios for route length are explored setting $\gamma = \{8, 10\}$. Thus, fleet size for experiments may take two different values as shown in Table 5.4. Finally, for each value for $|\mathcal{N}|$, ten instances named from A to J are available. Details about the instances generation may be found in Hernández-Pérez and Salazar-González (2004a) and Hernández-Pérez and Salazar-González (2004b).

EnCicla BSS instances

The second set of instances to solve is based on EnCicla operation, the public BSS program in Medellín, Colombia. This set includes those instances described in Section 3.4.1. Since Medellín is

Table 5.4: Number of vehicles for 2E–BRPSD instances

γ	$\lvert\mathcal{K}\rvert$							
	20	30	40	50	60	100	200	300
8	4	5	6	8	9	14	26	31
10	3	4	5	6	7	11	21	26

divided into six zones, all EnCicla instances for the 2E–BRPSD consider six vehicles. (i.e., one vehicle per zone). Recall that vehicles capacity is set to 45 ($Q = 45$).

5.7.2 Results on 2E–BRPSD MILP

Table 5.5 shows results for total traveling costs when solving benchmark instances with up to 50 stations. Firstly, for instances with 20 stations and three vehicles, it is possible to find the optimal solution (bold numbers) within the maximum computation time. If four vehicles are used only four out of ten instances are solved optimally. Then, for instances with 30 and 40 stations, the solver reports at least one feasible solution in all cases. However, for most of the larger instances (50 stations), no feasible solution is reported by the optimizer within one hour. Only for five of the 20 instances with 50 stations, a feasible solution is retrieved. In a similar way, Table 5.6 reports gaps retrieved by the solver once it stops after 3600 seconds. These gaps are computed as: $gap(\%) = \frac{UB-LB}{UB} \cdot 100$ where UB and LB are the best solution found and the lower bound reported by the solver, respectively. For instances with 20 stations, gaps do not exceed 11.97%. On the other hand, for instances with 30 and 40 stations, gaps increase up to 32.30% and 41.62%, respectively. Average gaps over the 20 runs for 30-station instances is 19.24% and it increases up to 33.66% for instances with 40 stations. Since gaps for instances with 50 stations vary from 38.81% to 49.73%, larger instances are not solved via the commercial solver. Finally, CPU time is also reported in parenthesis for instances with 20 stations solved optimally.

5.7.3 Results on matheuristic algorithms

This section summarizes results on matheuristic algorithms for benchmark instances. Since MILP is able to solve instances with up to 40 stations, matheuristic algorithms performance is compared with commercial solver results. Then, for larger instances, only solutions found via matheuristic algorithms are presented since solutions obtained via commercial optimizer are not available.

Tables 5.7, 5.8 and, 5.9 show results for benchmark instances with 20, 30 and 40 stations, respectively. In each table, results for the ten instances are reported varying the fleet size as described

in Table 5.4. Firstly, the optimal solution or upper bound retrieved by the commercial solver is reported in column MILP. Then, results on four algorithms are presented. Groups of columns Constructive, SPP-Math, GTSP-Cost and, GTSP-Id describe results for the hybrid constructive algorithm, the SPP-based matheuristic, the GTSP-based algorithm when traveling cost target increases iteratively and, GTSP- based algorithm with route identifier, respectively. Columns Min. and Avg. show minimum and average total traveling costs retrieved by each solution strategy over ten runs. Lastly, in group of columns Improvement (%) MILP, improvements for each strategy over MILP solution are reported. These improvements are computed as follows:

$$Improvement\ Construct._{MILP}(\%) = \frac{MILP - Min._{constructive}}{MILP} \cdot 100 \tag{5.71}$$

$$Improvement\ SPP-Math_{MILP}(\%) = \frac{MILP - Min._{SPP-Math}}{MILP} \cdot 100 \tag{5.72}$$

$$Improvement\ GTSP-Cost_{MILP}(\%) = \frac{MILP - Min._{GTSP-Cost}}{MILP} \cdot 100 \tag{5.73}$$

$$Improvement\ GTSP-Id_{MILP}(\%) = \frac{MILP - Min._{GTSP-Id}}{MILP} \cdot 100 \tag{5.74}$$

For 20, 30 and 40 stations, GTSP-Cost strategy outperforms constructive, SPP-Math and GTSP-Id algorithms, on average. Particularly, for all instances with 20 stations, GTSP-Cost finds the solution reported by commercial solver (optimal traveling costs if $|\mathcal{K}| = 3$). For a subset of 20 station instances, constructive, SPP-Math and GTSP-Id, are not able to find those best solution and the three matheuristic algorithms reports an increased average traveling cost if three and four vehicles

Table 5.5: MILP results for total traveling cost on 2E–BRPSD instances

		Instance									
$\|\mathcal{N}\|$	$\|\mathcal{K}\|$	A	B	C	D	E	F	G	H	I	J
20	3	**5287**	**5459**	**6456**	**6366**	**6842**	**5100**	**6002**	**6070**	**5189**	**4733**
	4	**5261**	**5713**	6952	6613	6764	**5382**	6129	6563	5502	**5104**
30	4	7134	7585	7779	7503	6936	6787	10509	8016	6352	8624
	5	7616	8255	7533	8125	6866	7044	10922	7701	6818	7562
40	5	9612	8784	9191	10031	10025	9830	10969	9143	9597	9194
	6	10003	9772	10321	10328	9100	10180	10137	9667	9461	9959
50	6	-	-	13919	13817	-	-	-	-	12111	-
	8	-	-	-	14889	-	-	-	-	13335	-

Table 5.6: MILP gaps and CPU times (s) for 2E–BRPSD instances

		Instance									
$\|\mathcal{N}\|$	$\|\mathcal{K}\|$	A	B	C	D	E	F	G	H	I	J
20	3	0.00% (715.36)	0.00% (118.20)	0.00% (394.83)	0.00% (181.52)	0.00% (3133.57)	0.00% (398.53)	0.00% (1242.53)	0.00% (187.93)	0.00% (201.28)	0.00% (101.77)
	4	0.00% (1549.29)	0.00% (2859.20)	11.97%	2.82%	4.06%	0.00% (2689.48)	9.33%	6.15%	4.00%	0.00% (2474.17)
30	4	14.09%	13.97%	20.87%	15.74%	11.76%	16.83%	16.79%	23.62%	12.90%	32.30%
	5	20.48%	22.01%	19.66%	22.92%	11.33%	21.15%	21.99%	21.94%	20.36%	24.11%
40	5	30.29%	32.81%	21.56%	25.96%	38.90%	36.82%	37.62%	32.13%	30.56%	34.19%
	6	35.85%	41.62%	34.62%	30.32%	34.38%	33.59%	33.02%	37.57%	32.91%	38.47%
50	6	-	-	44.49%	38.81%	-	-	-	-	43.02%	-
	8	-	-	-	42.28%	-	-	-	-	49.73%	-

are available. For 30 station instances, GTSP-Cost also outperforms the other proposed solution strategies. For this size of instances, GTSP-Cost improvements are positive, on average. Thus, some solutions retrieved by the matheuristic improve the total traveling cost reported by the optimizer solving MILP. Strategy GTSP-Id also finds better solutions than MILP for particular instances (H if $|\mathcal{K}| = 4$, B, G, H and J if $|\mathcal{K}| = 5$). Lastly, for 40 stations instances, SPP-Math, GTSP-Cost and GTSP-Id improve MILP solutions, on average. Furthermore, if average improvements are compared for both fleet size, GTSP-Cost keeps finding larger improvements than SPP-Math and GTSP-Id, on average.

Since MILP solutions are not available for $|\mathcal{N}| \geq 40$ and GTSP-Cost shows the best performance for instances with up to 40 stations, results for instances with 50 and 60 stations are compared over GTSP-Cost. Tables 5.10 and 5.11 summarize results for instances with 50 and 60 stations, respectively. For each instance and solution strategy a value p is reported. p denotes the probability of finding a solution over ten runs. GTSP-based matheuristic algorithms are able to find at least one solution each time the instance is solved. However, hybrid constructive algorithm and SPP-Math may not find a solution in each one of the ten runs. As a particular case, hybrid constructive algorithm did not found a feasible solution for instance D after ten runs of the algorithm, when fleet size is eight. Similar to smaller instances, when 50 stations are available, GTSP-Cost keeps outperforming other proposed algorithms. Therefore, improvement percentages for GTSP-Cost are computed over constructive algorithm, SPP-Math and GTSP-Id strategies. These improvements are computed as:

$$Improvement\ Construct._{GTSP-Cost}(\%) = \frac{Min._{Constructive} - Min._{GTSP-Cost}}{Min._{Constructive}} \cdot 100 \tag{5.75}$$

$$Improvement\ SPP-Math_{GTSP-Cost}(\%) = \frac{Min._{SPP-Math} - Min._{GTSP-Cost}}{Min._{SPP-Math}} \cdot 100 \tag{5.76}$$

$$Improvement\ GTSP-Id_{GTSP-Cost}(\%) = \frac{Min._{GTSP-Id} - Min._{GTSP-Cost}}{Min._{GTSP-Id}} \cdot 100 \tag{5.77}$$

Table 5.7: Comparative results on matheuristic algorithms for instances with 20 stations

	MILP	Constructive		SPP-Math		GTSP-Cost		GTSP-Id		Improvement (%) MILP			
		Min.	Avg.	Min.	Avg.	Min.	Avg.	Min.	Avg.	Construct.	SPP-Math	GTSP-Cost	GTSP-Id
$\|\mathcal{K}\| = 3$													
A	5287	5287	6047.10	5493	5702.90	5287	5496.79	5451	5728.53	0.00%	-3.90%	0.00%	-3.10%
B	5459	5459	6042.22	5459	5674.12	5459	5789.12	5897	6093.75	0.00%	0.00%	0.00%	-8.02%
C	6456	6983	7440.43	6886	6957.71	6456	7029.44	7155	7417.26	-8.16%	-6.66%	0.00%	-10.83%
D	6366	6366	7778.65	6366	6807.34	6366	6762.16	6850	7578.53	0.00%	0.00%	0.00%	-7.60%
E	6842	7515	7780.33	6852	6855.59	6842	7218.46	7220	7315.59	-9.84%	-0.15%	0.00%	-5.52%
F	5100	5100	5908.36	5100	5390.90	5100	5293.91	5501	5653.25	0.00%	0.00%	0.00%	-7.86%
G	6002	6002	6565.68	6180	6229.93	6002	6245.78	6295	6418.51	0.00%	-2.97%	0.00%	-4.88%
H	6070	6373	6869.10	6070	6400.81	6070	6354.20	6455	6837.58	-4.99%	0.00%	0.00%	-6.34%
I	5189	5189	6644.11	5601	5758.17	5189	5725.23	5785	5932.48	0.00%	-7.94%	0.00%	-11.49%
J	4733	4733	5769.49	4754	5224.92	4733	4998.17	5203	5370.53	0.00%	-0.44%	0.00%	-9.93%
Average										-2.30%	-2.21%	0.00%	-7.56%
$\|\mathcal{K}\| = 4$													
A	5261	5724	6152.23	5338	5669.13	5261	5428.37	5622	5758.00	-8.80%	-1.46%	0.00%	-6.86%
B	5713	6549	6765.19	5917	5956.82	5713	5849.41	6187	6409.74	-14.63%	-3.57%	0.00%	-8.30%
C	6952	7964	8389.09	6952	7236.81	6952	7137.91	7299	7606.27	-14.56%	0.00%	0.00%	-4.99%
D	6613	7100	8435.46	6613	6951.17	6613	7311.30	7561	8153.85	-7.36%	0.00%	0.00%	-14.34%
E	6764	7336	8363.58	7318	7492.56	6764	7530.38	7580	7704.57	-8.46%	-8.19%	0.00%	-12.06%
F	5382	6430	6785.53	5388	5766.89	5382	5627.20	5981	6174.51	-19.47%	-0.11%	0.00%	-11.13%
G	6129	6821	7111.10	6143	6425.37	6129	6385.77	6422	6794.55	-11.29%	-0.23%	0.00%	-4.78%
H	6563	6918	7259.87	6862	7023.32	6563	6846.32	7268	7342.53	-5.41%	-4.56%	0.00%	-10.74%
I	5502	5784	6947.93	5502	5950.14	5502	5979.19	5759	6055.75	-5.13%	0.00%	0.00%	-4.67%
J	5104	5880	6242.43	5213	5594.07	5104	5565.11	5805	5972.26	-15.20%	-2.14%	0.00%	-13.73%
Average										-11.03%	-2.03%	0.00%	-9.16%

Table 5.8: Comparative results on matheuristic algorithms for instances with 30 stations

	MILP	Constructive		SPP-Math		GTSP-Cost		GTSP-Id		Improvement (%) MILP			
		Min.	Avg.	Min.	Avg.	Min.	Avg.	Min.	Avg.	Construct.	SPP-Math	GTSP-Cost	GTSP-Id
$\|\mathcal{K}\| = 4$													
A	7134	8282	8665.88	7437	7770.90	7103	7353.61	7557	7998.25	-16.09%	-4.25%	0.43%	-5.93%
B	7585	8768	9009.90	7866	7989.13	7659	7826.61	8068	8337.50	-15.60%	-3.70%	-0.98%	-6.37%
C	7779	8131	8673.09	7920	8219.38	7534	7696.83	7812	8181.75	-4.53%	-1.81%	3.15%	-0.42%
D	7503	8882	9392.12	8906	9147.56	7503	8256.33	8099	8486.75	-18.38%	-18.70%	0.00%	-7.94%
E	6936	8051	8674.22	7285	7303.04	6987	7342.13	7085	7500.64	-16.08%	-5.03%	-0.74%	-2.15%
F	6787	8360	8777.39	7493	75654.10	6811	7228.80	7631	7819.20	-23.18%	-10.40%	-0.35%	-12.44%
G	10509	10840	11173.11	10424	10788.60	10338	10481.70	10575	10783.00	-3.15%	0.81%	1.63%	-0.63%
H	8016	8244	9083.75	7542	7622.13	7559	7896.66	7800	8080.74	-2.84%	5.91%	5.70%	2.69%
I	6352	7462	8458.68	6446	6446.79	6352	7054.54	7296	7488.55	-17.47%	-1.48%	0.00%	-14.86%
J	8624	8669	9108.97	8123	8249.40	7563	7919.97	8147	8193.75	-0.52%	5.81%	12.30%	5.53%
Average										-11.78%	-3.28%	2.12%	-4.25%
$\|\mathcal{K}\| = 5$													
A	7616	8729	9881.89	7980	8316.99	7345	7576.02	7883	8143.53	-14.61%	-4.78%	3.56%	-3.51%
B	8255	8984	9428.90	8317	8549.23	7879	8035.61	7855	8168.25	-8.83%	-0.75%	4.55%	4.85%
C	7533	8132	8912.14	7875	8128.62	7478	7529.10	7591	7843.06	-7.95%	-4.54%	0.73%	-0.77%
D	8125	8589	9709.80	8669	9202.37	7808	8110.34	8134	8285.27	-5.71%	-6.70%	3.90%	-0.11%
E	6866	8491	8851.34	7111	7922.40	7037	7341.53	7668	7713.75	-23.67%	-3.57%	-2.49%	-11.68%
F	7044	7960	8396.38	7489	7753.36	7013	7157.67	7081	7530.58	-13.00%	-6.32%	0.44%	-0.53%
G	10922	11243	11817.17	10471	10798.50	10049	10267.21	10320	10733.25	-2.94%	4.13%	7.99%	5.51%
H	7701	8555	9234.39	8232	8706.14	7376	7797.90	7632	7972.30	-11.09%	-6.90%	4.22%	0.90%
I	6818	7865	8824.90	7531	7760.71	6818	7240.83	6973	7400.43	-15.36%	-10.46%	0.00%	-2.27%
J	7562	8397	9309.87	7767	8154.20	7541	7659.62	7545	7852.77	-11.04%	-2.71%	0.28%	0.22%
Average										-11.42%	-4.26%	2.32%	-0.74%

Table 5.9: Comparative results on matheuristic algorithms for instances with 40 stations

	MILP	Constructive		SPP-Math		GTSP-Cost		GTSP-Id		Improvement (%) MILP			
		Min.	Avg.	Min.	Avg.	Min.	Avg.	Min.	Avg.	Construct.	SPP-Math	GTSP-Cost	GTSP-Id
$\|\mathcal{K}\| = 5$													
A	9612	9535	10236.30	9083	9138.92	8058	8331.46	8325	8717.80	0.80%	5.50%	16.17%	13.39%
B	8784	9988	10577.01	8618	9191.76	8148	8471.79	8306	9064.33	-13.71%	1.89%	7.24%	5.44%
C	9191	10215	10445.30	8833	9703.98	8186	8533.74	8945	9094.56	-11.14%	3.90%	10.93%	2.68%
D	10031	10966	11931.25	10178	11013.79	9379	9934.77	10102	10440.77	-9.32%	-1.47%	6.50%	-0.71%
E	10025	10253	11124.20	9873	10615.30	7932	8415.02	8732	9682.83	-2.27%	1.52%	20.88%	12.90%
F	9830	10792	11660.75	9553	9761.00	8520	9138.46	9573	9834.53	-9.79%	2.82%	13.33%	2.61%
G	10969	10910	11684.63	10184	10338.29	8831	9341.53	9660	10239.79	0.54%	7.16%	19.49%	11.93%
H	9143	9706	10472.38	9570	9740.38	8656	9031.04	8465	9178.77	-6.16%	-4.67%	5.33%	7.42%
I	9597	10143	10801.55	9488	10074.48	8607	8933.90	8805	9414.03	-5.69%	1.14%	10.32%	8.25%
J	9194	8572	10196.50	8751	8885.12	7559	7949.28	8496	8726.05	6.77%	4.82%	17.78%	7.59%
					Average					-5.00%	2.26%	12.80%	7.15%
$\|\mathcal{K}\| = 6$													
A	10003	9558	10847.40	9237	9731.21	8231	8664.21	9112	9292.34	4.45%	7.66%	17.71%	8.91%
B	9772	9491	11105.56	8988	9969.62	7887	8157.11	8902	9210.69	2.88%	8.02%	19.29%	8.90%
C	10321	9868	10560.88	9589	9836.23	8483	8852.38	9091	9243.84	4.39%	7.09%	17.81%	11.92%
D	10328	11206	11689.50	11209	11763.88	9746	9993.36	11014	11246.00	-8.50%	-8.53%	5.64%	-6.64%
E	9100	10949	12198.57	9648	10407.87	8443	8858.52	9332	9684.01	-20.32%	-6.02%	7.22%	-2.55%
F	10180	11259	12105.75	10140	10546.03	8782	9059.58	8916	9736.03	-10.60%	0.39%	13.73%	12.42%
G	10137	10989	12101.86	10032	10874.51	9037	9470.77	11647	11927.08	-8.40%	1.04%	10.85%	-14.90%
H	9667	9556	11280.20	9422	9870.43	8294	8573.48	9880	9880.05	1.15%	2.53%	14.20%	-2.20%
I	9461	10838	11521.92	9877	10198.56	8781	9323.94	10641	10641.07	-14.55%	-4.40%	7.19%	-12.47%
J	9959	9915	10817.11	9058	9650.08	8128	8415.66	8979	9282.09	0.44%	9.05%	18.39%	9.84%
					Average					-4.91%	1.68%	13.20%	1.32%

Table 5.10: Comparative results on matheuristic algorithms for instances with 50 stations

	Constructive			SPP-Math			GTSP-Cost			GTSP-Id			Improvement (%) GTSP-Cost		
	Min.	Avg.	p	Min.	Avg.	p	Min.	Avg.	p	Min.	Avg.	p	Construct.	SPP-Math	GTSP-Id
$\|\mathcal{K}\| = 6$															
A	10507	11432.27	1.00	9910	10322.25	1.00	8828	9262.04	1.00	9344	9451.80	1.00	15.98%	10.92%	5.52%
B	13686	13686.09	0.20	11550	12244.60	1.00	10953	11499.21	1.00	11621	12397.04	1.00	19.97%	5.17%	5.75%
C	12694	13397.76	0.30	11909	13112.41	1.00	10175	10988.73	1.00	11271	12776.10	1.00	19.84%	14.56%	9.72%
D	14836	15710.43	0.60	14537	15143.02	1.00	12027	12588.74	1.00	12805	13094.56	1.00	18.93%	17.27%	6.08%
E	14325	15527.88	0.50	12536	12789.34	1.00	10869	11154.55	1.00	11359	12637.01	1.00	24.13%	13.30%	4.31%
F	13068	13600.06	0.30	11207	11900.06	1.00	9961	10790.64	1.00	10807	11277.79	1.00	23.78%	11.12%	7.83%
G	12189	12319.07	0.30	11197	11911.14	1.00	9217	9583.70	1.00	9410	10161.41	1.00	24.38%	17.68%	2.05%
H	12654	13461.08	0.60	12006	13009.30	1.00	10935	11344.55	1.00	11603	13203.09	1.00	13.58%	8.92%	5.76%
I	12575	13229.01	0.60	12143	12386.30	1.00	9818	10523.62	1.00	10442	10700.00	1.00	21.92%	19.15%	5.98%
J	13319	14501.46	0.50	12992	13932.36	1.00	10714	11242.31	1.00	11946	14600.54	1.00	19.56%	17.53%	10.31%
Average													20.21%	13.56%	6.33%
$\|\mathcal{K}\| = 8$															
A	10875	12559.30	0.90	10519	10877.74	0.90	8335	8750.64	1.00	8836	9233.55	1.00	23.36%	20.76%	5.67%
B	13269	13269.00	0.10	12851	12851.00	0.10	10860	11164.67	1.00	11600	11797.53	1.00	18.16%	15.49%	6.38%
C	13702	14265.57	0.20	12259	12309.34	0.30	10367	10567.02	1.00	10924	11263.79	1.00	24.34%	15.43%	5.10%
D	-	-	0.00	14989	14989.00	0.10	12150	12361.84	1.00	12529	12960.56	1.00	-	18.94%	3.02%
E	13985	14457.50	0.50	13826	14582.04	0.70	11336	11578.24	1.00	11793	12291.50	1.00	18.94%	18.01%	3.88%
F	12619	14344.82	0.50	12377	13052.51	0.60	9013	9453.54	1.00	9504	9707.09	1.00	28.58%	27.18%	5.17%
G	14313	17059.09	0.20	9467	10624.94	0.90	8947	9081.45	1.00	10793	11422.09	1.00	37.49%	5.49%	17.10%
H	13105	14640.01	0.40	11725	13215.01	0.40	10470	10791.38	1.00	10699	11075.29	1.00	20.11%	10.70%	2.14%
I	12117	13998.01	0.20	12168	12414.02	0.30	9984	10176.71	1.00	10381	10786.03	1.00	17.60%	17.95%	3.82%
J	13586	14033.00	0.20	13507	15711.82	0.50	10395	10661.59	1.00	10376	10615.07	1.00	23.49%	23.04%	-0.18%
Average													23.56%	17.30%	5.21%

Table 5.11: Comparative results on matheuristic algorithms for instances with 60 stations

	Constructive			SPP-Math			GTSP-Cost			GTSP-Id			Improvement (%) GTSP-Cost	
	Min.	Avg.	p	Min.	Avg.	p	Min.	Avg.	p	Min.	Avg.	p	Construct. SPP-Math	GTSP-Id
$\|\mathcal{K}\| = 7$														
A	14940	15349.38	0.30	12875	13782.03	0.90	10317	11046.88	1.00	11069	11779.43	1.00	30.94% 19.87%	6.79%
B	15494	15542.51	0.20	12717	13053.14	1.00	10498	11151.41	1.00	11325	11619.09	1.00	32.24% 17.45%	7.30%
C	14106	14106.00	0.10	14936	16978.86	1.00	11505	11969.98	1.00	12599	13238.53	1.00	18.44% 22.97%	8.68%
D	15865	15882.51	0.20	17386	22682.46	0.70	13762	14136.38	1.00	13998	14694.34	1.00	13.26% 20.84%	1.69%
E	13771	14804.38	0.60	15525	21829.61	1.00	11540	12337.69	1.00	13048	13605.69	1.00	16.20% 25.67%	11.56%
F	-	-	0.00	13885	18524.39	0.33	11694	12118.93	1.00	12244	12547.03	1.00	- 15.78%	4.49%
G	-	-	0.00	13341	19478.44	1.00	11174	11973.69	1.00	12008	12148.08	1.00	- 16.24%	6.95%
H	13966	14824.76	0.30	12240	13431.83	1.00	10822	11284.28	1.00	11201	11568.02	1.00	22.51% 11.58%	3.38%
I	13899	14452.55	0.20	14455	16374.59	0.70	11571	12233.57	1.00	12131	13256.74	1.00	16.75% 19.95%	4.62%
J	14258	15094.71	0.30	13119	15037.28	1.00	11233	11738.31	1.00	11747	13037.34	1.00	21.22% 14.38%	4.38%
Average													21.45% 18.47%	5.98%
$\|\mathcal{K}\| = 9$														
A	12765	13207.08	0.20	12892	12986.67	0.50	10512	10754.11	1.00	11399	11428.03	1.00	17.65% 18.46%	7.78%
B	15075	15075.00	0.10	12995	13462.70	0.50	10826	10954.96	1.00	11299	11340.53	1.00	28.19% 16.69%	4.19%
C	14745	15273.59	0.20	-	-	0.00	10997	11354.23	1.00	11873	12130.04	1.00	25.42% -	7.38%
D	-	-	0.00	-	-	0.00	13254	13569.20	1.00	13447	14391.38	1.00	- -	1.44%
E	-	-	0.00	13265	13265.00	0.10	11580	11973.20	1.00	12283	12578.73	1.00	- 12.70%	5.72%
F	13311	13311.00	0.10	-	-	0.00	10683	11263.07	1.00	11192	11470.39	1.00	19.74% -	4.55%
G	-	-	0.00	13796	13796.00	0.10	10839	11220.00	1.00	11278	11873.68	1.00	- 21.43%	3.89%
H	-	-	0.00	12935	13635.32	0.80	10863	11160.21	1.00	11078	11393.74	1.00	- 16.02%	1.94%
I	16265	16265.00	0.10	14472	14472.00	0.10	11423	11727.49	1.00	11859	12111.42	1.00	29.77% 21.07%	3.68%
J	14017	14243.58	0.20	14976	15262.56	0.20	10516	11089.58	1.00	11721	11921.43	1.00	24.98% 29.78%	10.28%
Average													24.29% 19.45%	5.08%

As shown in Table 5.10, GTSP-Cost outperforms on average, up to 23.56% and 17.30% results on hybrid constructive algorithm and SPP-Math, respectively. For instances with 60 stations reported in 5.11, solution strategies performance does not change significantly. Improvement percentage for GTSP-Cost over the other three strategies are computed as in equations (5.75)–(5.77). On average, GTSP-Math reports lower values for traveling costs when compared with constructive algorithm, SPP-Math and GTSP-Id. GTSP-Cost outperforms up to 24.29% and 19.45% on average, constructive algorithm and SPP-Math, respectively. For six instances, the hybrid constructive algorithm is not able to find any feasible solution after ten runs. In a similar way, a solution is not reported for three instances via SPP-Math. GTSP-based algorithms are able to find at least one solution within each run of the algorithm.

For constructive algorithm and SPP-Math, the number of runs where a solution may be found, decreases as the number of stations and vehicles increase. Thus, for large instances with 100, 200 and 300 stations, Table 5.12 summarizes results only for GTSP-Cost algorithm. Despite it is possible to find at least one solution each time a 100 stations instance is solved, for instances with 200 and 300 stations, p starts to decrease. An experiment for instances with 400 station were conducted but none of the proposed algorithms was able to retrieve a feasible solution.

Table 5.13 summarizes average computation times for each size of instances and each solution strategy. As expected, hybrid constructive algorithm requires the least computational effort since no exact optimization procedure is performed within the strategy. Nonetheless, as mentioned before, the constructive approach is not able to retrieve solutions for instances where $|\mathcal{N}| \geq 100$. For instances with 20 and 30 stations, GTSP-Cost requires the largest computation times among the proposed strategies. However, for instances with 40, 50 and 60 stations SPP-Math requires more computational effort. On the other hand and despite GTSP-Cost ends its search on 200 and 300 stations instances in one hour (maximum computation time), recall that this strategy outperforms in terms of solution quality, the other three algorithms when $|\mathcal{N}| \leq 60$.

Tables 5.14, 5.15 and 5.16 depict the computational effort distribution per algorithm component for hybrid constructive, SPP-Math and GTSP-based strategies, respectively. The computational effort in hybrid constructive algorithm relies on VNDs procedures. Despite the checking feasibility for satellite depot procedure is based on a linear constraint satisfaction problem. This step does not requires more than 15.45% of total computation time. Since functions within SPP-Math and GTSP-based algorithms solve MILPs, computational effort distribution vary when comparing with hybrid constructive algorithm. As expected, mathematical models requires a greater computation time than heuristic functions as Split, local search and even VND. MILP solved in SPP-Math requires at least 54.00% of the total computation time. For GTSP-based algorithms (GTSP-Cost

Table 5.12: Results on matheuristic algorithms for instances with $|\mathcal{N}| \geq 100$

$\|\mathcal{N}\|$=100						$\|\mathcal{N}\|$=200						$\|\mathcal{N}\|$=300					
$\|\mathcal{K}\|$=11			$\|\mathcal{K}\|$=14			$\|\mathcal{K}\|$=21			$\|\mathcal{K}\|$=26			$\|\mathcal{K}\|$=26			$\|\mathcal{K}\|$=31		
Min.	Avg.	p	Min.	Avg.	p	Min.	Avg.	p	Min.	Avg.	p	Min.	Avg.	p	Min.	Avg.	p
15783	16814.00	1.00	14639	15105.86	1.00	26141	26255.81	0.60	22988	24329.73	0.80	25643	27245.88	0.20	36863	36863.00	0.10
18195	18978.72	1.00	16983	17048.21	1.00	22561	25537.02	0.80	23208	24252.59	0.60	30923	32754.66	0.30	34802	36051.10	0.20
19359	20149.57	1.00	17276	18669.40	1.00	24482	24612.82	0.90	39400	40413.34	0.40	24157	24157.00	0.10	27048	27048.00	0.10
19003	20045.09	1.00	18595	18735.76	1.00	30150	30781.05	0.50	30704	31284.10	0.40	32840	32840.00	0.10	30941	32125.67	0.20
17209	17899.14	1.00	15545	15886.75	1.00	27472	28012.10	0.80	24249	39747.25	0.60	36033	36884.34	0.30	31384	32490.54	0.20
15847	16486.18	1.00	16084	16691.19	1.00	32076	32985.66	0.50	28532	28532.00	0.10	25955	25955.00	0.20	29687	31132.75	0.30
16741	17412.80	1.00	15463	15955.84	1.00	21794	22412.09	0.70	27553	27553.00	0.10	31842	31842.00	0.10	29168	29168.00	0.10
16921	17623.65	1.00	16213	16552.50	1.00	29418	30115.54	0.60	24935	26035.65	0.50	29085	29085.00	0.10	26753	26753.00	0.10
19057	20115.26	1.00	18327	18560.71	1.00	26767	27123.43	0.50	22357	23157.39	0.60	33093	34511.89	0.20	34658	34658.00	0.10
18097	18792.69	1.00	16897	18127.22	1.00	27668	27979.52	0.80	-	-	0.00	25385	26650.02	0.30	34897	36265.08	0.20

Table 5.13: CPU times for 2E–BRPSD matheuristic strategies (s)

$\|\mathcal{N}\|$	Constructive	SPP-Math	GTSP-Cost	GTSP-Id
20	7.40	15.44	27.55	9.839
30	18.80	74.27	90.97	27.761
40	80.51	215.30	159.50	53.233
50	108.12	1936.40	685.32	478.181
60	98.45	1974.61	892.67	559.816
100	-	-	3149.20	-
200	-	-	3600.00	-
300	-	-	3600.00	-

Table 5.14: CPU time distribution for hybrid constructive algorithm components

$\|\mathcal{N}\|$	Constructive phase	VND	Split	Local search	Checking feasibilty
20	0.04%	83.60%	5.00%	4.45%	6.90%
30	0.06%	86.91%	2.23%	2.18%	8.51%
40	0.04%	84.36%	1.30%	1.22%	12.95%
50	0.06%	81.19%	1.15%	1.94%	15.45%
60	0.12%	85.27%	0.67%	0.70%	12.13%

Table 5.15: CPU time distribution for SPP-based algorithm components

$\|\mathcal{N}\|$	Constructive phase	VND	Split	Local search	SPP MILP
20	0.02%	44.43%	0.83%	0.72%	54.00%
30	0.06%	37.61%	0.39%	0.33%	61.60%
40	0.01%	23.20%	0.17%	0.13%	76.50%
50	0.00%	8.81%	0.06%	0.05%	91.08%
60	0.07%	10.50%	0.07%	0.03%	89.33%

and GTSP-Id), SPP and GTSP MILPs take at least 52.25% of the total computation time (instances with 200 stations).

Comments on GTSP-based matheuristic implementation

As described in Section 5.6, GTSP-based algorithms (GTSP-Cost and GTSP-Id) call SPP and GTSP MILPs in an iterative way. With the aim to improve performance when solving MILPs within the GTSP-based strategies, some solver parameters and functions were previously set to get results and performance presented before. A brief description on these parameters and functions setting follows:

Table 5.16: CPU time distribution for GTSP-based algorithm components

$\mathcal{N}$	Constructive phase	VND	Split	Local search	SPP MILP	GTSP MILP
20	5.49%	33.54%	0.62%	0.53%	39.06%	20.76%
30	4.05%	30.09%	0.31%	0.26%	36.84%	28.46%
40	2.45%	30.21%	0.23%	0.20%	45.58%	21.34%
50	2.01%	22.43%	0.13%	0.12%	49.90%	25.41%
60	2.85%	19.52%	0.10%	0.09%	53.22%	24.22%
100	4.03%	16.47%	0.05%	0.05%	54.38%	25.02%
200	15.10%	31.93%	0.08%	0.13%	36.62%	16.13%
300	19.40%	34.09%	0.07%	0.15%	33.13%	13.16%

- **Maximum computation times for GTSP MILP**: computation times for GTSP MILP may increase significantly if the number of stations and vehicles for secondary routes also increases. After several experiments, a suitable value for the GTSP maximum computation time in a particular iteration, is set as a function of times required in previous iterations. For ten first iterations, the maximum computation time was set to 1800 seconds. However, once ten iterations are completed, a new maximum CPU time is computed as the 90th percentile on those previous times. This new value is set for the next ten iterations. This updated is performed until the solution strategies finish the search.

- **GTSP callback**: As described in Section 5.6 GTSP-based algorithms stop when traveling cost over secondary routes (λ) plus GTSP objective function value exceeds the best solution found for the 2E–BRP. This stop criterion may be checked once GTSP MILP is solved. However, to anticipate whether the stop criterion could be met, lower bound for GTSP MILP is useful. To do so, a callback procedure was coded within the GTSP MILP. This callback works on branch and bound procedure and checks for lower bound updates. Thus, if GTSP lower bound plus traveling cost for secondary routes is greater or equal than 2E–BRPSD best solution found so far, GTSP optimization process stops.

- **Solver parameter tuning**: Gurobi provides a powerful tool able to search for an accurate value for several optimization parameters (e.g., cut strategies, feasibility heuristics, branch direction). This tool called *parameter tuning* automatically tries a number of different parameter settings. Once this settings are tested, the solver retrieves the best ones that it finds. This parameter tuning process was applied to SPP and GTSP MILPs within GTSP-based algorithms.

5.7.4 Results on EnCicla instances

After proposed MILP for the 2E–BRPSD and solution strategies were tested on benchmark problems, EnCicla instances were also solved. Table 5.17 shows the results on MILP in equations (5.1)– (5.21) for EnCicla instances. After one hour of computation time, the solver did not retrieve a certified optimal solution for any of the instances. On average, the optimizer reports a gap of 49.34% providing at least one feasible solution for 29 out of 35 instances.

Since solution quality for GTSP-Cost outperforms results retrieved from other proposed strategies when benchmark data sets are solved, only GTSP-Cost is used to solve EnCicla instances. Table 5.18 summarizes results on EnCicla instances solved via GTSP-Cost algorithm. Columns Min. f and Avg. f show the minimum and average value retrieved for total traveling cost after ten runs of the algorithm. It is worth to mention that for all instances, GTSP-Cost finds at least one solution after each run, thus the probability of finding a solution over ten runs is one for all instances.

In columns Improvement Min. UB and Improvement Avg. UB, obtained minimum and average improvements over MILP upper bound are computed, respectively. These improvements can be evaluated as:

$$Improvement\ Min.\ UB(\%) = \frac{\text{UB} - \text{Min. } f}{\text{UB}} \cdot 100 \tag{5.78}$$

$$Improvement\ Avg.\ UB(\%) = \frac{\text{UB} - \text{Avg. } f}{\text{UB}} \cdot 100 \tag{5.79}$$

Despite MILP finds better solutions than those reported via GTSP-Cost for nine of the instances, the matheuristic algorithm improves minimum values for total traveling cost, on average (i.e., the average for Improvement Min. UB is 6.96%). For the six instances with no feasible after one hour of MILP computation, GTSP-Cost is able to retrieve at least one solution in each one of the ten runs (i.e., instances 6, 15, 19, 29, 31 and 32).

Columns gap Min. LB and gap Avg. LB show minimum and average gaps over MILP lower bound. These values are computed as:

$$gap\ Min.\ LB(\%) = \frac{\text{Min. } f - \text{LB}}{\text{Min. } f} \cdot 100 \tag{5.80}$$

$$gap\ Avg.\ LB(\%) = \frac{\text{Avg. } f - \text{LB}}{\text{Avg. } f} \cdot 100 \tag{5.81}$$

Since average values for columns gap Min. LB and gap Avg. LB are lower than average gap computed in Table 5.17, values on gap Min. LB and gap Avg. prove that GTSP-Cost finds better solutions than MILP, on average. Finally, it is worth to mention that computation times required for

Table 5.17: MILP results on EnCicla instances

Instance	UB	LB	gap
1	4173	2552.05	38.82%
2	4332	2568.18	40.70%
3	4400	2556.58	41.89%
4	3643	2522.68	30.74%
5	4116	2647.20	35.67%
6	-	2864.28	100.00%
7	4739	2669.96	43.66%
8	3540	2532.21	28.45%
9	3878	2536.61	34.58%
10	4627	2582.75	44.18%
11	3965	2590.11	34.65%
12	4218	2590.23	38.57%
13	4067	2540.98	37.52%
14	5285	2639.31	50.05%
15	-	2481.00	100.00%
16	4622	2575.06	44.27%
17	4592	2559.56	44.25%
18	4926	2628.49	46.63%
19	-	2539.21	100.00%
20	4203	2417.50	42.47%
21	4284	2558.47	40.27%
22	4465	2621.28	41.28%
23	3921	2610.72	33.41%
24	3610	2539.37	29.64%
25	4381	2659.37	39.28%
26	4220	2494.00	40.90%
27	3684	2570.74	30.21%
28	4719	2409.99	48.93%
29	-	2526.62	100.00%
30	3875	2452.49	36.70%
31	-	2520.14	100.00%
32	-	2496.24	100.00%
33	3757	2561.54	31.81%
34	3831	2474.14	35.40%
35	4365	2531.14	41.99%
Average			49.34%

GTSP-Cost to find reported solutions are significantly less than those used by the solver. While solving each instance vian MILP takes 3600 seconds, GTSP-Cost requires 505.42 seconds, on average.

Table 5.18: GTSP-based matheuristic results on EnCicla instances

Instance	Min. f	Avg. f	Improvement Min. UB	Improvement Avg. UB	gap Min. LB	gap Avg. LB	Time (s)
1	3860	3860.00	7.50%	7.50%	33.88%	33.88%	121.63
2	4262	4298.90	1.62%	0.76%	39.74%	40.26%	374.94
3	4184	4360.90	4.91%	0.89%	38.90%	41.37%	450.36
4	3867	3967.80	-6.15%	-8.92%	34.76%	36.42%	398.61
5	4156	4324.30	-0.97%	-5.06%	36.30%	38.78%	721.98
6	5109	5109.00	-	-	43.94%	43.94%	197.62
7	4241	4568.50	10.51%	3.60%	37.04%	41.56%	256.98
8	3972	4017.00	-12.20%	-13.47%	36.25%	36.96%	520.35
9	3716	3827.20	4.18%	1.31%	31.74%	33.72%	354.27
10	3955	4174.50	14.52%	9.78%	34.70%	38.13%	668.94
11	3912	4094.50	1.34%	-3.27%	33.79%	36.74%	495.60
12	3765	3873.80	10.74%	8.16%	31.20%	33.13%	252.87
13	3708	3774.50	8.83%	7.19%	31.47%	32.68%	593.79
14	4181	4452.80	20.89%	15.75%	36.87%	40.73%	391.73
15	3472	3553.70	-	-	28.54%	30.19%	828.49
16	3669	4103.90	20.62%	11.21%	29.82%	37.25%	496.70
17	4230	4520.60	7.88%	1.55%	39.49%	43.38%	435.80
18	3625	3750.30	26.41%	23.87%	27.49%	29.91%	234.38
19	4169	4307.30	-	-	39.09%	41.05%	507.09
20	3346	3537.00	20.39%	15.85%	27.75%	31.65%	884.74
21	3902	4058.70	8.92%	5.26%	34.43%	36.96%	479.29
22	4034	4298.90	9.65%	3.72%	35.02%	39.02%	415.15
23	4328	4328.00	-10.38%	-10.38%	39.68%	39.68%	761.09
24	3705	3850.00	-2.63%	-6.65%	31.46%	34.04%	643.90
25	4346	4387.50	0.80%	-0.15%	38.81%	39.39%	737.17
26	3585	3911.50	15.05%	7.31%	30.43%	36.24%	689.68
27	3801	4160.10	-3.18%	-12.92%	32.37%	38.20%	267.50
29	4027	4027.70	-	-	37.26%	37.27%	364.81
30	3453	3571.80	10.89%	7.82%	28.98%	31.34%	501.76
31	3934	4179.90	-	-	35.94%	39.71%	570.06
32	4303	4535.00	-	-	41.99%	44.96%	380.26
33	3862	3995.40	-2.79%	-6.35%	33.67%	35.89%	530.76
34	3385	3508.30	11.64%	8.42%	26.91%	29.48%	775.74
35	3669	4329.20	15.95%	0.82%	31.01%	41.53%	880.39
Average			6.96%	2.63%	34.43%	37.22%	505.42

5.8 Concluding remarks

In this chapter a new bicycle repositioning problem is defined: the two-echelon bicycle repositioning problem with split demand (2E–BRPSD). In the 2E–BRPSD the repositioning operation is performed through a set of vehicles, each one devoted to serve a subset of BSS stations (secondary routes). Moreover one vehicle visits one of the stations within each secondary route (i.e., satellite depots) supporting rebalancing operation for each secondary route. In those connection points or depots, split demand operations are allowed.

This chapter proposed an MILP for the 2E–BRP as well as four solution strategies. Three of them are matheuristic algorithms where metaheuristic procedures and mathematical optimization models are combined. While solving MILP allows to hardly finds optimal solutions for instances with 20 stations, all metaheuristic algorithms are able to find solutions for instances with up to 60 stations and one of them solves instances up to 300 stations.

A set of real-world instances with 52 stations is also tested in MILP and the outperforming matheuristic algorithm. These instances are based on EnCicla operation, the BSS in Medellín, Colombia. Results show that matheuristic algorithm is able to find better solutions than those retrieved when solving 2E–BRP MILP via commercial solver.

Conferences and publications

Results on matheuristic algorithms for the 2E–BRPSD will be presented in the XXI Latin Ibero-American Conference on Operations Research (CLAIO) 2022:

- Palacio J.D., Rivera J.C. The two-echelon bicycle repositioning problem with split demand (2E–BRPSD). 21st Latin Ibero-American Conference on Operations Research (CLAIO). Buenos Aires, Argentina. 2022.

Chapter 6

General conclusions and future research directions

Vehicle routing problems have been widely studied in operational research field. Their computational complexity and their practical relevance in many logistic contexts to support decision making processes make VRPs an interesting research problem. Optimization techniques allow to study the vehicle routing problem and a very large number of variants that arise as long as private and public entities require for goods and services distribution.

Due to population growth and mobility issues in urban zones in many countries, bike sharing systems (BSSs) arise as a way to promote sustainable transportation. Moreover, to design and manage an efficient BSS become relevant to improve mobility, welfare and health for citizens.

In this thesis three families of vehicle routing problems arising in BSSs context have been studied. These problems are: the one-commodity pickup and delivery traveling salesman problem (1–PDTSP), the one-commodity pickup and delivery vehicle routing problem with route length constraints (1–PDVRPLC) and also, a new problem is defined: the two-echelon bicycle repositioning problem with split demand (2E–BRPSD). In these problems, one or several capacitated vehicles must visit the BSS stations and pickup or deliver bicycles to better fit expected demand at stations. Therefore, pickup and delivery is a key feature in problems studied in this thesis.

The 1–PDTSP models the case in which a single vehicle visits all the locations picking up and delivering commodity units. The motivation to study this problem was mainly based not only on its computational complexity but on the fact that 1–PDTSP can be seen as a sub problem for more complex applications like 1–PDVRPLC and 2E–BRPSD. Moreover, in BSSs contexts, the 1–PDTSP can be applied in small and medium size systems since repositioning for large BSSs tends

to be intractable if a single vehicle is available.

For the 1–PDTSP a mixed-integer linear programming model (MILP) and a metaheuristic solution strategy have been developed. The metaheuristic algorithm is based on a multi-start evolutionary local search (MS-ELS) framework in which seven local search operators are embedded in a variable neighborhood descent (VND) algorithm. The MS-ELS framework allows to adapt the solution strategy to a multi-start iterative local search (MS-ILS) and a greedy randomized adaptive search procedure (GRASP) by simply setting different values for MS-ELS algorithm parameters.

To test the performance of 1–PDTSP MILP and the metaheuristic approach, two different set of instances were solved. Firstly, a set of 1–PDTSP well-known benchmark instances with up to 500 nodes and a fixed value for vehicle capacity that leads to the hardest configuration to solve. Secondly, a set of instances based on the repositioning operation of 52 stations of EnCicla, the BSS in Medellín (Colombia) is also solved. Regarding benchmark instances, 1–PDTSP MILP is able to optimally solve instances with up to 40 nodes. For larger instances up to 60 nodes, upper bounds are provided. On the other hand, MS–ELS was tested with instances with up to 500 nodes and the obtained solutions outperform two solution strategies based on GRASP/VND and genetic algorithm previously reported in the literature. For instances with 50 and 60 nodes it was also possible to prove the optimality of solutions reported by MS-ELS since lower bounds obtained via commercial solver are available after solving MILP. For EnCicla based instances, similar results are obtained since MS-ELS solutions and lower bound values allow to certify optimality for 28 out of 35 instances. The metaheuristic algorithm solves instances in a competitive time (less than 20 seconds on average) which is a suitable computation time since static rebalancing decisions are made up once per day (mostly at night when system is not available for users).

A variant of the 1–PDTSP, the 1–PDTSP with split demand (SD1PDTSP) have been also studied in this thesis. For the SD1PDTSP, a MILP able to solve to optimality instances up to 40 locations is proposed. The results obtained when solving the mathematical model via commercial solver outperforms those reported in the literature obtained via a branch-and-cut. In a similar way, MILP requires less computational time to retrieve solutions when it is compared with the B&C algorithm. For the SD1PDTSP, experiments with different values for the vehicle capacity are presented as well as benefits of split delivery and storage when such capacity is tight.

From a practical perspective and large systems, mathematical models and solution strategies for the 1–PDTSP as the MS-ELS, can be easily integrated to clustering strategies in order to find several routes (one per cluster) if multiple vehicles are required. Additionally, considering relevant information from BSS (e.g. service times at stations) is straightforward in 1–PDTSP formulations and solution strategies.

The 1–PDVRP extends the 1–PDTSP when multiple vehicles are available to pickup and deliver commodity units. This thesis has studied the 1–PDVRP if route length constraints are imposed. For rebalancing operations in BSS and other practical contexts, route length constraints allow to reduce workload unbalances among vehicle operators and fairly assign tasks and operation times for commodity distribution.

For the 1–PDVRPLC, a MILP have been proposed as well as a set of matheuristic algorithms based on large neighborhood search procedures. For route length constraints, a maximum route duration in terms of the number of available vehicles and an upper bound based on 1–PDTSP solution is proposed. The mathematical model is solved via commercial solver and instances with up to 40 nodes and different target values for balancing routes were tested. The experiments allowed to provide some insights about how the number of nodes, number of vehicles and vehicle capacity affect imbalance throughout routes total traveling costs.

The proposed matheuristic algorithms are mainly LNS procedures in which destroy and/or repair methods are replaced by mathematical optimization models. The first algorithm is a LNS embedded on a multi-start iterative framework. For this algorithm, a single MILP takes destroy and repair decisions simultaneously. A second solution strategy is a matheuristic with an adaptive control to decide how to partially destroy routes and also, for whether is better to reduce the number of routes in the solution. Similar to the multi-start matheuristic, the adaptive algorithm calls a MILP to repair solutions. For the adaptive procedure, a variant based on an enumeration algorithm have been proposed. The enumeration algorithm replaces repair MILP and it is also able to search for better solution on repair process when it is compared to MILP since enumeration algorithm explores a larger region of solutions space. This difference on explored spaces is due to dominance rules designed to efficiently discard solutions with a promising larger value for total traveling costs while potential promising moves are limited on MILP.

Following the proposed strategy to compute maximum route lengths, several values for this parameter were considered to solve instances with up to 500 nodes. Results on solution strategies show that the enumeration algorithm embedded on adaptive LNS procedure outperforms the multi-start strategy as well as the adaptive algorithm with MILP as repair method. In a similar way, the proposed strategies were compared with a genetic algorithm reported in the literature for 1–PDVRPLC. Enumeration algorithm within the adaptive LNS framework outperforms solution quality of genetic algorithm for 90% of all tested instances.

In this thesis, a new generalization of bicycle repositioning problem, the 2E–BRPSD is also studied. The 2E–BRPSD involves pickup and delivery operations in a two-echelon configuration where also split demand is allowed. This problem have been studied as a first way to consider

cooperative operations in bike repositioning problems. Despite the 2E–BRPSD does not consider localization decisions explicitly, a set of satellite depots must be selected from the pool of available stations. Indeed, a collaborative environment for 2E–BRPSD takes place in satellite depots where one vehicle can support and/or complete other vehicle requirements.

Chapter 5 provides three main contributions based on 2E–BRPSD. Firstly, the problem is defined since there is not evidence of prior work on 2E–BRPSD in the literature. Secondly, a first mathematical linear model is proposed to describe the 2E–BRPSD. Lastly, four solution strategies are proposed. Three of the proposed strategies are mainly hybrid algorithms that combine metaheuristic algorithms with mixed-integer programming models.

From the pool of proposed solution strategies for 2E–BRPSD, a matheuristic algorithm outperforms the other three proposed strategies. The best performance algorithm heuristically creates a pool of routes for the second echelon and then, iteratively searches feasible combinations of first and second level routes via generalized traveling salesman problem (GTSP) and set partitioning problem (SPP) models, respectively.

While the other proposed strategies hardly find feasible solutions for medium size instances, the outperforming matheuristic retrieve solutions with up to 300 stations within a maximum computation time of 3600 seconds. Inspired on administrative division in Medellín (Colombia), 2E–BRPSD was also solved for EnCicla instances. For this set of instances, matheuristic algorithm outperforms results retrieved when solving 2E–BRPSD MILP via commercial solver.

In spite of the good results achieved on the 1–PDTSP, 1–PDVRPLC and 2E–BRPSD, there are several improvement opportunities for some of the solution strategies presented in this thesis. For the 1–PDVRPLC, different metaheuristic and matheuristic algorithms could be explored. Despite the theoretical relevance on designing mathematical optimization models to support search process within heuristic algorithms, computation effort could be improved. Inspired on the enumeration algorithm developed, exact techniques as dynamic programming could be explored.

In a similar way, since mathematical models within proposed solution strategies for the 2E–BRPSD require a high computation effort, more involved metaheuristic algorithms could be designed.

Future research directions are also based on the study of additional features for static bike repositioning problems and one-commodity vehicle routing problems. The objective behind these research paths is to close gaps between theoretical developments and required facets in real-world BSS scenarios.

Green-based objective functions for bike repositioning problems would be an interesting study path. CO_2 emissions is still a relevant component for vehicle routing problems and has been

emerged as a key metric for decision makers in public and private transportation systems. In line with this motivation on green vehicle routing problems, it could be interesting to explore mixed fleet of internal combustion engine vehicles and battery electric vehicles.

Finally, it could be interesting to explore research paths on integrated approaches for static and dynamic repositioning problems. Decisions for static repositioning affects dynamic routing during BBSs operation. Therefore, to design methods that integrates tactical and operational decisions within rebalancing contexts are worth to be explored. Indeed, a collaborative operation as the one modeled in 2E–BRPSD could mark an initial point to explore how static rebalancing support daily dynamic operation on several zones. Similarly, a validation on the 2E–BRPSD scheme as a strategy to deal with the dynamic version of the reposition problem could be performed; since it is possible to use several vehicles to perform short repositioning routes (with limited length), this strategy remain useful in an intra day operation.

Bibliography

Aksen, D., Kaya, O., Salman, F. S., and Tüncel, Ö. (2014). An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem. *European Journal of Operational Research*, 239(2):413–426.

Alhindi, A., Alsaidi, A., Alasmary, W., and Alsabaan, M. (2020). Vehicle routing optimization for surplus food in nonprofit organizations. *International Journal of Advanced Computer Science and Applications*, 11(3):680–685.

Alinaghian, M. and Shokouhi, N. (2018). Multi-depot multi-compartment vehicle routing problem, solved by a hybrid adaptive large neighborhood search. *Omega*, 76:85–99.

Altay, N. and Green III, W. G. (2006). OR/MS research in disaster operations management. *European Journal of Operational Research*, 175(1):475–493.

Alvarez-Valdes, R., Belenguer, J. M., Benavent, E., Bermudez, J. D., Muñoz, F., Vercher, E., and Verdejo, F. (2016). Optimizing the level of service quality of a bike-sharing system. *Omega*, 62:163–175.

Archetti, C., Bianchessi, N., and Speranza, M. G. (2014). Branch-and-cut algorithms for the split delivery vehicle routing problem. *European Journal of Operational Research*, 238(3):685–698.

Archetti, C. and Speranza, M. G. (2012). Vehicle routing problems with split deliveries. *International Transactions in Operational Research*, 19(1-2):3–22.

Babin, G., Deneault, S., and Laporte, G. (2007). Improvements to the or-opt heuristic for the symmetric travelling salesman problem. *Journal of the Operational Research Society*, 58(3):402–407.

Balcik, B., Iravani, S. M., and Smilowitz, K. (2010). A review of equity in nonprofit and public sector: a vehicle routing perspective. *Wiley Encyclopedia of Operations Research and Management Science*.

Bartholdi III, J. J., Platzman, L. K., Collins, R. L., and Warden III, W. H. (1983). A minimal technology routing system for meals on wheels. *Interfaces*, 13(3):1–8.

Belenguer, J.-M., Benavent, E., Labadi, N., Prins, C., and Reghioui, M. (2010). Split-delivery capacitated arc-routing problem: Lower bound and metaheuristic. *Transportation Science*, 44(2):206–220.

Belgin, O., Karaoglan, I., and Altiparmak, F. (2018). Two-echelon vehicle routing problem with simultaneous pickup and delivery: Mathematical model and heuristic approach. *Computers & Industrial Engineering*, 115:1–16.

Braekers, K. and Kovacs, A. A. (2016). A multi-period dial-a-ride problem with driver consistency. *Transportation Research Part B: Methodological*, 94:355–377.

Breunig, U., Baldacci, R., Hartl, R. F., and Vidal, T. (2019). The electric two-echelon vehicle routing problem. *Computers & Operations Research*, 103:198–210.

Breunig, U., Schmid, V., Hartl, R. F., and Vidal, T. (2016). A large neighbourhood based heuristic for two-echelon routing problems. *Computers & Operations Research*, 76:208–225.

Bulhões, T., Subramanian, A., Erdoğan, G., and Laporte, G. (2018). The static bike relocation problem with multiple vehicles and visits. *European Journal of Operational Research*, 264(2):508–523.

Caggiani, L. and Ottomanelli, M. (2013). A dynamic simulation based model for optimal fleet repositioning in bike-sharing systems. *Procedia-Social and Behavioral Sciences*, 87:203–210.

Caramia, M. and Guerriero, F. (2010). A milk collection problem with incompatibility constraints. *Interfaces*, 40(2):130–143.

Castañeda, C. and Villegas, J. G. (2017). Analyzing the response to traffic accidents in Medellín, Colombia, with facility location models. *IATSS Research*, 41(1):47–56.

Chemla, D., Meunier, F., and Wolfler Calvo, R. (2013). Bike sharing systems: Solving the static rebalancing problem. *Discrete Optimization*, 10(2):120–146.

Chen, H.-K., Chou, H.-W., Hsueh, C.-F., and Yu, Y.-J. (2015). The paired many-to-many pickup and delivery problem: an application. *Top*, 23(1):220–243.

Cheng, Y., Wang, J., and Wang, Y. (2021). A user-based bike rebalancing strategy for free-floating bike sharing systems: A bidding model. *Transportation Research Part E: Logistics and Transportation Review*, 154:102438.

Chung, H., Freund, D., and Shmoys, D. B. (2018). Bike angels: An analysis of citi bike's incentive program. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, pages 1–9.

Contardo, C., Morency, C., and Rousseau, L.-M. (2012). *Balancing a dynamic public bike-sharing system*, volume 4. Cirrelt Montreal, Canada.

Cordeau, J.-F. and Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594.

Cruz, F., Subramanian, A., Bruck, B. P., and Iori, M. (2017). A heuristic algorithm for a single vehicle static bike sharing rebalancing problem. *Computers & Operations Research*, 79:19–33.

Cuda, R., Guastaroba, G., and Speranza, M. G. (2015). A survey on two-echelon routing problems. *Computers & Operations Research*, 55:185–199.

Datner, S., Raviv, T., Tzur, M., and Chemla, D. (2019). Setting inventory levels in a bike sharing network. *Transportation Science*, 53(1):62–76.

De la Torre, L. E., Dolinskaya, I. S., and Smilowitz, K. R. (2012). Disaster relief routing: Integrating research and practice. *Socio-Economic Planning Sciences*, 46(1):88–97.

Dell'Amico, M., Hadjicostantinou, E., Iori, M., and Novellani, S. (2014). The bike sharing rebalancing problem: Mathematical formulations and benchmark instances. *Omega*, 45:7–19.

Dell'Amico, M., Iori, M., Novellani, S., and Stützle, T. (2016). A destroy and repair algorithm for the bike sharing rebalancing problem. *Computers & Operations Research*, 71:149 – 162.

Demir, E., Bektaş, T., and Laporte, G. (2012). An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, 223(2):346–359.

Detti, P., Papalini, F., and de Lara, G. Z. M. (2017). A multi-depot dial-a-ride problem with heterogeneous vehicles and compatibility constraints in healthcare. *Omega*, 70:1–14.

Doerner, K. F. and Salazar-González, J.-J. (2014). Chapter 7: Pickup-and-delivery problems for people transportation. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 193–212. SIAM.

Drexl, M. (2012). Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science*, 46(3):297–316.

Du, M., Cheng, L., Li, X., and Tang, F. (2020). Static rebalancing optimization with considering the collection of malfunctioning bikes in free-floating bike sharing system. *Transportation Research Part E: Logistics and Transportation Review*, 141:102012.

Duan, Y. and Wu, J. (2022). Spatial-temporal inventory rebalancing for bike sharing systems with worker recruitment. *IEEE Transactions on Mobile Computing*, 21(3):1081 – 1095.

Duhamel, C., Lacomme, P., Quilliot, A., and Toussaint, H. (2011). A multi-start evolutionary local search for the two-dimensional loading capacitated vehicle routing problem. *Computers & Operations Research*, 38(3):617–640.

Eisenhandler, O. and Tzur, M. (2019). The humanitarian pickup and distribution problem. *Operations Research*, 67(1):10–32.

Erdoğan, G., Battarra, M., and Calvo, R. W. (2015). An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. *European Journal of Operational Research*, 245(3):667–679.

Espegren, H. M., Kristianslund, J., Andersson, H., and Fagerholt, K. (2016). The static bicycle repositioning problem-literature survey and new formulation. In *International Conference on Computational Logistics*, pages 337–351. Springer.

Fallahtafti, A., Ardjmand, E., Young Ii, W. A., and Weckman, G. R. (2021). A multi-objective two-echelon location-routing problem for cash logistics: A metaheuristic approach. *Applied Soft Computing*, 111:107685.

Fathollahi-Fard, A. M., Hajiaghaei-Keshteli, M., and Mirjalili, S. (2020). A set of efficient heuristics for a home healthcare problem. *Neural Computing and Applications*, 32(10):6185–6205.

Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133.

Fikar, C. and Hirsch, P. (2015). A matheuristic for routing real-world home service transport systems facilitating walking. *Journal of Cleaner Production*, 105:300–310.

Forma, I. A., Raviv, T., and Tzur, M. (2015). A 3-step math heuristic for the static repositioning problem in bike-sharing systems. *Transportation Research Part B: Methodological*, 71:230–247.

Frade, I. and Ribeiro, A. (2015). Bike-sharing stations: A maximal covering location approach. *Transportation Research Part A: Policy and Practice*, 82:216–227.

Fricker, C. and Gast, N. (2016). Incentives and redistribution in homogeneous bike-sharing systems with stations of finite capacity. *Euro Journal on Transportation and Logistics*, 5(3):261–291.

Fu, C., Zhu, N., Ma, S., and Liu, R. (2022). A two-stage robust approach to integrated station location and rebalancing vehicle service design in bike-sharing systems. *European Journal of Operational Research*, 298(3):915–938.

Garcia-Gutierrez, J., Romero-Torres, J., and Gaytan-Iniestra, J. (2014). Dimensioning of a bike sharing system (BSS): a study case in Nezahualcoyotl, Mexico. *Procedia-Social and Behavioral Sciences*, 162:253–262.

García-Palomares, J. C., Gutiérrez, J., and Latorre, M. (2012). Optimizing the location of stations in bike-sharing programs: A gis approach. *Applied Geography*, 35(1-2):235–246.

Grangier, P., Gendreau, M., Lehuédé, F., and Rousseau, L.-M. (2016). An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research*, 254(1):80–91.

Grimault, A., Bostel, N., and Lehuédé, F. (2017). An adaptive large neighborhood search for the full truckload pickup and delivery problem with resource synchronization. *Computers & Operations Research*, 88:1–14.

Gutiérrez, E. V. and Vidal, C. J. (2013). Home health care logistics management: Framework and research perspectives. *International Journal of Industrial Engineering and Management*, 4(3):173–182.

Ham, A. (2021). Dial-a-ride problem: mixed integer programming revisited and constraint programming proposed. *Engineering Optimization*, pages 1–14.

Han, L., Luong, B. T., and Ukkusuri, S. (2016). An algorithm for the one commodity pickup and delivery traveling salesman problem with restricted depot. *Networks and Spatial Economics*, 16(3):743–768.

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467.

Hansen, P., Mladenović, N., Todosijević, R., and Hanafi, S. (2017). Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454.

Hemmelmayr, V. C., Cordeau, J.-F., and Crainic, T. G. (2012). An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & Operations Research*, 39(12):3215–3228.

Hernández-Pérez, H., Landete, M., and Rodriguez-Martin, I. (2021). The single-vehicle two-echelon one-commodity pickup and delivery problem. *Computers & Operations Research*, 127:105152.

Hernández-Pérez, H., Rodríguez-Martín, I., and Salazar-González, J. J. (2009). A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research*, 36(5):1639–1645.

Hernández-Pérez, H., Rodríguez-Martín, I., and Salazar-González, J.-J. (2016). A hybrid heuristic approach for the multi-commodity pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 251(1):44–52.

Hernández-Pérez, H. and Salazar-González, J.-J. (2004a). A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126–139.

Hernández-Pérez, H. and Salazar-González, J.-J. (2004b). Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, 38(2):245–255.

Hernández-Pérez, H. and Salazar-González, J.-J. (2007). The one-commodity pickup-and-delivery traveling salesman problem: Inequalities and algorithms. *Networks: An International Journal*, 50(4):258–272.

Hernández-Pérez, H. and Salazar-González, J.-J. (2009). The multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *European Journal of Operational Research*, 196(3):987–995.

Hernández-Pérez, H. and Salazar-González, J.-J. (2014). The multi-commodity pickup-and-delivery traveling salesman problem. *Networks*, 63(1):46–59.

Hernández-Pérez, H. and Salazar-González, J.-J. (2022). A branch-and-cut algorithm for the split-demand one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research*, 297(2):467–483.

Hernández-Pérez, H., Salazar-González, J. J., and Santos-Hernández, B. (2018). Heuristic algorithm for the split-demand one-commodity pickup-and-delivery travelling salesman problem. *Computers & Operations Research*, 97:1–17.

Ho, S. C. and Szeto, W. (2014). Solving a static repositioning problem in bike-sharing systems using iterated tabu search. *Transportation Research Part E: Logistics and Transportation Review*, 69:180–198.

Ho, S. C. and Szeto, W. Y. (2017). A hybrid large neighborhood search for the static multi-vehicle bike-repositioning problem. *Transportation Research Part B: Methodological*, 95:340–363.

Ho, S. C., Szeto, W. Y., Kuo, Y.-H., Leung, J. M., Petering, M., and Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, 111:395–421.

IFORS (2022). International Federation of Operational Research Societies. https://www.https://www.ifors.org/what-is-or/. Accessed: 2022-07-06.

Jia, Y., Zeng, W., Xing, Y., Yang, D., and Li, J. (2021). The bike-sharing rebalancing problem considering multi-energy mixed fleets and traffic restrictions. *Sustainability*, 13(1):270.

Jie, W., Yang, J., Zhang, M., and Huang, Y. (2019). The two-echelon capacitated electric vehicle routing problem with battery swapping stations: Formulation and efficient methodology. *European Journal of Operational Research*, 272(3):879–904.

Jotshi, A., Gong, Q., and Batta, R. (2009). Dispatching and routing of emergency vehicles in disaster mitigation using data fusion. *Socio-Economic Planning Sciences*, 43(1):1–24.

Kadri, A. A., Kacem, I., and Labadi, K. (2016). A branch-and-bound algorithm for solving the static rebalancing problem in bicycle-sharing systems. *Computers and Industrial Engineering*, 95:41–52.

Kloimüllner, C., Papazek, P., Hu, B., and Raidl, G. R. (2014). Balancing bicycle sharing systems: an approach for the dynamic case. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 73–84. Springer.

Koç, Ç. (2016). A unified-adaptive large neighborhood search metaheuristic for periodic location-routing problems. *Transportation Research Part C: Emerging Technologies*, 68:265–284.

Kovacs, A. A., Golden, B. L., Hartl, R. F., and Parragh, S. N. (2014). Vehicle routing problems in which consistency considerations are important: A survey. *Networks*, 64(3):192–213.

Kumar, A. A., Kang, J. E., Kwon, C., and Nikolaev, A. (2016). Inferring origin-destination pairs and utility-based travel preferences of shared mobility system users in a multi-modal environment. *Transportation Research Part B: Methodological*, 91:270–291.

Laporte, G., Meunier, F., and Calvo, R. W. (2015). Shared mobility systems. *4OR*, 13(4):341–360.

Larsen, J., Patterson, Z., and El-Geneidy, A. (2013). Build it. but where? the use of geographic information systems in identifying locations for new cycling infrastructure. *International Journal of Sustainable Transportation*, 7(4):299–317.

Lee, E., Son, B., and Han, Y. (2020). Optimal relocation strategy for public bike system with selective pick-up and delivery. *Transportation Research Record*, 2674(4):325–336.

Legros, B. (2019). Dynamic repositioning strategy in a bike-sharing system; how to prioritize and how to rebalance a bike station. *European Journal of Operational Research*, 272(2):740–753.

Li, H., Wang, H., Chen, J., and Bai, M. (2020). Two-echelon vehicle routing problem with time windows and mobile satellites. *Transportation Research Part B: Methodological*, 138:179–201.

Li, Y., Chen, H., and Prins, C. (2016). Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests. *European Journal of Operational Research*, 252(1):27–38.

Lim, A., Zhang, Z., and Qin, H. (2016). Pickup and delivery service with manpower planning in hong kong public hospitals. *Transportation Science*, 51(2):688–705.

Lin, J.-J. and Yu, C.-J. (2013). A bikeway network design model for urban areas. *Transportation*, 40(1):45–68.

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269.

Liu, M., Luo, Z., and Lim, A. (2015). A branch-and-cut algorithm for a realistic dial-a-ride problem. *Transportation Research Part B: Methodological*, 81:267–288.

Liu, R., Tao, Y., and Xie, X. (2019). An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits. *Computers & Operations Research*, 101:250–262.

Liu, R., Xie, X., Augusto, V., and Rodriguez, C. (2013). Heuristic algorithms for a vehicle routing problem with simultaneous delivery and pickup and time windows in home health care. *European Journal of Operational Research*, 230(3):475–486.

Liu, Y., Szeto, W., and Ho, S. C. (2018). A static free-floating bike repositioning problem with multiple heterogeneous vehicles, multiple depots, and multiple visits. *Transportation Research Part C: Emerging Technologies*, 92:208–242.

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

Louveaux, F. and Salazar-González, J.-J. (2009). On the one-commodity pickup-and-delivery traveling salesman problem with stochastic demands. *Mathematical Programming*, 119(1):169–194.

Lu, Y., Benlic, U., and Wu, Q. (2019). A population algorithm based on randomized tabu thresholding for the multi-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research*, 101:285–297.

Lu, Y., Benlic, U., and Wu, Q. (2020). An effective memetic algorithm for the generalized bike-sharing rebalancing problem. *Engineering Applications of Artificial Intelligence*, 95:103890.

Mahmoodian, V., Zhang, Y., and Charkhgard, H. (2022). Hybrid rebalancing with dynamic hubbing for free-floating bike sharing systems. *International Journal of Transportation Science and Technology*, 11(3):636–652.

Mancini, S. (2016). A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based matheuristic. *Transportation Research Part C: Emerging Technologies*, 70:100–112.

Marsh, M. T. and Schilling, D. A. (1994). Equity measurement in facility location analysis: A review and framework. *European Journal of Operational Research*, 74(1):1–17.

Masmoudi, M. A., Braekers, K., Masmoudi, M., and Dammak, A. (2017). A hybrid genetic algorithm for the heterogeneous dial-a-ride problem. *Computers & Operations Research*, 81:1–13.

Masmoudi, M. A., Hosny, M., Demir, E., and Pesch, E. (2020). Hybrid adaptive large neighborhood search algorithm for the mixed fleet heterogeneous dial-a-ride problem. *Journal of Heuristics*, 26(1):83–118.

Masson, R., Lehuédé, F., and Péton, O. (2013). An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science*, 47(3):344–355.

Melachrinoudis, E., Ilhan, A. B., and Min, H. (2007). A dial-a-ride problem for client transportation in a health-care organization. *Computers & Operations Research*, 34(3):742–759.

Mesbah, M., Thompson, R., and Moridpour, S. (2012). Bilevel optimization approach to design of network of bike lanes. *Transportation Research Record*, 2284(1):21–28.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.

Mladenović, N., Urošević, D., Ilić, A., et al. (2012). A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *European Journal of Operational Research*, 220(1):270–285.

Muelas, S., LaTorre, A., and Peña, J.-M. (2013). A variable neighborhood search algorithm for the optimization of a dial-a-ride problem in a large city. *Expert Systems with Applications*, 40(14):5516–5531.

Nair, D., Grzybowska, H., Fu, Y., and Dixit, V. (2018). Scheduling and routing models for food rescue and delivery operations. *Socio-Economic Planning Sciences*, 63:18–32.

Ni, W., Shu, J., and Song, M. (2018). Location and emergency inventory pre-positioning for disaster response operations: Min-max robust model and a case study of yushu earthquake. *Production and Operations Management*, 27(1):160–183.

Noon, C. E. (1988). *The generalized traveling salesman problem*. PhD thesis, University of Michigan.

Or, I. (1976). Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. *Ph. D. Thesis, Department of Industrial Engineering and Management Science, Northwestern University*.

Osaba, E., Yang, X.-S., Fister Jr, I., Del Ser, J., Lopez-Garcia, P., and Vazquez-Pardavila, A. J. (2019). A discrete and improved bat algorithm for solving a medical goods distribution problem with pharmacological waste collection. *Swarm and Evolutionary Computation*, 44:273–286.

Ozbaygin, G., Karasan, O., and Yaman, H. (2018). New exact solution approaches for the split delivery vehicle routing problem. *EURO Journal on Computational Optimization*, 6(1):85–115.

Pacheco, J. and Laguna, M. (2020). Vehicle routing for the urgent delivery of face shields during the covid-19 pandemic. *Journal of Heuristics*, 26(5):619–635.

Pal, A. and Zhang, Y. (2017). Free-floating bike sharing: Solving real-life large-scale static rebalancing problems. *Transportation Research Part C: Emerging Technologies*, 80:92–116.

Palacio, J. D. and Rivera, J. C. (2019). Mixed-integer linear programming models for one-commodity pickup and delivery traveling salesman problems. In *Workshop on Engineering Applications*, pages 735–751. Springer.

Palacio, J. D. and Rivera, J. C. (2022). A multi-start evolutionary local search for the one-commodity pickup and delivery traveling salesman problem. *Annals of Operations Research*, 319:979–1011.

Pan, L., Liu, X., Xia, Y., and Xing, L.-N. (2020). Tabu search algorithm for the bike sharing rebalancing problem. *IEEE Access*, 8:144543–144556.

Parragh, S. N. and Cordeau, J.-F. (2017). Branch-and-price and adaptive large neighborhood search for the truck and trailer routing problem with time windows. *Computers & Operations Research*, 83:28–44.

Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2010). Variable neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 37(6):1129–1138.

Parragh, S. N. and Schmid, V. (2013). Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 40(1):490–497.

Perboli, G., Tadei, R., and Vigo, D. (2011). The two-echelon capacitated vehicle routing problem: Models and math-based heuristics. *Transportation Science*, 45(3):364–380.

Pfeiffer, C. and Schulz, A. (2021). An ALNS algorithm for the static dial-a-ride problem with ride and waiting time minimization. *OR Spectrum*, pages 1–33.

Pollock, S. M. and Maltz, M. D. (1994). Operations research in the public sector: An introduction and a brief history. *Handbooks in Operations Research and Management Science*, 6:1–22.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002.

Prins, C. (2009). A GRASP $\times$ evolutionary local search hybrid for the vehicle routing problem. In *Bio-inspired algorithms for the vehicle routing problem*, pages 35–53. Springer.

Rainer-Harbach, M., Papazek, P., Raidl, G. R., Hu, B., and Kloimüllner, C. (2015). Pilot, grasp, and vns approaches for the static balancing of bicycle sharing systems. *Journal of Global Optimization*, 63(3):597–629.

Rais, A. and Vianaa, A. (2011). Operations research in healthcare: A survey. *International Transactions in Operational Research*, 18(1):1–31.

Raviv, T., Tzur, M., and Forma, I. a. (2013). Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Transportation and Logistics*, 2(3):187–229.

Regue, R. and Recker, W. (2014). Proactive vehicle routing with inferred demand to solve the bikesharing rebalancing problem. *Transportation Research Part E: Logistics and Transportation Review*, 72:192–209.

Reiss, S. and Bogenberger, K. (2017). A relocation strategy for munich's bike sharing system: Combining an operator-based and a user-based scheme. *Transportation Research Procedia*, 22:105–114.

Resende, M. G. and Ribeiro, C. C. (2016). *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer-Verlag New York, 1 edition.

Rey, D., Almi'ani, K., and Nair, D. J. (2018). Exact and heuristic algorithms for finding envy-free allocations in food rescue pickup and delivery logistics. *Transportation Research Part E: Logistics and Transportation Review*, 112:19–46.

Rist, Y. and Forbes, M. A. (2021). A new formulation for the dial-a-ride problem. *Transportation Science*, 55(5):1113–1135.

Rivera, J. C., Afsar, H. M., and Prins, C. (2013). Multistart evolutionary local search for a disaster relief problem. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 129–141. Springer.

Rodríguez-Martín, I. and Salazar-González, J. J. (2012). A hybrid heuristic approach for the multi-commodity one-to-one pickup-and-delivery traveling salesman problem. *Journal of Heuristics*, 18(6):849–867.

Ropke, S. and Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472.

Rottkemper, B., Fischer, K., Blecken, A., and Danne, C. (2011). Inventory relocation for overlapping disaster settings in humanitarian operations. *OR Spectrum*, 33(3):721–749.

Sabouhi, F., Bozorgi-Amiri, A., Moshref-Javadi, M., and Heydari, M. (2019). An integrated routing and scheduling model for evacuation and commodity distribution in large-scale disaster relief operations: a case study. *Annals of Operations Research*, 283(1):643–677.

Sacramento, D., Pisinger, D., and Ropke, S. (2019). An adaptive large neighborhood search metaheuristic for the vehicle routing problem with drones. *Transportation Research Part C: Emerging Technologies*, 102:289–315.

Sakiani, R., Seifi, A., and Khorshiddoust, R. R. (2020). Inventory routing and dynamic redistribution of relief goods in post-disaster operations. *Computers & Industrial Engineering*, 140:106219.

Salazar-González, J.-J. and Santos-Hernández, B. (2015). The split-demand one-commodity pickup-and-delivery travelling salesman problem. *Transportation Research Part B: Methodological*, 75:58–73.

Schuijbroek, J., Hampshire, R. C., and Van Hoeve, W.-J. (2017). Inventory rebalancing and vehicle routing in bike sharing systems. *European Journal of Operational Research*, 257(3):992–1004.

Shaheen, S. A., Martin, E. W., and Cohen, A. P. (2013). Public bikesharing and modal shift behavior: A comparative study of early bikesharing systems in north america. *International Journal of Transportation*, 1(1).

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pages 417–431. Springer.

Sherali, H. D. and Smith, J. C. (2001). Improving discrete model representations via symmetry considerations. *Management Science*, 47(10):1396–1407.

Shi, X., Zhao, F., and Gong, Y. (2009). Genetic algorithm for the one-commodity pickup-and-delivery vehicle routing problem. In *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 1, pages 175–179. IEEE.

Shi, Y., Boudouh, T., Grunder, O., and Wang, D. (2018). Modeling and solving simultaneous delivery and pick-up problem with stochastic travel and service times in home health care. *Expert Systems with Applications*, 102:218–233.

Shu, J., Chou, M. C., Liu, Q., Teo, C.-P., and Wang, I.-L. (2013). Models for effective deployment and redistribution of bicycles within public bicycle-sharing systems. *Operations Research*, 61(6):1346–1359.

Shui, C. and Szeto, W. (2018). Dynamic green bike repositioning problem–a hybrid rolling horizon artificial bee colony algorithm approach. *Transportation Research Part D: Transport and Environment*, 60:119–136.

Shui, C. and Szeto, W. (2020). A review of bicycle-sharing service planning problems. *Transportation Research Part C: Emerging Technologies*, 117:102648.

Singla, A., Santoni, M., Bartók, G., Mukerji, P., Meenen, M., and Krause, A. (2015). Incentivizing users for balancing bike sharing systems. In *Twenty-Ninth AAAI conference on artificial intelligence*.

Smith, S. L. and Imeson, F. (2017). GLNS: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, 87:1–19.

Sohn, K. (2011). Multi-objective optimization of a road diet network design. *Transportation Research Part A: Policy and Practice*, 45(6):499–511.

Sun, P., Veelenturf, L. P., Hewitt, M., and Van Woensel, T. (2020). Adaptive large neighborhood search for the time-dependent profitable pickup and delivery problem with time windows. *Transportation Research Part E: Logistics and Transportation Review*, 138:101942.

Tüű-Szabó, B., Földesi, P., and Kóczy, L. T. (2020). The discrete bacterial memetic evolutionary algorithm for solving the one-commodity pickup-and-delivery traveling salesman problem. In

Computational Intelligence and Mathematics for Tackling Complex Problems, pages 15–22. Springer.

Usama, M., Shen, Y., and Zahoor, O. (2019). A free-floating bike repositioning problem with faulty bikes. *Procedia Computer Science*, 151:155–162.

Van Breedam, A. (1994). *An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a Selectrion of Problems with Vehicle-related, Customer-related, and Time-related Constraints*. Ph.D. thesis, University of Antwerp.

Vidović, M., Ratković, B., Bjelić, N., and Popović, D. (2016). A two-echelon location-routing model for designing recycling logistics networks with profit: MILP and heuristic approach. *Expert Systems with Applications*, 51:34–48.

Villegas, J. G., Prins, C., Prodhon, C., Medaglia, A. L., and Velasco, N. (2010). GRASP/VND and multi-start evolutionary local search for the single truck and trailer routing problem with satellite depots. *Engineering Applications of Artificial Intelligence*, 23(5):780–794.

Villegas, J. G., Prins, C., Prodhon, C., Medaglia, A. L., and Velasco, N. (2013). A matheuristic for the truck and trailer routing problem. *European Journal of Operational Research*, 230(2):231–244.

Wang, J., Tsai, C.-H., and Lin, P.-C. (2016). Applying spatial-temporal analysis and retail location theory to public bikes site selection in taipei. *Transportation Research Part A: Policy and Practice*, 94:45–61.

Wang, K., Shao, Y., and Zhou, W. (2017). Matheuristic for a two-echelon capacitated vehicle routing problem with environmental considerations in city logistics service. *Transportation Research Part D: Transport and Environment*, 57:262–276.

Wang, Y. and Szeto, W. (2018). Static green repositioning in bike sharing systems with broken bikes. *Transportation Research Part D: Transport and Environment*, 65:438–457.

Wang, Y. and Szeto, W. (2021a). The dynamic bike repositioning problem with battery electric vehicles and multiple charging technologies. *Transportation Research Part C: Emerging Technologies*, 131:103327.

Wang, Y. and Szeto, W. (2021b). An enhanced artificial bee colony algorithm for the green bike repositioning problem with broken bikes. *Transportation Research Part C: Emerging Technologies*, 125:102895.

Wohlgemuth, S., Oloruntoba, R., and Clausen, U. (2012). Dynamic vehicle routing with anticipation in disaster relief. *Socio-Economic Planning Sciences*, 46(4):261–271.

Wolf, S. and Merz, P. (2007). Evolutionary local search for the super-peer selection problem and the p-hub median problem. In *International Workshop on Hybrid Metaheuristics*, pages 1–15. Springer.

Xiao, Y., Zhao, Q., Kaku, I., and Xu, Y. (2012). Development of a fuel consumption optimization model for the capacitated vehicle routing problem. *Computers & Operations Research*, 39(7):1419–1431.

Yi, W. and Kumar, A. (2007). Ant colony optimization for disaster relief operations. *Transportation Research Part E: Logistics and Transportation Review*, 43(6):660–672.

Zhang, B., Li, X., and Saldanha-da Gama, F. (2022). Free-floating bike-sharing systems: New repositioning rules, optimization models and solution algorithms. *Information Sciences*, 600:239–262.

Zhang, D., Yu, C., Desai, J., Lau, H., and Srivathsan, S. (2017). A time-space network flow approach to dynamic repositioning in bicycle sharing systems. *Transportation Research Part B: Methodological*, 103:188–207.

Zhang, J., Meng, M., Wong, Y. D., Ieromonachou, P., and Wang, D. Z. (2021). A data-driven dynamic repositioning model in bicycle-sharing systems. *International Journal of Production Economics*, 231:107909.

Zhang, Z., Liu, M., and Lim, A. (2015). A memetic algorithm for the patient transportation problem. *Omega*, 54:60–71.

Zhao, F., Li, S., Sun, J., and Mei, D. (2009). Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Industrial Engineering*, 56(4):1642–1648.