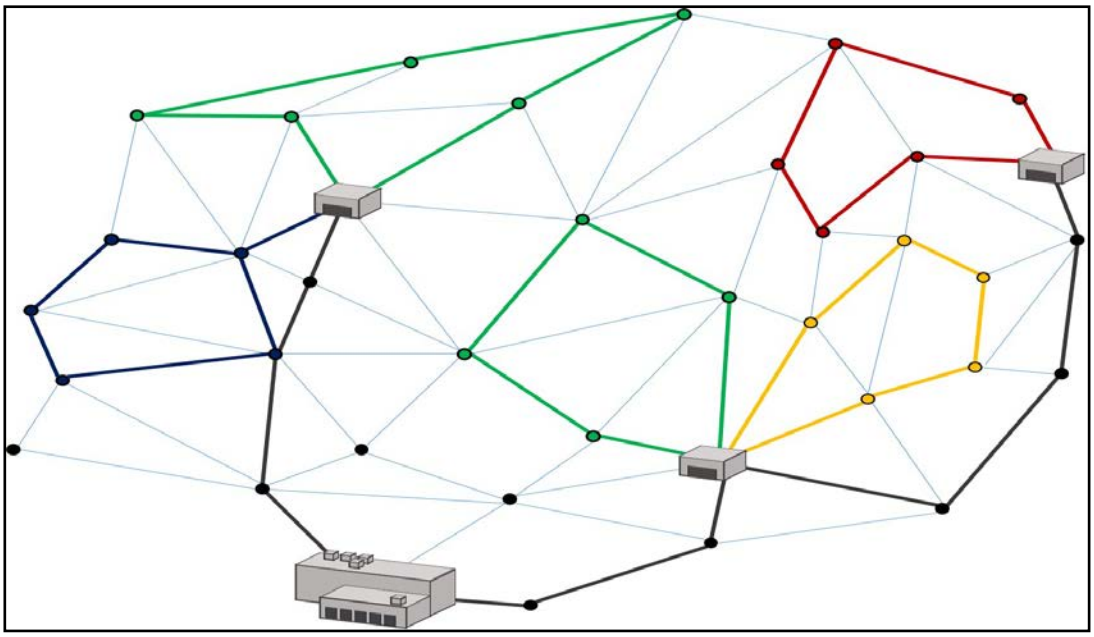
Selective and co-modal transport : models and algorithms

Youcef Amarouche

Par **Youcef AMAROUCHE**

Selective and co-modal transport : models and algorithms

Thèse présentée
pour l'obtention du grade
de Docteur de l'UTC



Soutenue le 2 décembre 2019
Spécialité : Informatique et Sciences et Technologies de l'Information et des Systèmes : Unité de recherche Heudyasic (UMR-7253)

D2522

A thesis presented for the degree of Doctor

UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

Spécialité : Informatique et Sciences et Technologies de l'Information et des Systèmes

---

Selective and co-modal transport: models and algorithms

02/12/2019

---

by **Youcef AMAROUCHE**

Jury composition:

Jacques CARLIER	Univ. de Technologie de Compiègne
Rym N. GUIBADJ	Univ. du Littoral Côte d'Opale
Sohaib LAFIFI	Univ. D'Artois
Aziz MOUKRIM	Univ. de Technologie de Compiègne
Ammar OULAMARA	Université de Lorraine
Gabriel PLASSAT	Agence De l'Environnement et de la Maîtrise de l'Énergie (ADEME)
Caroline PRODHON	Univ. de Technologie de Troyes

List of Publications

International conferences

- **Youcef Amarouche**, Rym Nesrine Guibadj, and Aziz Moukrim, "A Neighborhood Search and Set Cover Hybrid Heuristic for the Two-Echelon Vehicle Routing Problem", 18th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2018, August 23-24, 2018, Helsinki, Finland, pp. 11:1-11:15, 2018.
- **Youcef Amarouche**, Rym Nesrine Guibadj, and Aziz Moukrim, "A Set Cover based heuristic for the Two-Echelon Vehicle Routing Problem", in OR2018, International Conference on Operations Research, 2018.

National conferences

- **Youcef Amarouche**, Rym Nesrine Guibadj, and Aziz Moukrim, "Méthode de résolution en deux phases pour le problème de tournées de véhicules à deux-échelons", in ROADEF 2017, 18ème congrès de la société française de recherche opérationnelle et d'aide à la décision, 2017.

Submitted journal article

- **Youcef Amarouche**, Rym Nesrine Guibadj, Elhadja Chaalal and Aziz Moukrim, "An Effective Multi-Start Iterated Local Search Algorithm for the Team Orienteering Problem with Time Windows". [Submitted to Computers & Operational Research].

Abstract

Title: Selective and co-modal transport: models and algorithms

The notion of "co-modality" was introduced by the European Commission in 2006 as part of its new transport policy. It refers to the "efficient use of different modes on their own and in combination" for the purpose of achieving "an optimal and sustainable utilisation of resources" [55]. Unlike previous European transport policies, co-modality does not seek to oppose road transportation to its alternatives, but rather seeks to take advantage of the domains of relevance of different transportation modes and of their combinations to optimize services. In this thesis, we focus on developing new solution algorithms for variants of vehicle routing problems for city logistics systems arising in the wake of co-modality. We were particularly interested in developing effective solution methods for selective variants of the vehicle routing problem and two-echelon variants.

First, we address the Team Orienteering Problem with Time Windows (TOPTW), a selective variant of routing problems that takes into account customer availability. We propose an effective algorithm based on a neighborhood search that alternates between two different search spaces, and uses a long term memory mechanism to benefit from information gathered while exploring the search space, to solve the TOPTW.

In the second part of this work, we deal with the Two-Echelon Vehicle Routing Problem (2E-VRP), a variant of the problem that introduces intermediate facilities, refered to as "satellites". In a two echelon system, freight is first moved from the depot to the satellites using large trucks, and then delivered from the satellites to the customers using smaller vehicles. To solve the 2E-VRP, we introduce a novel algorithm that combines heuristic methods with mathematical programming techniques.

Finally, we consider the Orienteering Problem with Hotel Selection (OPHS), another selective variant that shares similarities with the 2E-VRP, namely, the use of intermediate facilities called hotels. For this problem, we propose a new integer linear programming model, valid inequalities and a Branch-&-Cut solution method.

Extensive experimentation on benchmark instances available in the literature demonstrate the competitiveness of our solution methods.

Keywords: urban distribution, city logistics, co-modality, selective vehicle routing problems, two-echelon routing problems, operations research, exact and heuristic solution methods.

Contents

List of figures

List of tables

Liste des algorithmes

Introduction

Freight distribution is one of the essential activities in today's society, and transport infrastructure and accessibility have become key factors for regional development. With growing urbanization and e-Commerce thriving, urban distribution has grown to become a large part of freight distribution, and a key element of the urban economy. However, urban transport, if poorly managed, is also a major source of noise, pollution, congestion, and a disturbance to the citizens' well-being. Both people and goods move in the urban environment, the former transported by their individual vehicles and collective transports, the latter by freight carriers and shippers. As traffic increases in towns and city centers, a better way to ensure an efficient and effective urban mobility for both passengers and goods becomes essential. Urban space being a limited resource, it is commonly argued that the movement of passengers and goods in urban areas have a strong interaction. In this context, the concepts of City Logistics emerged as viable means to mitigate the problem. The goal of City Logistics is to optimize freight transportation within city areas while considering traffic congestion and environmental issues as well as costs and benefits to the freight shippers [237].

According to the European Commission [56], rethinking urban mobility involves optimizing the use of all the available resources. This can be done by organizing "co-modality" between the public transport modes (train, tram, metro, bus, taxi) and private transport modes. The major benefit of such policy is to reconcile private transport and public transport interests instead of pitting them against each other. The notion of "co-modality" was introduced by the European Commission in 2006 as part of its new transport policy. It refers to the "efficient use of different modes on their own and in combination" for the purpose of achieving "an optimal and sustainable utilization of resources" [55]. Unlike previous European transport policies, co-modality does not seek to oppose road transportation to its alternatives, but rather seeks to take advantage of the domains of relevance of different transportation modes and of their combinations to optimize services. In order to improve urban mobility, different logistics systems have been proposed.

Potential solutions for urban freight distribution include *Cooperative freight transport*

systems (CFTS) and *shared "passengers & goods" transport systems*. CFTS can be defined as "systems in which multiple entities cooperatively use and operate the whole or a part of the transport elements of their logistic activities" [235]. Such systems can reduced the number of vehicles used and increase their utilization. As for shared "passengers & goods" transport systems, they are systems that make use of the existing networks of collective transport for freight distribution. They can be implemented by adapting the existing network, or by using the spare capacity of public transport vehicles (e.g. buses, tramways, etc.) to transport goods [250].

When designing City Logisitcs solutions, decision-makers often rely on optimization models to evaluate the potential of a solution before implementing it which, can be costly. Among optimization models, Vehicle Routing Problems (VRPs) are essential tools for modeling City Logistics, and since their advent in the 1960s, they have been widely used in practice to tackle transportation and logistics related problems.

Basic vehicle routing problems require few information, namely the location of customers, road network conditions, travel times/distances, and customer requests. However, motivated by real life applications, several extensions and variants were studied over te years. They include further information and specific constraints on the delivery process such as time windows for customer availability, traffic regulations, variable travel times, periodic planing, environmental costs, etc. Solving these problems aims at optimizing the utilization of the available transportation resources while satisfying both carrier and customer requirements.

In this thesis, we develop effective solution approaches for vehicle routing problem variants arising in City Logistics. In particular, we are interested in two variants of the problem: selective vehicle routing problems and the two-echelon vehicle routing problem. Selective routing problem have the characteristic that not all customer demands need to be satisfied. In the Team Orienteering Problem (TOP), given a limited fleet of vehicles, and a set of customers associated with non-negative profits, one must find a set of routes that visit a subset of customers and maximize the total collected profit. Routes must start and end respectively, at fixed departure and arrival locations. Moreover, the profit associated with each customer can be collected at most once. The Two-Echelon Vehicle routing problem (2E-VRP), on the other hand, is a variant where delivery from one or more depots to the customers is managed by shipping and consolidating freight through intermediate facilities called *satellites*. Freight is first moved from the depots to the satellites using large trucks, and then delivered from the satellites to the customers using smaller vehicles. The 2E-VRP models logistics systems that rely on Urban Consolidation Centers for last-mile

distribution.

This thesis is organized as follows. In the first part of Chapter 1, we provide an overview on freight distribution in urban areas, its importance, and its challenges with regards to environmental issues. It presents City Logistics as a potential source of solutions to the challenges of urban transportation, in particular, concepts such as co-modality and collaborative distribution. In the second part of the chapter, we provide more details on Vehicle Routing Problems namely, an overview of existing methods and problem variants related to this thesis, especially selective routing.

In Chapter 2, we propose an effective neighborhood search for the Team Orienteering Problem with Time Winows (TOPTW). The TOPTW is an extension of the TOP that takes into consideration customer availability. Our algorithm is based on a neighborhood search that (1) alternates between a *route search space*, and a *giant tour search space* using a powerful splitting algorithm, and (2) uses a long term memory mechanism to keep high quality routes encountered in elite solutions. The goal is to benefit from information gathered while exploring the search space. Extensive computational experiments highlight the competitiveness of our method as, it outperforms state-of-the-art algorithms in terms of overall solution quality and computational time.

The Two-Echelon Vehicle Routing Problem (2E-VRP) is tackled in Chapter 3. To solve the 2E-VRP, we designed a new hybrid heuristic method that relies on two components. The first component effectively explores the search space in order to discover a set of interesting routes using neighborhood search heuristics, in order to determine promising routes. These routes are then combined into high quality solution using a set covering formulation of the 2E-VRP. The resolution of the Set Covering model aims to identify high quality solutions that might have been missed by the neighborhood search procedure. Experimentation on benchmark instances from the literature show that our algorithm is on par with state-of-the-art algorithms as, it consistently achieves high quality solutions within short computational time.

The OPHS is a recently introduced routing problem that shares aspects with both the Orienteering Problem and the 2E-VRP. In the OPHS, given a number of days D, a set of intermediate facilities called "hotels", and a set of "Points Of Interest", the goal is to find a maximal profit tour of D connected trips which, must start and end at one of the available hotels, and visit each POI at most once. In Chapter 4, we propose a new mathematical model for the OPHS that uses an exponential number of constraints. We present several valid inequalities to enhance our mathematical model and solve it using

an exact algorithm. Furthermore, in order to compute lower bounds for the OPHS, we propose a fast multi-start heuristic based on the *order-first split-second* approach. To that effect, we designed a split procedure specifically for the OPHS. The results obtained on the OPHS benchmark instances show the effectiveness of our method.

Finally, in the conclusion of this thesis, we summarize our contributions and results, and discuss potential avenues for future research work.

General context

Urban freight distribution is an essential factor underpinning the socio-economic growth and development of countries. However, urban transport is also one of the more disturbing factors for the quality of life in cities. It is associated with several issues, especially traffic congestion, noise, and pollution which, disturb people's well-being. As traffic increases in towns and city centers, a better way to ensure an efficient and effective urban transport system becomes essential. The concepts of City Logistics have the potential to provide efficient solutions to the issues related to urban freight distribution. One way of dealing with this issues is through the development of co-modal transportation systems. In this chapter, we present the concepts of City Logistics and co-modality. First, we recall issues related to urban distribution and present City Logistics and co-modality as means to deal with them. We then present the routing optimization problems involved and treated in this thesis with a discussion on the exact and heuristic methods used to solve them.

1.1 Urban distribution

Urban freight transport refers to "all movements of goods in to, out from, through or within the urban area made by light or heavy vehicles, including service transport and demolition traffic as well as waste and reverse logistics" [53]. It includes a wide range of transport operations and logistics activities.

Urban freight transport is a key factor in the economic development of cities, and have a significant impact on the life in cities. Indicators compiled from different studies show the significance of freight demand in cities from developed countries. Although collected for specific regions of the world, these indicators do converge [73]. They show that a city generates:

- 0.1 delivery or pick-up per person per day

- 1 delivery or pick-up per job per week
- 300 to 400 truck trips per 1000 people per day, and
- 30 to 50 tons of goods transported per person per year.

Nowadays, urban distribution activities are rapidly gaining importance, in regards to their economic, social and environmental impact. Rapid urbanization is being observed around the world. According to the Population Division of the United Nations Department of Economic and Social Affairs, more than half of the world's population lives in urban areas, and the proportion is expected to increase to 66% by 2050[1]. In Europe, more than 350 million people live in densely populated areas, and it is expected that the proportion of people living in city will reach 80% by 2020 [2]. Even tough a growing population could arguably be an opportunity for cities to prosper, it also means that cities will be facing an important challenge with regards to logistic issues. Furthermore, the ongoing growth of *e-commerce* is also a major reason of the increase of fright flow in urban distribution. The UN conference on Trade and Development estimates that global e-commerce sales grew 13% in 2017[2], and it is very likely to be keep growing in the future. E-commerce places a high priority on customer demands. As a result, e-retailers and logistics service providers are required to change their logistics to meet the costumer expectations while maintaining economically competitive operations. The problem is that, doing so increases the number of deliveries and creates a larger dispersion of delivery points.

Due to its continues growth, urban transport is becoming a major factor with negative impact on the quality of life. In large cities and in metropolitan areas, intensive daily delivery flow hinders road traffic and causes congestion. The narrow streets that characterize old European cities leave little room for passing traffic. Also, the lack of accommodations and reserved areas for loading and unloading operations, and the imbalance between available parking places and parking demand are factors that further add to the issue. Furthermore, urban distribution is more polluting than long haul distribution, because of the average age of the vehicles [73]. Air pollution is responsible for respiratory and cardiovascular diseases and is considered the leading environmental factor in premature death in the European Union. Figure 1.1 shows the evolution of CO_2 emissions by activity sector in France, between 1990 and 2017. Transport was responsible

[1]More than half of world's population now living in urban areas, UN survey finds - `https://news.un.org/en/story/2014/07/472752-more-half-worlds-population-now-living-urban-areas-un-survey-finds`

[2]Global e-Commerce sales surged to $29 trillion - `https://unctad.org/en/pages/PressRelease.aspx?OriginalVersionID=505`

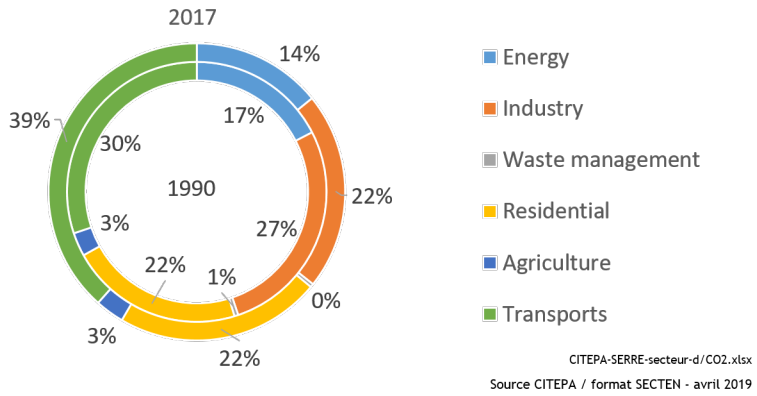


Figure 1.1 – CO_2 emissions in France by activity sector. [52]

for 38% of CO_2 emissions in 2017. It is estimated that freight transportation is responsible for a fourth of the CO_2 emissions caused by transport activities in European cities, and for 30 to 50% of other transport related pollutants [73]. To reduce emissions of air pollutants related to transport activities, while at the same time maintaining the flow of freight, cleaner vehicles could be used. However, it only addresses part of the problem as cleaner vehicles do not influence congestion nor the nuisance caused by freight distribution. Urban freight also accounts for a significant part of noise pollution. In the city of Bordeaux, in France, is was established that during the morning rush hour, the circulation of freight transport vehicles added five decibels (dB(A)) to the noise from the circulation of private cars [73]. Finally, Smells, vibrations and stress due to congestion are other forms of disturbances to life quality. Finally, in large cities, while commercial vehicles represent a small proportion of traffic, they are involved in the majority of accidents that also involve *vulnerable road users* [2]. These negative impacts of urban distribution influence the livability of cities.

The concept of City Logistics is one approach to solving the issues due to urban distribution. It aims to optimize freight transportation within city areas while considering traffic congestion and environmental issues as well as costs and benefits to the freight shippers [237].

1.2 City Logistics

Taniguchi et al. [237] define City Logistics as "the process for totally optimizing the logis-

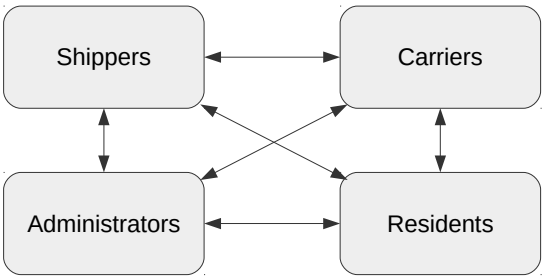


Figure 1.2 – Stakeholders in City Logistics.

tics and transport activities by private companies with support of advanced information systems in urban areas considering the traffic environment, the traffic congestion, the traffic safety and the energy savings within the framework of a market economy". It aims to globally optimize logistic systems within urban areas while considering the costs and benefits for shippers and freight carriers, and the implications on environmental issues and traffic congestion which, are of concern to the residents. In particular, the objectives of city logistics can be linked, in no particular order, to [235]:

- Mobility, i.e. ensuring the fluidity of the flow of goods in urban areas by alleviating traffic congestion.
- Sustainability, i.e. minimizing negative environmental impacts such as air pollution and noise pollution.
- Livability, i.e minimizing the inconvenience caused to residents.
- Resilience, especially for disaster relief logistics.

Since City logistics aims to optimize the last stage of supply chain management, it must consider the requirements of everyone that is involved in or impacted by urban freight distribution. According to Taniguchi et al. [237], in city logistics, there are four key stakeholders with their specific and often conflicting objectives: shippers, freight carriers, administrators, and residents.

- Shippers are manufacturer, wholesalers or retailers. They are the customers of the freight carriers and they want reliable logistics services with lower costs.
- Freight carriers want to be able to meet customer requests and provide better services at a lower cost. They also need to minimize the cost of collecting and delivering goods in order to maximize their profits.

- Residents are the people who live, work in the city. They are not keen on large trucks coming into their residential area. They want to minimize traffic congestion, noise, air pollution and traffic accidents near their homes and workplaces.

- City administrators have for objective to improve the economic development of the city, improve livability, and reduce environmental impact. Their role is to co-ordinate and facilitate city logistics initiatives.

Because of the difference in perspective between stakeholders, urban freight transport issues are very complicated. Thus, the design of city logistics solutions requires the collaboration between several disciplines in order to come up with schemes that reduce the economic, social and environmental costs of urban freight distribution [234].

Several city logistics solutions were implemented around the world. Taniguchi and Thompson [235] classifies them into four management categories : (1) traffic management, (2) better transport method, (3) harmony with other urban planning, and (4) others. Traffic management schemes aim at reducing carrier costs, traffic congestion and environmental impact. “Better transport methods” solutions are concerned with the smart use of the available resources and means of transport. Finally, “harmony with other urban planning” deals with the regulation of logistics facilities building, in order to be consistent with land use planning. Among this various city logistics schemes, we are interested in:

- The optimization of vehicle routing and scheduling which, is part of the traffic management category;

- Cooperative freight transport, in the category of better transport methods;

- Co-modal freight transport which, is also part of the better transport methods category.

1.2.1 Optimizing vehicle routing and scheduling

When designing City Logisitcs solutions, decision-makers need to test the potential solution in order to identify unexpected side-effects before proceeding with the implementation. Mathematical models are useful for explaining City Logistics systems and assessing the effects of applying policy measures on them. Typically, two types of models are used in the context of city logistics: optimization models and simulation models [237].

In general, optimization models are used to find how to make the best use of the available resources. Among optimization models, Vehicle Routing Problems (VRPs) are essential tools for modeling City Logistics. Their aim is to determine the customer-truck assignments and the visiting order of customers during freight distribution.. They are widely used in practice to minimize the costs of vehicle operations of freight carriers. Basic vehicle routing models require few information, namely the location of customers, road network conditions, travel times/distances, and customer requests. However, several extensions and variants were studied to accommodate further information and specific constraints on the delivery process such as time windows for customer availability, traffic regulations, variable travel times, ... etc. Yet, in practice, it is difficult to fully replicate the real-world conditions of logistics operations, and many elements are ignored in the modeling process. However, Vehicle Routing models remain useful for understanding the gap between the current level of efficiency in vehicle operations and the optimal solutions, and provide a good benchmark for the level of economic and environmental costs [235].

1.2.2 Cooperative freight transport

Cooperative freight transport systems (CFTS) are "systems in which multiple entities cooperatively use and operate the whole or a part of the transport elements of their logistics activities." [269]. As such, CFTS can provide shippers and carriers with cost saving opportunities that would be impossible to achieve if they were to focus only on their internal logistics process. Furthermore, sharing resources like vehicles or load capacity could reduce the number of necessary vehicles while increasing their utilization and decreasing the total travel time. Thus helping in the achievement of City Logistics goals like alleviating traffic congestion and reducing environmental impact.

Various types of CFTS have been discussed in both professional and academic literature. Of interest is so-called "Horizontal logistics cooperation" which, refers to the collaboration between firms that perform comparable logistics functions [261]; for example, collaboration between carriers or between shippers even if they are competitors.

Cooperation between carriers can be achieved by sharing or exchanging customer requests in order to improve their efficiency and profitability which, can result from an increase in capacity utilization, or from reduced total transportation costs due to improved transportation planning [261]. There are different techniques to implement order sharing, including *joint route planning*. It consists in regrouping all customer orders from all the participating carriers in a central pool then setting up efficient route plans for all requests

simultaneously [261]. This results in scales economies in terms of reduced travel distance, empty vehicle movements and number of required vehicles [71]. Several authors studied joint planning from a routing optimization point-of-view, and several VRP models were proposed [71, 150, 172, 74].

In shipper collaboration, multiple shippers consolidate their customer orders and outsource transportation to a single carrier. They try to identify sets of customer requests that can be presented to a carrier as bundle, instead of individually, in order to get better rates. Similarly to carrier collaboration, there are academic studies that study the operational planning and cost allocation in such systems using vehicle routing optimization models (see Defryn et al. [79], Lv et al. [175], Lyu et al. [176]).

The implementation of CFTS between shippers benefits greatly from the availability of Urban Consolidation Centers (UCCs). They are logistics facilities located in the vicinity of cities where goods destined to the area are delivered and from which consolidated deliveries as well as additional logistics services are realized [3]. Thus, UCCs play an essential role in the implementation of CFTS as bases for collecting and delivering goods but also by facilitating reception, processing, and dissemination of information [269].

A good example of CTFS with a consolidation center can be found in Motomachi Shopping Street in the Japanese city of Yokohama. An association of 300 shop owners established a consolidation center with the support of the Yokohama City government and the local police. Carriers were requested to deliver their orders destined for Motomachi Street to the consolidation center, where they are transferred to low emission vehicles fueled by compressed natural gas (CNG). This CFTS reduced the local emissions by decreasing total vehicle kilometers, increasing the load factor of cooperative vehicles compared with carriers' trucks, but also through lower emissions from CNG vehicles and human-powered carts compared with diesel-powered trucks [269].

In 2010, DHL established its depot at the consolidation center of the city Bath (United Kingdom). At this center, goods are consolidated for onward delivery using electric vehicles into central Bath. The electric vehicles used by the center reduced energy consumption by 55.7% compared to diesel trucks. From January 2011 to the end of April 2012, the number of deliveries to participating retailers was reduced on average by 76% [253].

1.2.3 Co-modal freight transport

Co-modal freight transports are relatively new systems which involve making use of the services offered by all available transport modes, including public passenger transport vehicles such as trains, trams, buses or taxis for transporting goods as well as passengers within urban areasThompson and Taniguchi [245]. The idea of such systems is to take advantage of the free load capacity of public transport vehicles which, is often underutilized outside of peak periods. The concept of co-modality is discussed in more details in Section 1.3.

1.3 Co-modality

The notion of *co-modality* was introduced by the European Commission in 2006 as part of its new transport policy. It refers to the "efficient use of different modes on their own and in combination" for the purpose of achieving "an optimal and sustainable utilisation of resources" [55]. Unlike previous European transport policies, co-modality does not seek to oppose road transportation to its alternatives, but rather seeks to take advantage of the domains of relevance of different transportation modes and of their combinations to optimize services. For example, railway transportation could be a viable alternative for transporting large shipments between districts of a metropolitan area. It is far less polluting than trucks, but is also less flexible and has higher transshipment costs.

Co-modality is best depicted by recent implementations in some cities that consist in combining the use of trucks with railways, trams, subways, inland water shipping, bicycles, and motorcycles for urban freight transport to promote more efficient and sustainable use of resources. *Cargo-tram & e-Tram* in Zurich (Switzerland) is an example of a co-modal system that uses tram infrastructure for waste collection within the city. CargoTram is used for the collection of bulky waste while e-Tram is reserved for electronic items. In Japan, the Yamato Transport Company uses a tram system for delivering goods to the Arashiyama district in Kyoto Taniguchi et al. [236]. The system uses a two carriage tram with one carriage being reserved for transporting goods into the district where, they are picked up by electric bicycles and delivered to customers. This system has considerably reduced the number of trucks used for delivering parcels. Leonardi et al. [162] report on a trial in which a major supplier of stationery and office supplies to businesses in central London replaced their diesel vans with electrically-assisted tricycles and electric vans operating from a urban micro-consolidation center. The results show that the total

distance traveled and the CO2 equivalent emissions per parcel delivered were considerably reduced. Furthermore, the trial proved successful from the company's perspective in terms of transport, environmental and financial costs.

The promotion of co-modality prompted another trend in City Logistics that involves integrating passenger and freight transport systems, more specifically, using passenger transport systems for carrying goods within the city. For example, spare capacity in buses, tram, and subways can be used for supplying retail stores, and Taxis can transport small parcels when transporting a passenger, or when they are idle. Combining passenger traffic and freight transport has the potential to lead to better efficiency and sustainability in urban distribution, by reducing the number or required vehicles to meet the transportation needs [249, 178]. Public transport companies can benefit from additional sources of income from carrying goods on less crowded vehicles. Shippers benefit by having convenient courier services. Fewer trucks also leads to less congestion, pollution, and noise. However, using public transport vehicles for transporting goods can require additional handling equipment which, might be impossible to use in crowded areas and in narrow streets. Furthermore, there is the risk of deteriorating the service level for passengers. Thus proper coordinating, planning and scheduling strategies need to be determined to improve the financial viability and the service quality of a co-modal on-demand services [91].

A pilot project was implemented in the city of Sapporo (Japan) to integrate the city's subway system with the truck delivery services operating between the suburbs and the city center [148]. This new delivery service operated during off-peak hours when passenger flow is low, and used delivery carts to move the parcel. The carts were loaded in ordinary subway cars on empty wheelchair spaces. Other workers unloaded them at the arrival station. The aim of this system was to mitigate urban transport problems, particularly during winter when heavy snowfall impairs traffic operation. It was well received by the public (subway users). Trentini et al. [250] performed a case study on the utilization of spare capacity on buses for the distribution of goods within the city of La Rochelle in France. Their study assumed that goods were packed into rolls, loaded from a single distribution center situated at the bus depot, and delivered to bus stops along the bus route. The delivery is then completed by tricycles. They present a mathematical formulation of the problem as a vehicle routing problem with transfers. The obtained results of the case study provide valuable information on the efficiency of the system. A similar study was performed by Masson et al. [178]. Li et al. [163, 164, 165] studied the benefits and the drawbacks of combining people and parcel flows using the same taxi network. They proposed two mathematical models to determine the best schedules and

routes for satisfying a set of people and parcel transport requests. The first model is based on the Dial-A-Ride Problem (a routing problem) and aims at finding a set of initial routes. The second model aims at inserting parcel requests into a set of predetermined routes. They performed numerical studies using real taxi trail data to show the potential of their approach. Finally, Ghilas et al. [116, 117] discussed the potential of integrating truck delivery with public transportation. They consider scheduling a fleet of vehicles for freight delivery, in a system where goods can be transported to station-hubs from where they can continue their journey on a public fixed schedule line. Afterwards, they may eventually be picked up again by another vehicle to be delivered to their final destination. Transferring freight requests to fixed-scheduled lines can be beneficial during off-peak hours when the utilization of the scheduled lines is low.

In terms of modeling, these new forms of distribution fall in the category of Vehicle Routing Problems with additional constraints and features such as capacity constraints, time windows, multiple-echelons, and transfers between vehicles. In this thesis, we address the design of decision tools that provide "co-modal" itineraries optimized in terms of cost with a focus on two variants of the selective vehicle routing problems, and the two-echelon vehicle routing problem.

1.4 Overview on vehicle routing problems

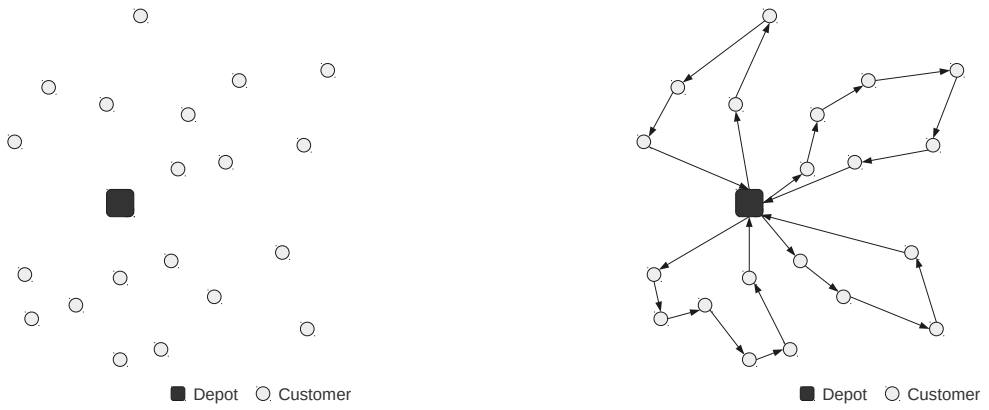
1.4.1 Vehicle Routing Problems

Vehicle routing problems (VRPs) are some of the most notable combinatorial optimization problems of the last fifty years. Due to their major economic impact, their wide range of applications, and the difficulty and the variety of characteristics combinations of many real-life settings, VRPs have been, and still are, the subject of intensive research effort. VRPs are concerned with the design of cost-effective delivery routes to serve a set of geographically-dispersed customers with respect to some constraints that stem from real-life settings.

The problem from which is originate vehicle routing problems is the Travelling Salesperson Problem (TSP) [78] which, is the oldest and most studied routing problem, and also one of the most recognizable optimization problems. The problem describes the situation where a salesperson leaves to visit n cities before returning to their starting point, and wants to find the sequence of visits that minimizes the travel distance. The TSP is usually defined

on a weighted digraph $G = (V, A)$, where $V = \{1, ..., n\}$ represents the city to be visited, and each arc $(i, j) \in A$ represents the path from city i to j. A travel distance (or cost) c_{ij} is associated with each arc (i, j). An optimal solution to the TSP corresponds to a shortest Hamiltonian cycle on graph G.

Usually, VRP refers to the Capacitated Vehicle Routing Problem; a generalization of the TSP where a fleet of m identical vehicles with limited load capacity Q has to visit the set of customers, instead of just one salesperson. Each vehicle starts from a depot-node, visits a subset of customers, with respect to capacity limitations, and then returns to the depot. As such, the VRP involves two decisions: one is the assignment of customer demands to capacitated vehicles which, coincides with a Bin Packing Problem, and the other is to design the best route for each vehicle which, coincides with a TSP. The VRP is defined on a weighted undirected graph $G = (V, E)$ where, V is a set of $|V| = n + 1$ nodes with node $v_0 \in V$ representing the depot and the remaining nodes the customers. A non-negative demand d_i is associated with each customer $i \in V \setminus \{v_0\}$. Each edge $(i, j) \in E$ represents the path from node i to j, and has a non-negative travel cost c_{ij}. A fleet of m homogeneous vehicles with a maximum load capacity Q is available at the depot to serve the customers. A solution of the VRP is a set of m or less delivery routes, i.e. sequences of deliveries to customers, that start and end at depot, in which each customer is visited exactly once and the total demand of the customers served in a single route does not exceed vehicle capacity. The objective of the VRP is to find a set of routes that minimizes the total travel cost. An example of a VRP instance with its solution is given in Figure 1.3.



(a) VRP instance. (b) VRP solution

Figure 1.3 – Example of a VRP instance with its solution.

1.4.2 Solution methods

Over the years, a wide array of solution methods for the VRP were presented in the literature, be it exact methods, heuristics, or meta-heuristics. These methods are surveyed in the works of Toth and Vigo [246], Baldacci et al. [22], Cordeau et al. [61], Gendreau et al. [113], Eksioglu et al. [93], Laporte [154], Vidal et al. [263], Toth and Vigo [248]. In the following, we briefly describe the main methods used to solve VRPs.

1.4.2.1 Exact methods

Exact methods for the VRP range from basic Branch-&-Bound algorithms to remarkably refined mathematical programming methods.

The first Branch-&-Bound (B-&-B) algorithm for the VRP was introduced by Christofides and Eilon [49]. They first reduce the problem to a m-TSP with $m + 1$ fictive depots, and then use the minimum spanning tree to compute a lower bound. Christofides et al. [50] introduced two new lower bounds which, greatly improved the performance of B-&-B algorithms. Laporte and Nobert [158] proposed another B-&-B algorithm based on integer linear programming formulation of the problem. Hadjiconstantinou et al. [129] used the lower bounds of Christofides et al. [50] in their improved B-&-B algorithm.

Few works deal with VRPs through Dynamic Programming (DP) methods. To the best of our knowledge, Eilon et al. [92] and Christofides et al. [51] were the only ones to solve the VRP with a DP method.Desrosiers et al. [84] proposed a DP approach to solve a dial-a-ride problem with Time Windows. Dynamic programming is mostly used as components of other exact approaches, for example, as means to compute bounds, or as part of heuristic methods.

Nowadays, exact algorithms for the VRP are mostly Branch-&-Cut, Branch-&-Price and Branch-&-Cut-&-Price algorithms. These algorithms are designed based on one of three main Integer Linear Programming (ILP) formulations of the problem [154, 218, 195]:

- The *Vehicle Flow Formulation* is an extension of the classical TSP formulation of Dantzig et al. [78], and uses binary variables x_e to indicate how many times edge $e \in E$ is traversed in the solution. This formulation was first introduced in [158, 159] and has been widely used ever since then. It can be reinforced through the inclusion of various valid inequalities.

- The *Set Partitioning Formulation* was first introduced by Balinski and Quandt [23]. This formulation considers feasible routes as the basic object to work with. Given the set of all feasible routes, it associates a binary variable to each route to indicate if it is part of the solution or not, i.e, a solution to the model decides which feasible routes to include. Because route feasibility is implicit, the SP formulation has the benefit of being generic and can model several variants of the problem without additional constraints. However, it requires a potentially exponential number of binary variables. The SP formulation is rather dominant for designing exact algorithms for the VRP with Time Windows, and performs very well for the VRP when used inside Branch-&-Cut-&-Price algorithms.

- The *Commodity Flow Formulation* uses binary variables to represent the load carried on arc (i, j). It was discussed by Gavish and Graves [107], but the authors did not report numerical results. Baldacci et al. [18] used a similar formulation with a Branch-&-Cut algorithm to solve the VRP. Their formulation is defined on an extended graph that includes a fictive copy of the depot-node to properly model single-customer routes.

1.4.2.2 Heuristics

Since the 1960s, numerous heuristic methods were proposed to solve the VRP. They can broadly be classified into *constructive* and *local improvement* heuristics. Laporte [154] refers to the methods presented in this section as *classical heuristics* because, in contrast with more recent algorithms and meta-heuristics, they only progress towards a local optimum in a *greedy* manner and do not allow the objective function to deteriorate from one iteration to the next.

As meta-heuristics rose to prominence, constructive heuristics fell into disuse as stand-alone solution methods. However, they are still very often employed as means to provide initial solutions to a wide range of meta-heuristics, or to generate new solutions inside meta-heuristics like GRASP. The *savings algorithm* of Clarke and Wright [54] is one of the better-known constructive heuristics. It is simple, easy to implement, and fast, which, is why it is still widely used inside more sophisticated algorithms. The *savings* algorithm starts by constructing back-and-forth routes (v_0, v_i, v_0) for each customer $v_i \in V \setminus \{v_0\}$, and then progressively merges routes at each iteration. More specifically, it identifies and merges the two routes $(v_0, \ldots, v_i, v_0)$ and $(v_0, v_j, \ldots, v_0)$ that maximize the *saving* $s_{ij} = c_{i0} + c_{0j} - c_{ij}$, under the condition that the resulting route is feasible. The algorithm stops

when it is no longer possible to merge routes. Numerous improvements and acceleration methods were proposed for this algorithm, notably by Gaskell [103] and Yellow [270].

Petal algorithms are a family of constructive heuristics, the idea of which is to solve the set partitioning formulation of the VRP over a subset of promising routes (called "petals") generated in a heuristic manner. The *sweep algorithm* of Gillett and Miller [118] is one of the earliest examples of such algorithms. This method assumes that customers are distributed on a plane, and uses a set non-overlapping petals. To generate petals, it scans the plane in a circular manner around the depot, starting from one customer and continuing in increasing polar angles. Customers are successively inserted, in this order, at the end of the current route as long as route constraints are satisfied. Whenever an insertion becomes infeasible due to route constrains, a new route is initiated. Several other petal algorithms with more complex petal generation heuristics were presented by Foster and Ryan [100], Ryan et al. [212] and Renaud et al. [207] to allow embedded and intersecting routes.

Beasley [25] presented a heuristic approach where the assignment and sequencing of customers are made in two different phases. This approach, usually referred to as *route-first cluster-second* algorithm, starts by solving a TSP on the VRP graph to obtain a *giant tour* which determines the order of visits. The giant tour is then split into several feasible routes by solving a shortest path problem on an auxiliary graph. For a long time, route-first cluster-second heuristics did not receive much attention, but they started gaining traction since Prins [199] used them in a very well-performing genetic algorithm for the CVRP. Another approach, called *cluster-first route second* heuristic was proposed by Fisher and Jaikumar [99]. They used a clustering algorithm to group customer visits into subsets, followed by TSP optimization to find the best route for each cluster. However, they received less attention, since they needed significant computational effort for routes optimization.

Improvement heuristics are used to optimize a complete VRP solution, usually generated by means of a constructive heuristic. Improvement heuristics can be divided into *intra-route* moves which, improve each route separately, and *inter-route* moves which, operate on multiple routes simultaneously. The most popular intra-route operators are improvement heuristics that were designed for the TSP, such as the $\lambda - Opt$ exchanges of Lin [168]. The latter consists in removing λ edges from the solution, and replacing them by λ other ones. In Lin and Kernighan [169], the value of λ is modified dynamically throughout the search, but in VRP literature, neighborhoods with a fixed value of λ are more commonly used; in particular, $2 - Opt$ and $3 - Opt$ exchanges.

Inter-route moves mostly consist in removing one or several visits from their current route and relocating elsewhere. Among the most commonly used inter-route moves, *relocate* (also called *shift*) removes a sequence of visits from one route and reinserts them into another one, *swap* exchanges sequences of visits between two routes, and $2-Opt*$ removes two edges from two routes (one from each) and reconnects them differently. Worthy of note is that, other heuristics with much larger neighborhoods exist in the literature, for example, the *b-cyclic/k-transfer* moves of Thompson and Psaraftis [244] which, consist in generating a circular permutation of b routes and shifting k customers for each route to the next one in the permutation. As such, the three previously described heuristics can be considered special cases of the latter one.

One major issue of improvement heuristics is the cost of exploring a neighborhood, which can be impractical for large instances. Thus it is necessary to restrict evaluations to a reduced subset of possible moves. One popular technique to achieve this, called *granular search* [247], consists in avoiding the evaluation of moves that involve distant customers. For each customer i, a list of closest customers C_i is predetermined, and only moves that involve i and $j \in C_i$ are considered. Another way of reducing computation cost is to use pertinent information on sub-sequences of customers, like partial demands and distances, to speed the evaluation of a move's feasibility and cost [136].

1.4.2.3 Meta-heuristics

Meta-heuristics were applied to the VRP very early on, and nowadays they constitute the bulk of the literature on heuristic methods. Over the last fifteen years, a wide variety of VRP meta-heuristics were published, and it would take too long to elaborate on them, even if we were to stick to notable methods only. Rather, we will briefly describe the principle of each class of algorithms, while mentioning some important contributions.

1.4.2.3.1 Neighborhood search algorithms Starting from an initial solution, *Neighborhood Search algorithms* explore the solution space by moving from the current solution to another solution in its neighborhood. Typically, they combine inter-route improvement moves with intra-route moves for re-optimization. However, in contrast with classical improvement heuristics, the cost of the selected neighbor is not necessarily better then that of the current solution, thus, neighborhood search heuristics must include mechanisms to avoid cycling. Well-known Neighborhood search algorithms include Tabu Search (TS), Simulated Annealing (SA), Variable Neighborhood Search (VNS), and

Adaptive Large Neighborhood Search (ALNS)

The main idea of TS is to move from a solution to the best solution in a subset of neighboring solutions, even if it has a worse objective value. To avoid cycling, a short-term memory, called *tabu list*, is used to reject recently explored solutions or solutions sharing some attributes with the current solutions. However, if such a solution constitutes a new best known solution, it is accepted as an exception to the rule. During the 1990s and early 2000s, TS algorithms were prevalent among CVRP meta-heuristics and concepts introduced for TS algorithms inspired various other meta-heuristics. Notable TS implementations include the "Taburoute" of Gendreau et al. [109], the TS with Adaptive Memory of Taillard [230] and Rochat and Taillard [210], the Unified TS of Cordeau et al. [58, 60], and the Granular TS of Toth and Vigo [247].

In SA algorithms, at each iteration i, the new solution s is randomly selected in the neighborhood of the current one s_i. If s has a better objective value than s_t, it is accepted as the incumbent of the next iteration $s_{i+1} = s$. Otherwise, this solution is accepted with a probability p_i which is controlled by a parameter T called *temperature*. The higher the temperature, the higher the probability to accept a deteriorating solution. Temperature evolves during the search according to a cooling process. It starts high to favor diversification, then progressively decreases to accept fewer deteriorating moves to favor intensification. Osman [188] is one of the earliest applications of SA to the VRP. Dueck [89] and Li et al. [166] implemented a deterministic variant of SA, called *Record-to-Record* algorithm where, a solution is accepted if it is not much worse than the current one.

Variable Neighborhood Search (VNS) uses an ordered list of neighborhoods $N_1, N_2, \ldots, N_p$, which are usually of increasing complexity. Starting from an initial solution, the algorithm applies the first neighborhood of the list until it reaches a local minimum. When it does, it switches to the next one in the list. The search process goes back to the first neighborhood whenever it finds an improvement, or when the all neighborhoods have been explored. The reason behind this process is the fact that local optima are defined for a given neighborhood, thus switching neighborhoods might allow further improvement of the solution. Kytöjoki et al. [151] designed a VNS algorithm to solve large-scale real-life VRP instances.

Finally, Large Neighborhood Search algorithms and Adaptive LNS algorithms simultaneously destroy and repair parts of the current solution and are related to the ruin-and-recreate moves of Shaw [219]. Both usually use several destroy and repair heuristics. The

difference between the two is the way the choose the destroy and the repair heuristic to be used during each iteration. In LNS, the choice is usually random, while in ALNS, the probability of choosing a heuristic is a function of its success rate in previous iterations. The ALNS was successfully applied to the VRP by Pisinger and Ropke [194].

1.4.2.3.2 Population search method *Population search methods* are based on the improvement of a set solutions, called a *population*, using mechanisms often inspired from nature. Genetic Algorithms (GA) [134] are the best-knwon example of meta-heuristics from this category. They evolve a population of solutions through mechanisms such as crossover, selection and mutation. At each iteration, parent solutions are extracted from the population and recombined to generate child solutions (crossover) which, replace the worst individuals of the population (selection). Often, a mutation is applied to the child solutions after the crossover process, for the purpose of diversification. To the best of our knowledge, all genetic algorithms that were successfully applied to the VRP rely on local search procedures to guide the search towards promising solutions. The obtained algorithms are sometimes referred to as *memetic algorithms*. Furthermore, many of this algorithms rely on a route-first cluster-second approach with solutions being represented by a giant-tour with no trip delimiters. This strategy reduces the size of the search space and allows the use of simple crossover operators. Examples of successful implementations include the works of Prins [199, 202], Sörensen and Sevaux [227], Nagata and Bräysy [184] and Vidal et al. [262].

Among other population search methods that were successfully applied to the VRP, we mention Path-Relinking algorithms [133, 240]. Path-Relinking algorithms differ from GA, in the way they recombine solutions and in the size of the pool. Instead of random crossovers, PR algorithms use an initial solution in which they progressively insert characteristics of a guiding solution. By doing so, it creates a trajectory connecting the two of them and which, might contain a new improving solution. Finally, Ant Colony Optimization (ACO) approaches [87] mimic the behavior of ants foraging for food. Simply put, ants lay pheromones on their path as they look for food sources. Pheromones gradually accumulate on the shortest paths to the food source, which are then followed by more ants. In an ACO, several ants construct solutions using constructive heuristics that exploit information on search history (pheromone levels) to favor edges that appear frequently in good solutions. Examples of implementations of ACO for the VRP include Bullnheimer et al. [40], Bell and McMullen [27], Reimann et al. [206] and Yu et al. [271].

1.4.2.3.3 Hybrid algorithms Hybrid meta-heuristics are methods which combine concepts that were originally developed for independent methodologies and blend them into one single algorithm that benefits of their respective strengths. The hybridization can be in the form of a method where different techniques are juxtaposed, like two algorithms being called consecutively or concurrently, or otherwise in the form of an algorithm where the methods and the various concepts are indissociable from each other. Several hybrid methods have been applied to the VRP during the last decade. Some of them are based on the blending of population-based methods with neighborhood-based methods and local search, for example the combined GA and TS of Perboli et al. [191]. Yet again, most population search approaches for the VRP rely on local search which, in a way, makes them hybrid algorithms. Other hybrid approaches combine concepts from different Neighborhood-Search paradigms into a single algorithm, such as the restart procedures from GRASP, the probabilistic acceptance criterion from SA, or the use of variable neighborhoods like VNS. For example, Prins [201] combine a Greedy Randomized Adaptive Search Procedure (GRASP) with Evolutionary Local Search to solve the VRP, while Cordeau and Maischberger [62] blend concepts from Iterated Local Search, ruin-and-recreate moves, and a tabu memory.

Another increasingly popular form of hybridization is the combination of meta-heuristics with mathematical programming, resulting in the so-called *matheuristics*. The popular strategy among matheuristics consists in using integer programming to combine elements from promising solutions into a complete solution [210, 238, 123, 228].

1.4.3 Variants

Routing problems can be found in a variety of practical applications, each with their own set of requirements. Naturally, this wide range of settings led to the definition of many variants, intended to capture in more details the characteristics of real-life situations, or to better account for different decision contexts. The large variety of practical applications, characteristics and VRP variants is addressed by a vast literature, and for the sake of conciseness, it would be impractical to provide a detailed review on VRP variants herein. Therefore, we refer the reader to the works of Gendreau et al. [114], Vidal et al. [264], Toth and Vigo [248], and Braekers et al. [36] for comprehensive surveys on the matter.

In the following, we present some common VRP variants that are of relevance to the present work.

1.4.3.1 Distance Constrained Vehicle Routing Problems - DVRP

In the distance-constrained VRP, an additional limitation is imposed on route durations. It allows to account for real-life considerations, such as vehicle fuel autonomy or driver work-shift durations. Each customer is associated with a service time s_i and each edge (i, j) with a travel time t_{ij}. The duration of a route is the sum of the service times of its customers and the travel times between them, and must not exceed a limit L. In most of the classic benchmark instances from the literature, the travel time and the travel cost are considered the same, i.e. $t_{ij} = c_{ij}$ for all $(i, j) \in E$. There are only a few works that have considered exclusively the DVRP. Laporte et al. [155, 159] presented exact methods to solve this problem. Prins [199] designed a genetic algorithm where the chromosomes are permutations of n customers without trip delimiters, and their evaluation is done by an optimal splitting procedure like that of Beasley [25]. It should be noted that most of the heuristics and meta-heuristics for the classical VRP can be easily adapted to solve the DVRP.

1.4.3.2 Vehicle Routing Problems with Time Windows - VRPTW

In the Vehicle Routing Problem with Time Windows (VRPTW), the service of each customer must occur within a predefined time interval, known as a *time window*. Moreover, each customer is associated with a service duration, and each arc with a travel duration. Waiting times are allowed in case of early arrival at the customer location. On the other hand, late arrivals are either forbidden if *hard time windows* are considered, or are allowed at the cost of penalties if the time windows are *soft*. The VRPTW has been widely studied since the 1970s, and numerous exact and heuristic algorithms have been developed. For extensive reviews on the VRPTW, the reader is referred to Bräysy and Gendreau [37, 38], Kallehauge [143], Gendreau and Tarantilis [115], and Desaulniers et al. [83]. The most effective exact methods that were published over the last years (Jepsen et al. [139], Desaulniers et al. [82], Baldacci et al. [20], Pecin et al. [190]) are based on a set-partitioning formulation of the problem and can solve instances with up to 100 customers. the Branch-&-Cut-&-Price of Pecin et al. [190] is even able to solve most of the instance with up to 200 customers. State-of-the-art metaheuristics for the VRPTW are of various kinds, among them: Tabu Search [60], Iterated Local Search [62], Large Neighborhood Search [179, 198], Genetic Algorithms [265], Evolutionary Algorithms [208], Path Relinking [130], and Memetic Algorithms [185, 31]. The relaxation of time-constraints during the search process to allow the exploration of infeasible solutions

is a recurrent idea to improve the performance of meta-heuristics for the VRPTW.

1.4.3.3 Multi-Depot Vehicle Routing Problems - MDVRP

The Multi-Depot Vehicle Routing Problem (MDVRP) is a variant of the CVRP where several depots are considered for the service of customers. Each vehicle is assigned to a specific depot which, constitutes the departure and the arrival of its route. Typically, the supply capacity of a depot is considered, the size of the fleet is limited, and vehicle are homogeneous. The MDVRP describes a situation where customers are not evidently clustered around each depot, otherwise, the problem can be solved as several individual VRPs. The literature on the MDVRP is rather small when compared to the VRP or the VRPTW. Exact methods for the MDVRP were introduced in Laporte et al. [160], Baldacci and Mingozzi [19], Contardo and Martinelli [57]. Several heuristics and meta-heuristics were proposed for the MDVRP. Some of the recent ones are the ILS and Set Covering approach of Subramanian et al. [228], the Hybrid Genetic Search with Advanced Diversity Control of Vidal et al. [262], the parallel Tabu Search of Cordeau and Maischberger [62], the Unified hybrid Genetic Search of Vidal et al. [266], the hybrid granular TS algorithm was proposed by Escobar et al. [96], and the cooperative co-evolutionary algorithm of Oliveira et al. [186]. For more detailed reviews on the MDVRP, we refer the reader to papers of Ombuki-Berman and Hanshar [187], Karakatič and Podgorelec [144], Montoya-Torres et al. [182].

1.4.3.4 Split Delivery Vehicle Routing Problems - SDVRP

In the classical VRP, each customer is visited exactly once. However, a single visit might not be enough, if, for example, the requested quantity of goods in larger than vehicle capacity. The Split Delivery Vehicle Routing Problem (SDVRP) is a variant of the VRP where customer demands can be satisfied by multiple vehicles, each moving a part of their request. More than making the model more realistic, the SDVRP can also lead to savings in transportation costs and fleet size with respect to the VRP (see Dror and Trudeau [88], Archetti et al. [9]). The literature on the SDVRP is reviewed in Archetti and Speranza [10] and Irnich et al. [137]. Dror and Trudeau [88] proved that the SDVRP is NP-Hard despite it being a relaxation of the VRP. Furthermore, two versions of the SDVRP can be distinguished with respect to fleet-size policy: in the SDVRP-UF the fleet-size is unlimited, whereas in the SDVRP-LF, the fleet size is limited. Exact-solution approaches for the SDVRP were proposed by Belenguer et al. [26], Jin et al. [141], Jin et al. [142],

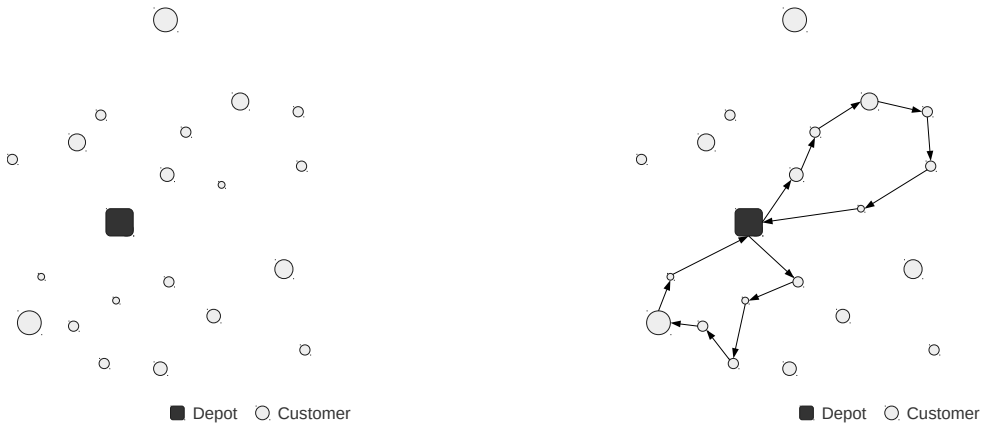
Archetti et al. [6], and Ozbaygin et al. [189]. Several meta-heuristics were developed for the SDVRP; many of them adopt a Tabu Search approach. Archetti et al. [12] implemented an algorithm that uses integer programming to explore promising parts of the search space identified by a Tabu Search. Aleman and Hill [1] presented a Tabu Search with a vocabulary building that determines attractive attributes from a population of solutions and uses them to construct new ones. Berbotto et al. [30] proposed a randomized Granular Tabu Search to solve the problem. Other heuristic methods include the Memetic Algorithm of Boudia et al. [32], the Attribute Based Hill Climber (ABHC) of Derigs et al. [81], the Iterated Local Search of Silva et al. [220], and the heuristic of Chen et al. [48].

1.4.3.5 Vehicle Routing Problems with Profits - VRPP

Vehicle Routing Problems with Profits (VRPP) arise in situations where for practical reasons, it is either not possible or not desirable to serve all customers. This may be the case if the company does not have the necessary resources to satisfy customer demands, or if the marginal cost for serving a customer, given a routing plan, out-weighs the marginal revenue gained from the service. In the latter case, the total profit could improve if those customers are left out. As such solving a VRPP implies an implicit choice of which customers to serve, and which to omit in order to maximize profit. Several routing problems fall into the category of VRPPs. We provide a brief description of some notable VRPP variants in Section 1.5.

1.5 Selective routing: the orienteering problem

Selective Vehicle Routing Problems, also known as Vehicle Routing Problems with Profits (VRPPs), are a class of VRPs where the service of some or all customers is optional, but rewarded with a prize. As such, an additional decision has to be made on whether or not optional customers should be served. In selective routing problems, each customer is associated with a profit (or score) which is collected at most once if the customer is served. The goal is to define a set of routes such that the total collected profit is maximized and travel costs are minimized. Hence, VRPPs can be formulated either as bi-objective discrete optimization problems, or as single objective problems where one of the goals is the objective function and a constraint is imposed on the other [5]. Figure 1.4 show an example of a selective VRP where only two vehicles are available. Customers with larger profits are represented with bigger nodes.



(a) Selective VRP instance. (b) Selective VRP solution

Figure 1.4 – Example of a selective VRP instance with its solution.

VRPPs can be used to model several classes of real-world applications. Tsiligirides [252] and Ramesh and Brown [204] describe the problem of scheduling the visits of salesperson with not enough time to visit all their customers. In Golden et al. [120], an application in home fuel delivery planing is described. Heating fuel must be delivered to a large number of customers on a daily basis such that the fuel inventory is maintained at an adequate level. Scores are used to represent urgency due to low levels of inventory. In Golden et al. [121], selective routing problems are used to route oil tankers to supply ships at different locations. Tang and Miller-Hooks [232] describe the application of a selective routing problem to determine the daily schedule of technicians to service customers while taking into account customer importance and task urgency. More recently, several papers used routing problems with profits to deal with the Tourist Trip Design Problem (TTDP) [259, 225, 256, 260, 104]. Usually, when visiting a city or a region, it is impossible to visit everything, and it is necessary to select which attractions someone is more likely to enjoy. The TTDP is concerned with making a feasible itinerary to visit these attractions in the available time span. Other applications of VRPPs found in the literature include the single-ring design problem when building telecommunication networks [243], collaborative logistics [167, 101], high-school athlete recruiting [41], and the scheduling of the daily operations of a steel rolling mill [15, 17].

The literature dealt with several variations of the VRPP, the most basic of them being variants with only one tour. The most notable of them are the Orienteering Problem (OP) [252], the Prize Collection TSP (PCTSP) [15, 16], and the Profitable Tour Problem (PTP) [80]. The main difference between these problems is their objective functions. In the OP, the goal is to maximize the total collected profit such that the travel cost does not exceed an upper bound, while on the PCTSP, the objective is to minimize the travel

cost under the constraint that the total collected profit cannot be smaller than a given threshold. On the other hand, the objective of the PTP is to maximize the difference between the total collected profit and the total travel cost. Among these three problems, the OP is the most studied in the literature.

The OP was introduced by Tsiligirides [252]. It originates from the orienteering sport, in particular the score orienteering events. In score events, participants are given a map indicating a large number of "control points" with associated scores. Instead of completing the course, participants need to score as many points as possible by visiting controls, and then return within the specified time limit, otherwise they receive large penalties. The OP is also referred to in the literature as the Selective Traveling Salesperson Problem [156, 111, 243], the Maximum Collection Problem [41], and the Bank Robber Problem [14]. The OP can be formulated on a complete graph $G = (V, A)$ where V is a set of $|V| = n + 2$ nodes and A the of arcs. Node $v_0 \in V$ and $v_{n+1} \in V$ correspond to the departure and the arrival depots, respectively, while the remaining nodes correspond to customers. A non-negative profit p_i is associated with each customer $i \in V \setminus \{v_0, v_{n+1}\}$. Each arc $(i, j) \in A$ represents the path from node i to j, and has a non-negative travel time t_{ij}. It is supposed that travel times satisfy the triangle inequality. One vehicle is available to visit a subset of customers and collect their profit. The profit of each customer can be collected at most once. The vehicle is uncapacitated but has a maximum route duration T_{max}. The vehicle route starts and ends at the departure and arrival nodes respectively. The objective of the OP is to find a vehicle route that maximizes the total collected profit while satisfying the maximum duration constraint on the route.

Several exact solution algorithms were proposed for the OP. Laporte and Martello [156] proposed schemes to derive upper and lower bounds for the OP, which they then used in an exact enumerative algorithm. Valid inequalities were later added to their formulation by Leifer and Rosenwein [161]. The latter used a cutting-plan algorithm to obtain better upper bounds. Ramesh et al. [205] used a Branch-&-Bound algorithm and Lagrangean relaxation to solve the OP. Additional families of valid inequalities were introduced by Fischetti et al. [98], Gendreau et al. [110] who used Branch-&-Cut algorithms to solve the problem. The algorithm of Fischetti et al. [98] is able to solve instances with up to 500 vertices.

Several authors proposed heuristic methods for the OP. Tsiligirides [252] introduced a stochastic and a deterministic heuristic for the OP. Both are based on the generation of a set of paths and then selecting the best one, but differ in how they generate the paths. The first uses a Monte-Carlo method to insert vertices into the path, while the

other uses a variation of the algorithm of Wren and Holliday [268]. Ramesh and Brown [204] designed a four-phase heuristic for the OP. First a vertex insertion phase builds a tour by relaxing the time limit constraint, then an improvement phase reduces the travel cost, after that vertices are removed during the third phase, and finally another insertion heuristic is used to improve the profit. Chao et al. [46] proposed a two-step heuristic. They used an ellipse of which, the focal points are the start and end vertex, and the major axis is T_{max}, to define a subset of reachable customers. Their heuristic generates different paths between the departure and the arrival. Afterwards, during each iteration, it improves the best one by means of node interchanges, then removes some customers from the best path obtained and restarts. Gendreau et al. [112] designed a tabu search algorithm that iteratively inserts clusters of vertices into the OP tour or removes a chain of vertices. Schilde et al. [216] tackled the bi-objective OP using a pareto Ant Colony Optimization Algorithm and a multi-objective VNS, both hybridize with path relinking procedures. They reported results on single objective OP instances which, show that their approach out-performed the previous ones.

The extension of the OP to multiple vehicles is known as the Team Orienteering Problem (TOP) [47]. It was first studied in Butt and Cavalier [41] under the name of Multiple Tour Maximum Collection Problem. Due to the complexity of the problem, research on the TOP focused mainly on heuristic algorithms, and few exact methods were developed. Butt and Ryan [42] solved the TOP using a column generation algorithm. Boussier et al. [35] proposed a Branch-&-Price algorithm for the TOP and the TOP with time windows. They used various acceleration techniques and two pre-processing procedures to increase the performance of their algorithm. Poggi et al. [196] proposed three different formulations for the TOP and a Branch-&-Cut-&-Price algorithm. Dang et al. [75] defined a set of dominance properties and valid inequalities, and solved the problem using a Branch-&-Cut algorithm. Keshtkaran et al. [147] enhanced the Branch-&-Price approach of Boussier et al. [35] with a new approach to solve the pricing sub-problem, new relaxations of the pricing sub-problem and valid inequalities. El-Hajj et al. [94] used a cutting-planes algorithm to solve the problem. They derived valid inequalities from incompatibility graphs, and from solving sub-instances to gain information on the original instance.

Several heuristics and metaheuristics were developed to solve the TOP. Chao et al. [47] introduced the first heuristic for the TOP by adapting their heuristic for the OP so it chooses the m best paths instead of only the best one. Tang and Miller-Hooks [232] developed a Tabu Search algorithm with adaptive memory that alternates between small and large neighborhoods. Archetti et al. [7] developed two tabu search algorithms, one

that only uses feasible solutions and one that also explores unfeasible solutions the search process. They also developed a slow and a fast variable neighborhood search. An Ant Colony Optimization (ACO) procedure was introduced by Ke et al. [145]. It uses four different methods to construct feasible solutions. Vansteenwegen et al. [254] proposed an algorithm which uses a combination of local search heuristics, and includes Guided local search (GLS) to improve some of the heuristics. A Skewed Variable Neighbourhood Search (SVNS) for the TOP was introduced by Vansteenwegen et al. [256]. It uses two types of intensification mechanisms, one to increase the score and the other to decrease the travel time. Diversification is carried out by three different procedures. Souffriau et al. [223] designed a Greedy Randomised Adaptive Search Procedure (GRASP) with Path Relinking. At each iteration, the algorithm generates an initial solution using a construct procedure that chooses which vertex to insert based on a greediness to randomness ratio. The initial solution is then improved using local search. Afterwards, the path relinking is applied on a pool of high quality solutions, and the new solution is then used to update said pool. Another population-based method is introduced in [33] who proposed a Memetic Algorithm (MA) for the TOP. The MA uses a giant-tour encoding to represent solutions. Individuals are represented as an ordered sequence of all the customers from which a valid solution is extracted using an Optimal Split Procedure. Muthuswamy and Lam [183] proposed a Discrete Particle Swarm Optimization (DPSO) algorithm. Dang et al. [76] proposed a PSO-based MA that extends the work of Bouly et al. [33]. This work was later extended by Dang et al. [75]. They proposed a new more effective optimal splitting procedure based on interval graph models. This new procedure has a smaller complexity than that of their previous works, and allows to explore more solutions without increasing the global computation time. Other meta-heuristics for the TOP include the Multi-start Simulated Annealing of Lin [170], the Genetic Algorithm of Ferreira et al. [97], and the Pareto Mimic algorithm of Ke et al. [146].

Over the last years, research on orienteering problems has been shifting focus towards variants and extensions of the OP and the TOP. Notable variants include Orienteering Problems with Time Windows [255, 251, 153, 152, 170, 135, 72], Time Dependant Orienteering Problems [102, 105], Capacitated Orienteering Problems [5, 239, 28], Arc Orienteering Problems [260, 11, 8, 106]. For more extensive surveys on orienteering problems, we invite the reader to refer to the works of Vansteenwegen et al. [257], Archetti et al. [13], Archetti and Speranza [11], and Gunawan et al. [128].

1.6 Conclusion

In this chapter, we recalled the concepts related to this thesis. In the first part of the chapter, we recalled key issues of urban distribution and presented City Logistics, that is the practices that seek to solve this issues. Among City Logistics research directions, we focused on "co-modality" and collaboration.

In the second part of the chapter, we described vehicle routing problems related to the topics discussed above. We recalled exact and heuristic methods that were developed to solve these problems. We then gave descriptions of some VRP variants that are of interest in the remaining chapters. Afterwards, we focused on selective routing problems, in particular, the Orienteering and the Team Orienteering Problem.

In the following chapters, we provide a more detailed description for the problems tackled in here and present the solution methods we proposed to solve them.

Effective Neighborhood Search with Optimal Splitting and Adaptive Memory for the Team Orienteering Problem with Time Windows

The Team Orienteering Problem with Time Windows (TOPTW) is an extension of the well-known Orienteering Problem. Given a set of locations, each one associated with a profit, a service time and a time window, the objective of the TOPTW is to plan a set of routes, over a subset of locations, that maximize the total collected profit while satisfying travel time limitations and time window constraints. In this chapter, we present an effective neighborhood search for the TOPTW based on (1) the alternation between two different search spaces, a *giant tour search space* and a *route search space*, using a powerful splitting algorithm, and (2) the use of a long term memory mechanism to keep high quality routes encountered in elite solutions. We conduct extensive computational experiments to investigate the contribution of these components, and measure the performance of our method on literature benchmarks. Our approach outperforms state-of-the-art algorithms in terms of overall solution quality and computational time. It finds the current best known solutions, or better ones, for 93% of the literature instances within reasonable runtimes. Moreover, it is able to achieve better average deviation than state-of-the-art algorithms within shorter computation times. Plus, new improvements for 61 benchmark instances were found.

2.1 Introduction

In the Team Orienteering Problem (TOP), we are given a transportation network in which a starting and an ending point are specified. The network connects a set of points that

correspond to customer locations. Each one of them is associated with a profit and a service time. For each pair of locations, a travel time is specified. The aim of the problem is to find a fixed number of disjoint paths from the starting point to the final destination through a subset of locations, each not exceeding a given time limit, that maximize the total profit collected from visiting customers.

In this chapter, we consider the Team Orienteering Problem with Time Windows (TOPTW), a natural extension of the TOP motivated by different practical situations. Possible applications of the problem range from logistics Golden et al. [121], Tang and Miller-Hooks [233] to leisure related applications like tourism Vansteenwegen et al. [255]. In the TOPTW, each customer must be visited within a predefined time interval, specified by an earliest and a latest time, into which the service must start. We assume that the time windows are *hard* constraints. This means that early arrivals to a location are permitted, but the agent must wait for it to be "open" before the service can start. Late arrivals, however, are not allowed.

Herein, we propose an effective approach that follows the basic structure of a Multi-Start Iterated Local Search (MS-ILS) to solve the TOPTW. We design a rather straightforward method that is able to effectively explore the search space to achieve high-quality solutions within very short computation times.

- First, we investigate the alternation between two search spaces: a *giant tour search space* and a *route search space*. In the *route space*, solutions are represented as genuine TOPTW solutions i.e., a set of feasible routes, one for each vehicle in use, while in the *giant tour space*, they are represented with an ordered list of customers with no route delimiters. The transition from the *giant tour search space* to the *route search space* is achieved using a powerful split algorithm. This idea is a follow up on preliminary work carried out by Guibadj and Moukrim [124]. They proposed a Memetic Algorithm chapterTW, which uses a *giant tour representation* for encoding individuals. MS-ILS, however, uses the giant tour representation more efficiently within an algorithm that is conceptually simpler and much faster than the MA.

- Second, we integrate an adaptive memory mechanism to overcome the drawbacks due to pure multi-start heuristics being memoryless. The adaptive memory is used to store individual routes extracted from diverse high quality solutions. These routes are then combined to construct promising new starting solutions at each iteration of the multi-start algorithm.

- Third, we conduct extensive computational experiments to investigate the contribution of these components to the search performance of a basic MS-ILS, and measure the performance of our approach on literature benchmarks. The obtained results show that our MS-ILS outperforms state-of-the-art algorithms in terms of solution quality and computation time. It is able to find the current best-known solutions, or better ones, for 94% of the available benchmark instances. It achieves an overall average relative gap of 0.17% and 0.14% on the two benchmarks of the literature, respectively, while being faster than most state-of-the-art algorithms. Additionally, it was able to improve the solutions of 61 instances for which no optimal solutions have yet been found. In comparison, the previous best performing approach in the literature finds 65% of the current best-known solutions, and achieves a relative gap of 0.80% and 0.34% respectively, on the two benchmarks.

- Finally, we show that our algorithm can be tuned to either favor solution quality at the cost of more computational effort, or to considerably reduce computation times while maintaining good solution quality. As such, it can serve as a good basis for future developments on more complex variants of selective vehicle routing problems.

The remainder of this chapter is organized as follows. Section 2.2 provides a brief overview of the literature related to the TOPTW, and Section 2.3 gives a formal description of the problem. Section 2.4 gives a detailed description of our proposed algorithm. First, the general framework of the method is introduced, and then each of its aspects is described in detail, including the solution representation, the optimal split procedure, and other components and parameters. The effectiveness of our approach is shown in Section 2.5. Finally, Section 2.6 concludes this chapter and provides possible directions for future research.

2.2 Literature review

Orienteering problems are well-known NP-hard optimization problems, and solving them within a reasonable amount of time may prove to be rather difficult. There are relatively few studies that deal with exact methods for the OPTW and TOPTW. Righini and Salani [209] solve the OPTW using a dynamic programming (DP) algorithm with *Decremental State Space Relaxation* (DSSR) Giovanni and Matteo [119], a relaxation of the problem so that customers can be visited more than once. The solution process consists in solving the DSSR, and then incrementally tightening the relaxed problem by disallowing multiple

visits to a critical customer set of increasing size. Duque et al. [90] extend the pulse algorithm Lozano et al. [173] to solve the OPTW. The algorithm is presented as a general-purpose framework for hard shortest path problems, and includes two novel pruning strategies to discard sub-optimal and infeasible solutions. The proposed algorithm performs better than the dynamic programming approach of Righini and Salani [209]. The authors, however, only reported results on a subset of the benchmark instances. Tae and Kim [229] presented the first Branch-&-Price (B&P) algorithm for solving the TOPTW. Recently, Gedik et al. [108] introduced an exact approach for the TOPTW based on a constraint programming (CP) formulation of the problem. The CP-optimizer is able to solve several benchmark instances from Montemanni and Gambardella [180] and Vansteenwegen et al. [255] and provides good upper bounds for the more difficult ones.

Most of the research on orienteering problems with time windows focuses on the design of heuristic approaches. The fact that profits and travel times are independent, and that a good solution with respect to one criterion is often unsatisfactory with respect to the other, make it more difficult to devise consistently good heuristics to solve orienteering problems, despite them being seemingly simple Gendreau et al. [111]. The presence of time window constraints increases this difficulty since it becomes harder to pinpoint the customers that should be included in the solution.

Montemanni and Gambardella [180] proposed a heuristic method to solve the OPTW and the TOPTW using an Ant Colony System (ACS) algorithm. The ACS was later improved by Montemanni et al. [181] who identified some of its drawbacks and provided ways to overcome them by only considering the best solution found during the construction phase, and only applying the local search procedure on solutions on which it has not yet been applied. Vansteenwegen et al. [255] introduced a fast and simple Iterated Local Search (ILS) for the problem. Their algorithm was made fast and simple by only keeping the *insert* and *shake* steps. The authors also introduced a new data set with more difficult instances constructed from the instances of Solomon [221] and Cordeau et al. [59]. Tricoire et al. [251] defined the Multi-Period Orienteering Problem with Multiple Time Windows (MuPOPTW) as a means for scheduling sale representatives to visit customers. The MuPOPTW is a generalization of OPTW and TOPTW where visits to customers span a period of time (days), and where multiple time windows are allowed. As such, each customer may be visited on a different day, and may have several time windows for each given day. To solve the MuPOPTW, the authors designed a Variable Neighborhood Search (VNS) algorithm, which also provides good solutions for the OPTW and the TOPTW. However, their algorithm requires relatively large computation times.

An algorithm that combines a Greedy Randomized Adaptive Search Procedure (GRASP) and an Evolutionary Local Search algorithm (ELS) was introduced by Labadie et al. [153]. Their method uses simple constructive heuristics inside the GRASP to generate distinct initial solutions, which are then improved using the ELS algorithm. In another study, Labadie et al. [152] applied the concept of granularity to a VNS approach in order to reduce the size of the local search neighborhoods to moves that are more likely to lead to good quality solutions. Using the dual optimal solutions of an LP-problem, they partition the arc set into intervals of granularity from which they choose those with the most promising arcs. Two Simulated Annealing based algorithms for the TOPTW were presented by Lin and Yu [171]. The first one is a Fast Simulated Annealing (FSA) that is directed towards applications that require quick responses, while the second one is a Slow Simulated Annealing (SSA) that focuses on solution quality. Guibadj and Moukrim [124] designed a Memetic Algorithm (MA) that uses the *route-first cluster-second* approach to solve the TOPTW. Solutions are represented as permutations of a subset of reachable customers called *giant tours*, and a splitting procedure is applied to extract the optimal set of feasible routes with respect to the order of visits in the giant tour. Souffriau et al. [224] introduced the Multi-Constraint Team Orienteering Problem with Multiple Time Windows (MC-TOP-MTW), a variant of the TOPTW that includes additional knapsack constraints to limit node selection. Such constraints can be used, for example, to model entrance fees in the case of tourist trip planning applications. The authors proposed a hybridization of GRASP and ILS, and evaluated the performance of their algorithm on benchmarks for the related problems, including the TOPTW. An iterative framework comprising three components, namely I3CH, was proposed by Hu and Lim [135]. The first two components are a Local Search (LS) and a Simulated Annealing (SA) that are used to explore the solution space and to discover a set of routes that are stored in a pool. The last component recombines the stored routes using a Set Packing formulation to produce good quality solutions. Cura [72] solves the TOPTW using an Artificial Bee Colony (ABC) approach that mimics the foraging behavior of honey bees. He introduced a new food source acceptance criterion based on SA and a new scout bee search behavior based on a local search procedure. Gunawan et al. [125] introduced another ILS algorithm to solve the OPTW. The algorithm generates an initial solution using a greedy insertion heuristic that chooses the customers to be inserted based on Roulette-Wheel selection. The initial solution is then improved using an ILS that uses different local search operators and a combination of acceptance criteria and perturbation mechanisms to balance between diversification and intensification of the search. The ILS was later extended with the inclusion of more local search components by Gunawan et al. [127].

The latter also introduces, for the TOPTW, a more general mathematical model that can accommodate heterogeneous fleets and routes with different starting and ending points. A hybridization that embeds an ILS into a SA algorithm was presented by Gunawan et al. [126] to address the drawback that is the early termination of the ILS. Schmid and Ehmke [217] proposed an"Effective Large Neighborhood Search" (ELNS) approach to solve the TOPTW. The algorithm starts by generating an initial solution. During each iteration of the algorithm, the solution is destroyed and repaired through operators specifically designed for the TOPTW. To the extent of our knowledge, the ELNS is currently the approach that provides the largest proportion of best-known solutions for the TOPTW benchmark instances.

Finally, for a more detailed overview of the literature about orienteering problems, we invite the reader to refer to Vansteenwegen et al. [257] and Gunawan et al. [128] who provide reviews on several relevant variants of the orienteering problem, including the TOPTW, and discuss applications and solution methods, both exact and heuristic, for OP variants.

2.3 Problem definition

The TOPTW is defined on a complete graph $G = (V, A)$, where $V = \{0, 1, 2, ..., n\}$ is the vertex set, and $A = \{(i, j) : i \neq j,\ i, j \in V\}$ the arc set. Vertex 0 represents the depot, which corresponds to the starting and ending points, and each vertex $i \in V \backslash \{0\}$ represents a customer associated with a non-negative profit (score) P_i, a non-negative service time σ_i, and a predefined time window $[e_i, l_i]$. The profit of each customer $i \in V \setminus \{0\}$ can be collected at most once by a vehicle within its associated time window, i.e., a vehicle cannot visit the customer i if it arrives later than l_i, and in the event that it arrives earlier than e_i, it has to wait until e_i before the service can start. A time window $[e_0, l_0]$ is associated with the depot, where $e_0 = 0$ refers to the earliest departure time and l_0 refers to the latest possible arrival time at the depot. A non-negative travel time $c_{i,j}$ is associated with each arc $(i, j) \in A$. Travel times are assumed to satisfy the triangle inequality.

A fleet of m identical vehicles is available at the depot, and each of them performs at most one route. A route is an ordered subset of q customers that are visited by the same vehicle. Routes must start and end at the depot. We note the total profit collected during a route r, $P(r) = \sum_{i=1}^{i=q} P_{r[i]}$, and its total duration (travel cost) $C(r) = c_{0,r[1]} + \sum_{i=2}^{i=q} (c_{r[i-1],r[i]} + W^r_{r[i]} + \sigma_{r[i]}) + c_{r[q],0}$, where $r[i]$ denotes the i^{th} customer of the route and

W_k^r the vehicle's waiting time at customer k. Note that this travel cost corresponds to the arrival time at the depot. The total duration of each route is constrained within a predefined time limit L_{max}. We assume that L_{max} equals the latest possible arrival time at the depot, i.e., $L_{max} = l_0$. A route is considered feasible if and only if $C(r) \leq L_{max}$ and if each customer is visited within its time window. Thus, a feasible TOPTW *solution* S consists of at most m feasible routes in which each customer is visited at most once. The objective is to find a solution S that maximizes the total collected profit, i.e., that maximizes $\sum_{r \in S} P(r)$. For mixed integer linear programming formulations of TOPTW (see [180, 255]).

2.4 Solution approach

2.4.1 Method overview

In this work, we propose a randomized Multi-Start Iterated Local Search procedure (MS-ILS) enhanced by an adaptive memory mechanism. Originally, MS procedures are simple memoryless algorithms that sample the solution space by applying a local or neighborhood search from multiple randomly generated initial solutions. If an initial solution falls inside the attraction basin of a global optimum, the local search will pull it to this global optimum. Due to their simplicity, more often than not, MS procedures alone have difficulties to compete with more aggressive meta-heuristics and must be strengthened by complementary diversification techniques to help surmount local optima.

Our approach employs an Iterated Local Search (ILS) procedure as the local search step of the MS metaheuristic. The ILS is a simple metaheuristic, the principle of which is to build a sequence of improved local optima to explore the search space. More specifically, starting from a solution S, at each iteration, the ILS generates a new solution S' by perturbation of S, then improves it using a local search to obtain S''. If S'' satisfies an acceptance criterion, it becomes starting solution for the next iteration; otherwise the algorithm goes back to S.

The efficiency of our algorithm stems from two key aspects. The first one is to alternate between two search spaces, each one using a different solution representation. This idea has proven to be very effective for various vehicle routing problems [200, 203], including the Team Orienteering Problem (TOP) [34, 77]. In our case, the two solution representations are the *route representation* and the *giant tour representation*. The *route representation*

is a genuine TOPTW solution, i.e., a set of feasible routes, one for each vehicle in use. To ensure fast feasibility checks upon insertion of a customer, we record, for each route, additional information as regards the arrival time, the waiting time and the maximum delay allowed for the service at each customer. On the other hand, the *giant tour representation* consists of an ordered list T of all the accessible customers in V with no route delimiters. It is a permutation $T = (T[1], T[2], ..., T[n])$ of n customers that ignores route length constraints and time windows. The interesting part about this indirect solution representation is (1) the fact that one giant tour corresponds to multiple TOPTW solutions, and (2) that it is possible to retrieve the optimal solution with respect to the order of customers in polynomial time using a *splitting* procedure. Thus, it is possible to search the space of giant tours instead of the TOPTW solution set, without loss of information. The reverse transformation from the *route representation* to the *giant tour representation* is achieved through a simple concatenation procedure.

The second key component is the integration of a long-term memory mechanism in order to improve the performance of the algorithm by learning from local optima found during previous iterations. One important drawback of a pure multi-start procedure is being a memoryless method: each iteration is independent of the previous one and no information about the solutions is passed from one iteration to the other. In our approach, we integrate an adaptive memory mechanism to retain information about interesting customer sequences observed in high-quality solutions. The use of a memory mechanism is inspired from the probabilistic tabu search of Rochat and Taillard [210]. It consists in storing routes extracted from high-quality solutions inside a pool of elite but diverse routes, and then using them to generate new starting solution. In our approach, the stored routes are used to construct giant tours using a probabilistic function that is biased towards the selection of elements that appear more frequently in high-quality solutions.

Algorithm 1 describes the general structure of our MS-ILS approach. At the beginning, the adaptive memory is initialized using the iterative heuristic described in Section 2.4.2. During the process, S_{best} is set to the best solution found so far.

At each iteration of the main loop, the algorithm constructs a giant tour T using the routes stored inside the adaptive memory and extracts the solution S associated with tour T using the *Split* procedure described in Section 2.4.5. It then performs an ILS (ILS loop) with T as the starting point. The ILS loop is executed until the maximum number of iterations $iter_{ils}$ is reached. At each iteration of the ILS loop, T is perturbed by applying a random rotation, and a new giant tour T' is derived. The associated solution S' is then extracted from T', improved by a local search procedure, and the obtained routes are

Algorithm 1: MS-ILS algorithm for the TOPTW

Data: TOPTW instance
Result: S_{best} solution for the TOPTW instance

```
1  begin
2      i ← 0
3      S_best ← initializeMemory()                          // Algorithm 2
4      repeat                                               // Multi-start loop
5          T ← constructGiantTour(M)
6          S ← split(T)
7          for j ← 1 to iter_ils do                         // ILS loop
8              T' ← randomRotation(T)                       // Perturbation
9              S' ← split(T')
10             S'' ← localSearch(S')
11             updateMemory(M, S'')                         // see Section 2.4.3
12             if f(S'') ≥ f(S) then
13                 T ← concat(S'')
14                 S ← S''
15         if f(S) ≥ f(S_best) then S_best ← S
16         if new routes have been added into M then        // see Section 2.4.3
17             iter ← 0
18         else
19             iter ← iter + 1
20     until iter = iter_max
21     return S_best
```

inserted into the adaptive memory if they are of good enough quality; otherwise they are ignored. If S' is better than S, it is concatenated into a new giant tour which replaces T in the next ILS iteration, and S is updated. After the ILS loop, S_{best} is updated and the number of main loop iterations *iter* is reset to 0. Otherwise, *iter* is incremented and the main loop restarts. The algorithms stops when it fails to insert new routes in the adaptative memory during $iter_{max}$ consecutive iterations.

2.4.2 Memory initialization

The initial set of routes that compose the adaptive memory is generated using the fast heuristic procedure described in Algorithm 2. The proposed initialization heuristic starts by building a feasible solution using a Best Insertion Algorithm (BIA) to insert customers. The implemented BIA constructs a feasible solution by successively inserting customers in their best possible position. At each iteration, the procedure evaluates all the feasible insertions of unrouted customers and selects the insertion with the minimum cost. To

ensure that the feasibility of an insertion is checked in $O(1)$, we record, for each customer i included in a route r, its waiting time W_i^r and the maximum duration by which the service can be delayed $MaxShift_i^r$. All feasible insertions of each non-served customer u between two adjacent customers i and j are evaluated. The insertion of u is considered feasible if and only if it delays the visit of j by $Shift_u = (c_{i,u} + W_u^r + \sigma_u + c_{u,j} - c_{i,j})$ less or equal to $MaxShift_j^r$. The best insertion is evaluated using the following cost criterion: $cost(u) = Shift_u/(P_u)^\alpha$, where α is a control parameter, the value of which is randomly generated each time the BIA is used. The aim of $cost(u)$ is to favor the insertion of customers which maximize the gain in profit while limiting the increase in travel times. The computational method used for the generation of α is detailed in Section 2.5.2.

Algorithm 2: Memory initialization algorithm for the TOPTW.

Data: S empty solution
V vector of n customers
Result: S_{best} best solution found
$\mathcal{M}$ Adaptive memory containing a set of routes

```
1  begin
2      applyBestInsertion(S, V)
3      iter ← 0
4      start ← 0
5      l ← 1
6      while iter < iter_init do
7          applyDestructionOnEachRoute(S, start, l)
8          U ← getUnroutedCustomers(V, S)
9          applyBestInsertion(S, U)
10         updateMemory(M, S)                    // see Section 2.4.3
11         if f(S) ≥ f(S_best) then
12             S_best ← S
13             iter ← 0
14             l ← 0
15         else
16             iter ← iter + 1
17         start ← start + l
18         l ← l + 1
19         if l ≥ size(r_max)/2 then l ← 1
20     return S_best
```

At each iteration of the initialization heuristic, a limited number of customers is removed from the solution, which is then rebuilt by inserting unrouted customers using the BIA. The destruction consists in the removal of a sequence of consecutive customers from each route. These sequences are identified by a starting position *start* and a length of l customers. The length of the removed sequences l is first set to 1, and at the end of

each iteration, *start* is moved by l positions and l is incremented by one. If the solution is improved after repair, l is reset to its initial value. Destruction is applied in a circular manner, i.e., when the end of a tour is reached, the removal continues from the beginning of the tour. In order to avoid l from becoming too big and causing a large part of the solution to be destroyed at each iteration, it is reset to 1 each time its value reaches half the size of the largest route (r_{max}) in the solution, in terms of the number of visited customers. The destruction and construction phases are repeated until the algorithm reaches $iter_{init}$ iterations without improving the best solution.

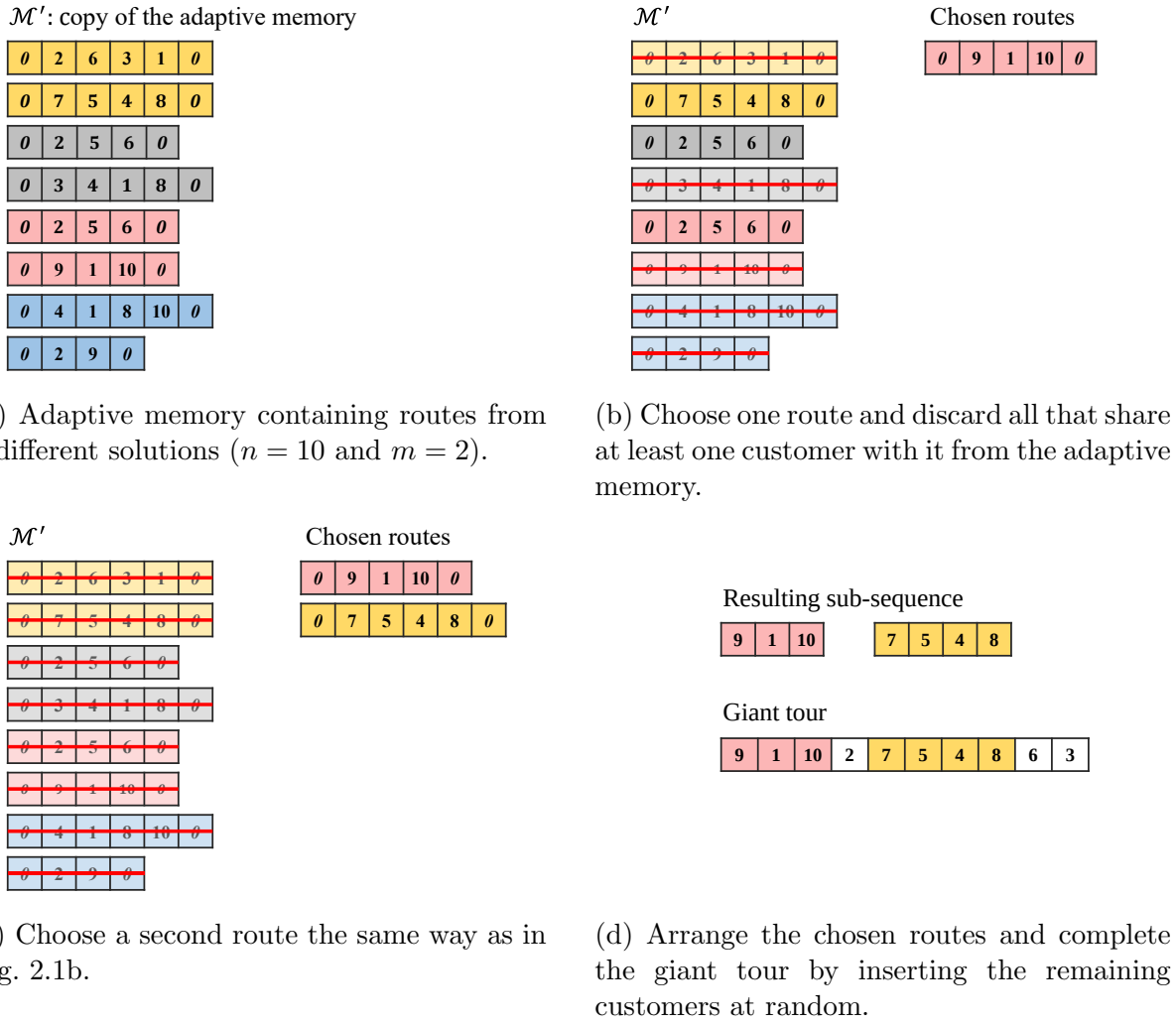
2.4.3 Memory update

For performance reasons, we limit the size of the adaptive memory to $\mathcal{M}_{size}$. When it becomes full, we need to remove some routes to be able to insert new ones. In order to keep promising routes, we apply the following strategy. Before its insertion, each route is labeled with the total collected profit and the total duration of the solution to which it belongs. The adaptive memory is then sorted in decreasing order of profits using the duration to resolve any equality. When the adaptive memory is full, the route to be inserted is first compared to the weakest route of the memory. If the new route's label is weaker, i.e. it has a lower total collected profit, the route is ignored. Otherwise, it is added to the memory and the weakest route is deleted. The idea is that routes that belong to solutions with higher scores are more likely to contain elements of optimal or sub-optimal solutions. To speed up the initialization of the adaptive memory, the routes of each solution generated by the BIA are considered for inclusion in the memory, so the heuristic is only used once.

2.4.4 Giant tour construction

The construction of giant tours plays an important role in our algorithm since it allows us to use information collected through past iterations for diversification and intensification purposes. To construct a giant tour, we select m routes from the adaptive memory and combine them as follows. Let $\mathcal{M}'$ be a copy of the current memory. First, we extract a route r from $\mathcal{M}'$ in a probabilistic way, and discard from $\mathcal{M}'$ all the routes r' that share at least one common customer with r. This process is then repeated until m routes are extracted, or until $\mathcal{M}'$ becomes empty. When selecting routes, we favor those extracted from solutions with higher quality. To that effect, we use the roulette wheel selection mechanism. After

that, the depot is removed from each of the selected routes. The resulting routes are then concatenated into a random order to form a partial giant tour, which we complete by randomly spreading out the unrouted customers over the tour. These unrouted customers are randomly inserted at the beginning of the tour, at the end, or between the routes in such a way that the m selected routes remain untouched, as can be seen in the example depicted in Fig. 2.1.



(a) Adaptive memory containing routes from 4 different solutions ($n = 10$ and $m = 2$).

(b) Choose one route and discard all that share at least one customer with it from the adaptive memory.

(c) Choose a second route the same way as in Fig. 2.1b.

(d) Arrange the chosen routes and complete the giant tour by inserting the remaining customers at random.

Figure 2.1 – Illustration of the giant tour construction process.

2.4.5 Splitting algorithm

In this section, we present the *optimal splitting algorithm* used by our MS-ILS to extract a TOPTW solution from a *giant tour*. The split consists of extracting m sub-sequences that do not share customers, and that respect route constraints and maximize the total collected profit from the *giant tour* T.

In their work, Bouly et al. [34] proved that solutions containing only *saturated routes* are dominant. Hence, it is unnecessary to enumerate all the feasible routes in T to be able to find the m ones that maximize the total profit; only *saturated routes* are needed. Starting from position i in T, the associated *saturated route* is obtained by including all the subsequent customers as long as all the constraints are satisfied, or until the end of the tour is reached.

Dang et al. [77] presented an evaluation procedure that efficiently uses the limited number of saturated routes to reduce the complexity of the splitting process. They reduce the splitting problem to a knapsack problem with conflicts (KPC) [231], and use a dynamic programming algorithm to determine the optimal splitting in polynomial time. In this paper, we extend this splitting procedure to tackle time window constraints.

2.4.5.1 Extraction of saturated routes

Let $R_s = \{\rho_1, \rho_2, ..., \rho_n\}$ denote the set of the n possible saturated routes extracted from the giant tour T. When extracting these routes, it must be ensured that each customer is visited within its time window and that the route is completed within the given time limit L_{max}, i.e., that $C(\rho_i) \leq L_{max}\ \forall i \in \{1, ..., n\}$. Starting from customer $T[i]$, we initialize the route $\rho_i = (0, T[i], 0)$. We then extend it by consecutively including the following customers $T[j], j \geq i$ as long as they are visited within their time windows. If the vehicle arrives at a customer $T[j]$ before the beginning of its time window, a waiting time is added. If the vehicle arrives too late at customer $T[j]$, if the inclusion of $T[j]$ violates the time limit constraint, or if the process reaches the end of the giant tour, the extension of ρ_i is stopped and the route is considered saturated. In the split algorithm for the basic TOP, all the saturated routes can be extracted in $O(n)$ because the cost of any sub-sequence $(i+1, ..., j, j+1)$ can be deduced in $O(1)$ from that of $(i, ..., j)$. However, this property is not satisfied by TOPTW sub-sequences. In our best implementation, route extraction runs in $O(n^2)$.

2.4.5.2 Selection of saturated routes

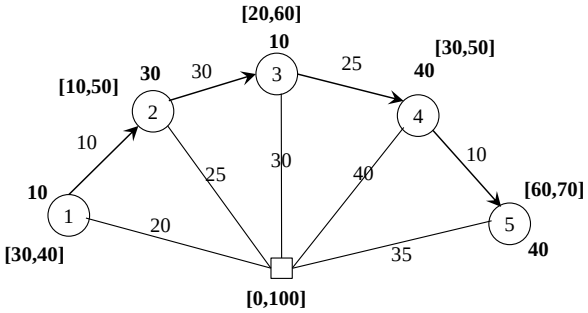
Once all the saturated routes have been extracted from T, the problem of selecting m routes that do not have common customers and that maximize the collected profit can be formulated as follows. Let $\mathcal{I} = \{1, 2, ..., n\}$ be the index set of all the saturated routes in previously defined set R_s. Set $\mathcal{I}$ can be seen as a set of n items where each item i

is associated with a unitary weight $w_i = 1$ and a value equal to the profit $P(\rho_i)$. Two items i and j are considered to be *in conflict* if the corresponding routes ρ_i and ρ_j include shared customers. Hence, the splitting problem can be formulated as a *Knapsack Problem with Conflicts* (KPC) where the objective is to select a subset $\mathcal{I}' \subset \mathcal{I}$ of items that maximizes the total value and such that the total weight of the selected items does not exceed the knapsack capacity m, and no selected items are in conflict with one another. In particular, our splitting problem is formulated as a Knapsack Problem with Interval Conflict Graph (KPICG). In short, a graph $H = (X, U)$ is an interval graph if there is a mapping I between the vertex set X and a collection of intervals in the real line such that two vertices in X are adjacent if and only if their respective intervals intersect. Then, for all i and j in X, $(i, j) \in U$ if and only if $I(i) \cap I(j) \neq \emptyset$ [242]. A giant tour T in our problem matches the real line, and each saturated route ρ_i extracted from T coincides with an interval $[i, j]$ where j is the last node of the route.

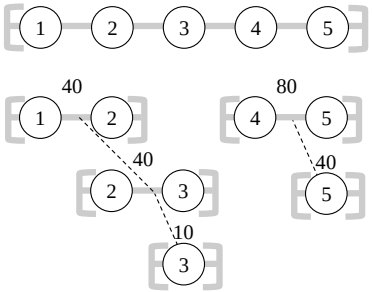
To solve the KPICG, we use the algorithm proposed by Sadykov and Vanderbeck [213]. They devised a pseudo-polynomial algorithm to solve a KPICG, defined with n items and a knapsack capacity equal to W, in $O(n \cdot W)$. Hence, the following result emerges.

Proposition 1. *Given a TOPTW instance with m available vehicles and n saturated tours extracted from a giant tour T, the splitting of T can be done optimally in $O(m \cdot n)$ time and space.*

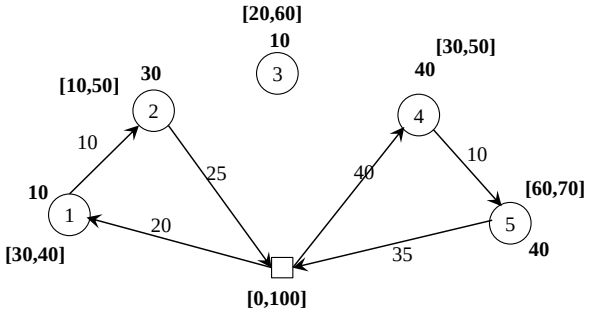
Figure 2.2 illustrates the splitting process. The graph in Fig. 2.2a shows a giant tour $T = (1, 2, 3, 4, 5)$ where each customer has a profit and a time window given in the square brackets. The number of available vehicles is $m = 2$, and the maximum operation time L_{max} is 100. For more simplicity, we assume that the service times for each customer are all equal to 0. The interval model is given in Fig. 2.2b. The first interval $[1, 2]$ for example with value 40 corresponds to the collected profit of the trip $(0, 1, 2, 0)$. The vehicle leaves the depot at time 0, waits 10 units of time at node 1 before being able to serve it, and then leaves to serve node 2 at time 40. Customer 3 cannot be included in this trip, since its time window is already closed when the vehicle reaches it at time 70. The other intervals $[i, j]$ of the graph are similarly defined. The maximum score obtained with two vehicles is equal to 120. Finally, the optimal solution to the splitting process is shown in Fig. 2.2c. It is composed of two routes starting with customer 1 and 4, respectively.



(a) Splitting problem with $m = 2$.



(b) Weighted interval model.



(c) Saturated optimal solution.

Figure 2.2 – Example of the splitting procedure.

2.4.6 Local search

The local search (LS) procedure is a key component of our algorithm since it serves as an intensification mechanism to improve the quality of the solutions constructed at each iteration. As previously mentioned, our algorithm uses two different solution representations, each with its own characteristics. Hence, we divide the neighborhood operators that compose the local search procedure into two sets: the first set is applied on *giant tours*, while the second set is performed on *routes*.

2.4.6.1 Giant tour neighborhoods

The LS includes two neighborhoods for *giant tours*:

- *shift operator*: moves a randomly chosen customer from its current position in the *giant tour* to a different one;
- *swap operator*: chooses a random customer from the *giant tour* and exchanges its

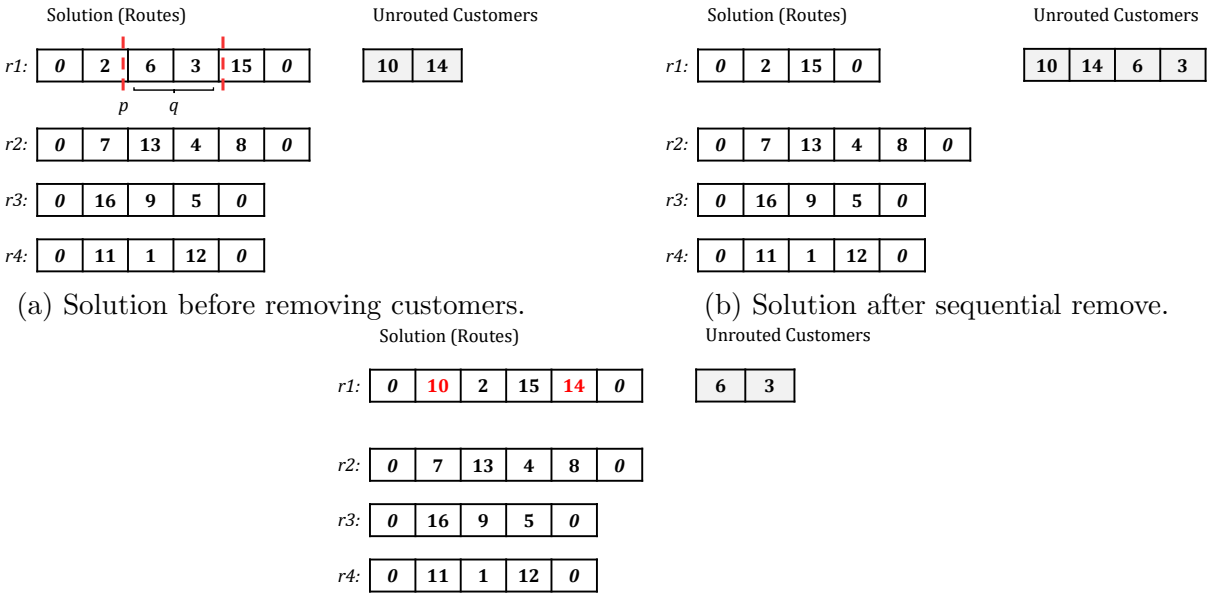
position with that of another randomly chosen customer;

Because the routes that compose a solution are not explicitly defined in the *giant tour representation*, the *shift* and *swap* operators do not need to check if feasibility is maintained, since every move is considered feasible in a giant tour representation. All that is needed to evaluate a move is to extract the new routes by splitting the giant tour. For performance purposes, we use the greedy splitting algorithm described in greater detail below.

2.4.6.2 Route neighborhoods

For *route* improvement, the LS procedure uses the classical routing neighborhoods *Or-Opt* and *2-Opt** to reduce the total travel time of each route, and three other neighborhoods to fill the resulting time saved in order to increase the total profit. The *2-Opt** operator replaces arcs $(i, i+1)$ from route u and $(j, j+1)$ from route v with arcs $(i, j+1)$ and $(j, i+1)$, thus interchanging the two sub-paths without altering the order of visits. The *Or-Opt* operator relocates a sequence of one or two successive customers in the same route, without altering their order of visit. The remaining operators follow the same *remove-and-repair* principle of the initialization heuristic described in Section 2.4.2 but differ in the way they select the customers to be removed and in how they repair the solution:

- *Random remove-and-repair*: removes d randomly chosen customers from the solution and, then, using the Best Insertion Algorithm (BIA), inserts currently unrouted customers. At each iteration, d is randomly generated in the interval $[1, D_{max}]$.
- *Sequential remove-and-repair*: this operator focuses on improving one single route at a time. This operator removes from each route r a sequence of q consecutive customers starting from position p mod $size(r)$ and uses BIA to insert new customers into the destroyed route. At each iteration, p is moved by q customers, and q is increased by one. When all the customers in a route have been removed at least once, it skips to the next route in the solution. This operator stops if an improvement is found, or once all the routes of the solution have been tested. Fig. 2.3 shows an iteration of this operator.
- *Parallel remove-and-repair*: Similar to the previous operator, this operator removes from each route r a sequence of q consecutive customers starting from position p mod $size(r)$. However, instead of focusing on one route, it is applied in parallel on



(a) Solution before removing customers.

(b) Solution after sequential remove.

(c) Solution after repair.

Figure 2.3 – Sequential remove-and-repair.

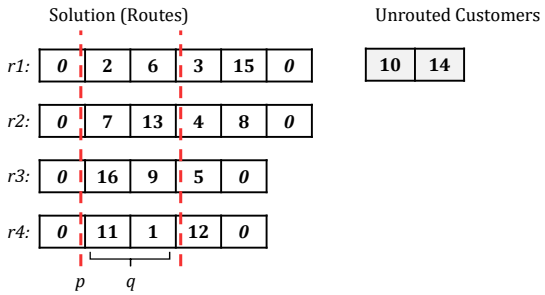
all the routes of the solution, and then inserts unrouted customers using the BIA. An example of this move is shown in Fig. 2.4. The search in this neighborhood stops when an improvement is found, or when each customer in the solution has been removed at least once. Removing parallel sequences from different routes creates free time slots across the whole solution and gives the opportunity to BIA to move customers between routes, and introduce more profitable ones in the solution.

2.4.6.3 Local search algorithm

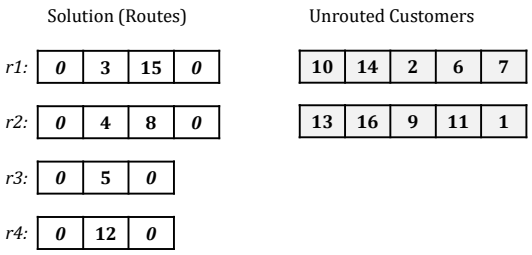
The local search procedure works as follows. At each iteration, the LS randomly selects one of the previous neighborhoods and explores it. All the neighborhoods have the same probability of being chosen whether they consider *routes* or *giant tours*. The search in a given neighborhood stops as soon as an improvement is found or when no improving move can be found. When a neighborhood operator fails to find an improvement, it is discarded until the solution is improved by another operator. This process is repeated until all neighborhoods fail to find an improvement to the current solution.

2.4.6.4 Greedy split and concatenation

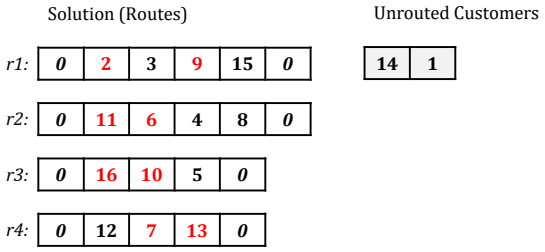
Because the LS procedure uses two different sets of neighborhoods, and because it can randomly switch from an operator of one set to that of the other, the number of times



(a) TOPTW solution.



(b) Solution after parallel remove.



(c) Solution after repair.

Figure 2.4 – Parallel remove-and-repair.

a solution shifts from one representation to the other can grow large very quickly. This negatively impacts computation times, mainly due to the optimal splitting procedure. To address this aspect of the LS, we devise a fast heuristic splitting method and a giant tour construction process to go with it.

Inside the LS algorithm, the construction of a *giant tour* T is achieved by concatenation of the routes that compose the solution one after the other, and then by appending the remaining unrouted customers at the end of the tour.

On the same principle as the above fast concatenation method, we describe a greedy splitting heuristic to evaluate a giant tour $T = (T[1], T[2], ..., T[n])$. Consider $\theta_k = (T[i], T[i+1], ..., T[n])$ a sub-sequence of T. It is possible to divide θ_k into two parts by extracting the saturated route $\rho_k = (T[i], T[i+1], ..., T[j])$ and considering the second part $\theta_{k+1} = (T[j+1], T[j+2], ..., T[n])$ as a new sequence of unrouted customers. Our greedy splitting heuristic consists in performing the previous bisection m times on consecutive sequences θ_k, starting with $\theta_1 = T$. Fig. 2.5 shows an example of splitting using the heuristic procedure. The heuristic splitting is used as a quick way to evaluate moves in the *swap* and *shift* neighborhoods. The optimal splitting algorithm is applied at the end of the LS procedure to extract the best routes from the giant tour.

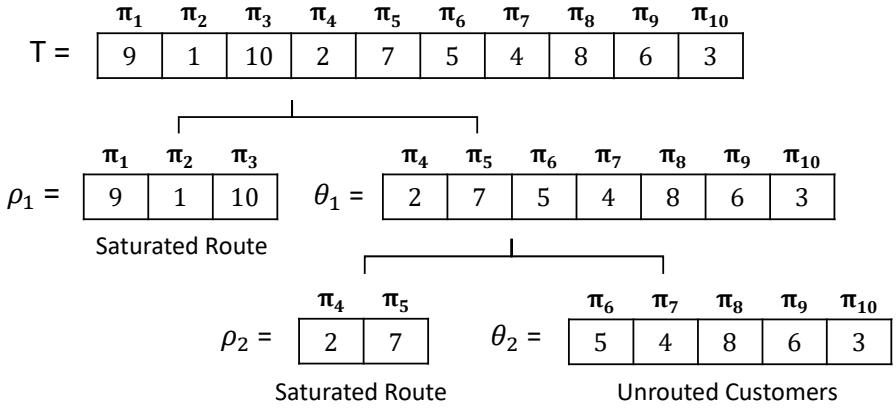


Figure 2.5 – Heuristic splitting procedure when $m = 2$.

2.5 Computational experiments

Our algorithm was coded in C++ using the Standard Template Library (STL) for data structures, and compiled using the GNU GCC compiler in a Linux environment. In the following, we present the computational experiments that were carried out in order to tune the algorithm, to evaluate its performance relative to state-of-the-art algorithms, and to assess the contribution of each of its components. All the experiments were conducted on a single thread on an Intel Xeon X7542 CPU at 2.67GHz processor.

2.5.1 Benchmark instances

Our algorithm was tested on the various TOPTW benchmark instances. All the literature benchmarks are available online at `http://www.mech.kuleuven.be/en/cib/`.

TOPTW benchmark instances are derived from Solomon et al.'s [221] instances for the Vehicle Routing Problem with Time Windows (VRPTW), and Cordeau's [59] instances for the Multi Depot Periodic VRPTW (MDPVRPTW). The instances were adapted by considering the customer demands in the original data sets as node profits in the TOPTW instances. Travel time between two vertices is assumed to be equal to the euclidean distance. It is rounded down to the first decimal for Solomon's instances and to the second decimal for Cordeau's instances. Solomon's instances are organized into six sets: C100, C200, R100, R200, RC100 and RC200, divided according to the distribution of vertices on the plane (clustered, random, random-clustered) and the width of the time windows (narrow, wide). Each instance contains either 50 or 100 customers. Since Cordeau's instances were originally meant for a periodic variant of the VRP, when using them for the TOPTW, we consider that all customers are available on the same day.

Instances in this data set contain a number of customers ranging from 48 to 288.

Righini and Salani [209] constructed the first OPTW instances using 29 of Solomon's instances from sets C100, R100 and RC100, and ten instances from Cordeau's (pr1 to pr10). Montemanni and Gambardella [180] proposed another 27 instances from sets C200, R200, and RC200 from Solomon's instances, and ten others from Cordeau's instances (pr11 to pr20). Additionally, they extended the earlier instances by increasing the number of vehicles m and varying it between one and four vehicles. Furthermore, Vansteenwegen et al. [255] used the original instances of Solomon and Cordeau to introduce new benchmark instances where the number of vehicles considered for each instance makes it possible to visit all the customers. Since it is possible to visit all customers, the optimal solution for these instances is known: it is equal to the sum of all customer profits. Like Hu and Lim [135], we refer to these instances as the "OPT" data set. As Vansteenwegen et al. [255] point out, algorithms to solve the TOPTW are generally not designed expecting the possibility of including all the customers, which makes solving some of the "OPT" instances to the optimum rather difficult.

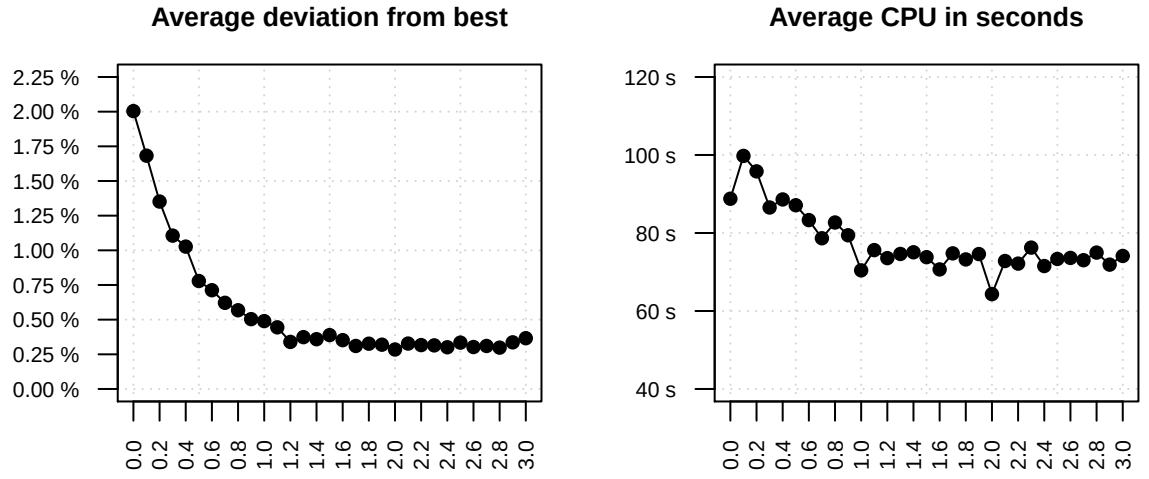
2.5.2 Parameter tuning

As described in Section 2.4, our proposed MS-ILS has five sets of parameters:

- α, the control parameter of the BIA;
- D_{max}, the maximum number of customers removed by the random removal operator;
- $\mathcal{M}_{size}$, the size of the adaptive memory;
- $iter_{init}$, the number of iterations of the initialization heuristic;
- $iter_{max}$ and $iter_{ils}$, the number of iterations of the main algorithm.

We proceeded first with finding the right value for α. To do this, we ran our algorithm on the previously mentioned instances while varying the value of α between 0 and 3 by increments of 0.1. The experiment was repeated ten times, each time with a different random seed. The remaining parameters of the algorithm were set to preliminary values: $D_{max} = 12$, $\mathcal{M}_{Size} = 60$, $iter_{ils} = 10$ and $iter_{max} = (20{*}n)/m$. We then computed, for each instance, the percentage deviation between the average solution value Z_{avg} and the best solution achieved over all the runs $MSILS$ as: $gap = (MSILS - Z_{avg})/MSILS * 100$.

The results of these experiments are summarized in Fig. 2.6, which shows the average deviation and average runtime over all the runs.

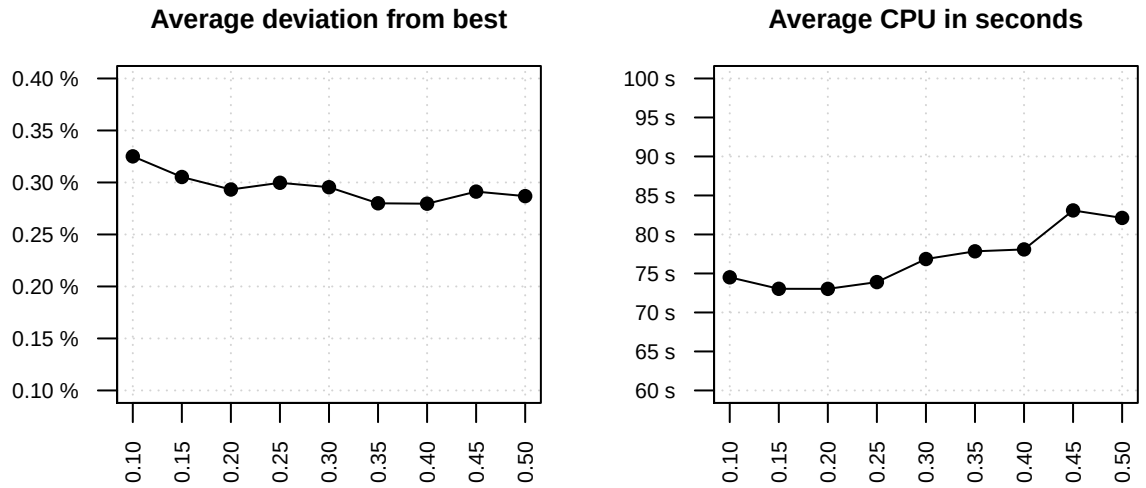


(a) Effect of α on solution quality. (b) Effect of α on CPU time.

Figure 2.6 – Tuning the value of α.

As can be seen on Fig. 2.6, solution quality improves as the value of α increases, and then starts slowly stabilizing when α becomes greater than one. This observation also holds for the average runtime: it decreases as α increases and becomes stable when $\alpha \geq 1$. The best performance is obtained when $\alpha = 2$. However, (1) the difference in performance of values between 1.8 and 2.8 is rather small, and (2) during our experiments, we observed that the value of α that performs better varies depending on the instance. Therefore, and in order to enable diversification during the construction process, we generate a random value in $[1.8, 2.8]$ for α at each iteration of the algorithm.

The experiments carried out to tune parameter D_{max} are similar to the previous ones. We invoked our MS-ILS on the test instances for different values of D_{max} and repeated it ten times, each time with a different random seed. For our experiments, we decided to express the value of D_{max} as a proportion of the number of customers, so that it would adapt to different-sized instances. We varied this value between $0.1 * n_{routed}$ and $0.5 * n_{routed}$ by increments of 0.05, where n_{routed} is the number of customers visited in the current solution. We bound the value of D_{max} by $0.5 * n_{routed}$, because we consider that removing more customers would alter the solution too much. We set α as described above and set the values of the remaining parameters to the preliminary values used before. We then compared the average deviation and the average runtime achieved by each value of D_{max} over the ten runs, like we did with α. Figure 2.7 summarizes the results of these experiments. We observe that the CPU time increases with the value of D_{max}. On the other hand, solution quality improves as D_{max} becomes larger. There are, however, only



(a) Effect of D_{max} on solution quality.

(b) Effect of D_{max} on CPU time.

Figure 2.7 – Tuning the value of D_{max}.

slight differences between the results of each value of D_{max} in terms of solution quality. Given these results, we opted to set D_{max} to $0.25 * n_{routed}$, as it offers a good compromise in terms of runtime and solution quality.

Finally, parameters $iter_{max}$ and $iter_{ils}$ control the computational time of the algorithm, while parameter $\mathcal{M}_{size}$ controls the size of the adaptive memory. All of these parameters influence both solution quality and search time, and by changing their values, we achieve different trade-offs in terms of solution quality and computational effort. Larger values of these parameters result in better solution quality, but significantly increase computation times. On the other hand, smaller values lead to a faster termination of the algorithm, however, at the expense of solution quality. Because of that, using meta-calibration or other automated tools for tuning our algorithm becomes difficult, as they will tend to select the maximum value for each parameter if we choose to maximize solution quality, and minimum values if we choose to minimize computation times. Hence, in order to calibrate these three parameters, we decided to build a Pareto test using different combinations of $iter_{max}$, $iter_{ils}$ and $\mathcal{M}_{size}$. The possible values of each parameter were set to: $\mathcal{M}_{size} \in \{40m, 50m, ..., 100m\}$, $iter_{ils} \in \{2, 3, 4, 5\}$, and $iter_{max} \in \{n/m, 2n/m, ..., 10n/m\}$. Afterwards, we chose the configuration that offers the best compromise between solution quality and computation time. The test results are shown in Fig. 2.8, where we plot for each combination the average deviation and the average runtime. We chose configuration $(iter_{max}, iter_{ils}, \mathcal{M}_{size}) = (2 * n/m,\ 4,\ 80 * m)$ because we estimated that the difference in ARPE compared to slower configurations was not enough compared to the difference in computation time.

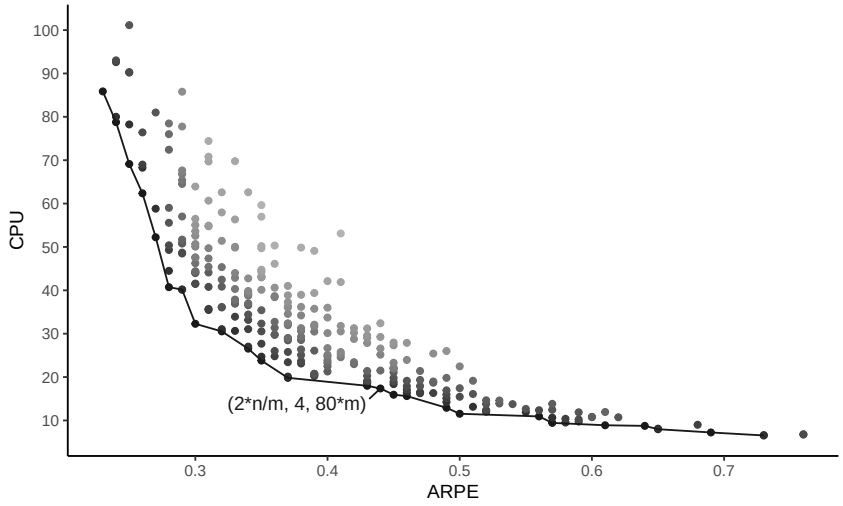


Figure 2.8 – Pareto front of results obtained with different combinations of ($iter_{max}$,$iter_{ils}$, $\mathcal{M}_{size}$).

The final calibration results are displayed in Table 2.1. Note that parameter $iter_{init}$ has little impact on the search process. Its value was chosen arbitrarily to allow the initialization heuristic to fill the adaptive memory with good solutions while keeping the initialization time very short.

Parameter	Description	Value
α	the control parameter of the BIA	$[1.8, 2.8]$
D_{max}	max. nb. of customers removed by the Random remove-and-repair operator	$0.25 * n_{routed}$
$\mathcal{M}_{size}$	size of the adaptive memory	$80 * m$
$iter_{init}$	nb. of iterations of the initialization heuristic	1000
$iter_{max}$	nb. of iterations of the main algorithm	$2 * n/m$
$iter_{ils}$	nb. of iterations of the ILS	4

Table 2.1 – Final parameter values.

2.5.3 Computational comparisons

In order to evaluate the performance of our proposed MS-ILS for TOPTW, we compared its results with those of the following state-of-the-art algorithms:

- Variable Neighborhood Search (VNS) proposed by Tricoire et al. [251],
- Greedy Randomized Adaptive Search procedure with Evolutionary Local Search algorithm (GRASP-ELS) of Labadie et al. [153],

- LP-Based Granular Variable Neighborhood Search (GVNS) of Labadie et al. [152],
- Iterative Three-Component Heuristic (I3CH) of Hu and Lim [135].
- Large Neighborhood Search (ELNS) of Schmid and Ehmke [217].

The results of GVNS, GRASP-ELS, I3CH, ELNS, and our MS-ILS were all obtained with five runs of the algorithm on each instance, while VNS was run ten times on each instance. Note that some results of VNS that were not originally reported by Tricoire et al. [251] were made available on the authors' website *http* : *//prolog.univie.ac.at/research/op/*. Since they are better results than those originally published, we used them instead for our comparisons.

For the sake of fair comparison between the algorithms in terms of computational effort, the reported running times of each algorithm were adjusted to account for the speed difference between the different computation setups. Similar to Hu and Lim [135], we used the *Super Pi* benchmark for this purpose. *Super Pi* is a single-threaded program that computes the digits of π up to a specified number and is commonly used as an estimate of CPU speed. Table 2.2 indicates the CPU used by each algorithm, its *Super Pi* score, which corresponds to the number of seconds needed to compute the first one million digits of π, and the associated time scaling factor. To obtain this factor, we estimated the performance of each processor by considering the performance of our machine to be 1. For GRASP-ELS, GVNS and I3CH, we opted to use the *Super Pi* scores reported by Hu and Lim [135]. Unfortunately, the scores of VNS and ELNS are not available. Furthermore, we only had limited information on the experimental setup used by each of them to be able to estimate their *Super Pi* scores. For the VNS, given that the paper was published in 2010, we decided to use the same scaling factor as the one used for the GVNS (0.32), because of the proximity of their publication dates. As for ELNS [217], the authors did not report comparisons based on CPU times, and we were not able to estimate their *Super Pi* score by only relying on the family of their CPU and its clock rate. Therefore, we assumed similar performance to our setup. In the remainder of this section, the time values reported in previous works are adjusted by the associated factors, in order to account for CPU differences. For example, the adjusted computational time of a solution obtained by I3CH can be obtained by multiplying its CPU time by 0.95.

Tables 2.3 to 2.5 summarize the results achieved by each algorithm. Comparison of solution quality is usually drawn in terms of relative percentage error (*rpe*) with respect to the best-known solution (BKS) for the standard benchmark, and in terms of average relative

Algorithm	CPU	Super Pi Estimate	Factor
VNS	2.4 GHz CPU (reference unknown)	Unknown	≤ 0.32
GRASP-ELS	Intel Pentium 4 processor, 3.00 GHz	44.3	0.32
GVNS	Intel Pentium (R) IV, 3 GHz CPU	44.3	0.32
I3CH	Intel Xeon E5430 CPU clocked at 2.66 GHz	14.7	0.95
ELNS	Intel Xeon 3.1 GHz (reference unknown)	Unknown	≈ 1
MS-ILS	Intel Xeon X7542 CPU at 2.67GHz	14.1	1

Table 2.2 – Estimation of single-thread performance.

percentage error (*arpe*) for the "OPT" benchmark, but we chose to include comparisons on the basis of RPE and ARPE for both benchmarks. These two metrics are computed as: $rpe = \frac{(BKS - Z_{max})}{BKS} * 100\%$ and $arpe = \frac{(BKS - Z_{avg})}{BKS} * 100\%$, where Z_{max} denotes the best score obtained over different runs and Z_{avg} the average score. Column cpu_{avg} of each table reports the average computational time of previous algorithms in seconds. The detailed results obtained by our MS-ILS are presented in Appendix A, where they are compared to the best-known solutions in the literature. These results are presented in tables, of which, each consists of two identically structured parts. Each part contains the name of the instance, the best-known solution (BKS) to the instance, including the ones found by our method, the maximum score (Z_{max}) obtained by our algorithm, the relative error (*rpe*), the average score (Z_{avg}), the average error (*arpe*), and the average computational time in seconds (cpu_{avg}).

Table 2.3 reports the results obtained by all the state-of-the-art algorithms and the MS-ILS using the previous configuration on the standard benchmark. As can be seen in the table, our MS-ILS achieves an average relative gap lower than those achieved by the other algorithms for every set of instances. As for the relative gap, it achieves better results than the literature on most instances, apart from instances $rc200$ with $m = 3$, and $rc100$ with $m = 4$ where ELNS does slightly better, but at the cost of higher computational effort. On Cordeau's instances, our method requires a little more computation time to find high-quality solutions compared to the likes of the GRASP-ELS and the GVNS, but is still much faster than I3CH and the VNS, and the quality of the solutions it obtains is much better than that of the other algorithms. Furthermore, note that Solomon's instances with wider time windows become easier to solve as m becomes larger.

The performance of our algorithm on the "OPT" data set is shown in Tables 2.4 and 2.5. The column $\#OPT$ indicates the number of optimal solutions found by the MS-ILS. Note that, in the case of column $\#OPT$, the value reported at the bottom of the two

Table 2.3 – Comparison of the MS-ILS to the state-of-the-art methods on the standard benchmark.

Instances	VNS			GRASP-ELS			GVNS			I3H			ELNS			MS-ILS		
	rpe%	*arpe*%	cpu	*rpe*%	*arpe*%	cpu	*rpe*%	*arpe*%	cpu	*rpe*%	*arpe*%	cpu	*rpe*%	*arpe*%	cpu	*rpe*%	*arpe*%	cpu
m=1																		
c100	**0.00**	0.11	31.5	**0.00**	**0.00**	7.2	0.56	1.22	53.3	**0.00**	-	24.0	**0.00**	**0.00**	19.0	**0.00**	**0.00**	2.0
r100	**0.00**	0.05	28.5	0.11	0.22	1.1	1.72	2.68	9.4	0.56	-	27.2	**0.00**	0.06	15.2	**0.00**	**0.02**	2.1
rc100	**0.00**	0.04	20.9	0.33	0.40	0.6	1.88	3.51	3.1	1.66	-	24.3	**0.00**	0.31	10.5	**0.00**	**0.02**	1.1
c200	**0.00**	0.21	179.3	0.40	0.61	10.3	0.55	1.11	61.6	0.40	-	80.2	**0.00**	0.08	47.0	**0.00**	**0.08**	9.9
r200	0.95	1.60	341.1	1.14	2.15	3.6	2.98	3.90	10.8	1.58	-	167.4	0.05	0.29	65.6	**0.03**	**0.10**	22.9
rc200	0.25	1.52	278.2	1.55	2.37	2.6	2.70	4.13	5.1	2.85	-	113.4	0.23	0.48	47.2	**0.01**	**0.15**	13.4
pr01-pr10	0.02	1.10	263.1	0.75	1.46	1.6	0.56	1.62	4.0	1.07	-	103.6	0.10	0.18	40.1	**0.02**	**0.18**	13.0
pr11-pr20	1.44	3.41	334.7	2.20	3.42	2.5	3.21	4.30	7.8	4.31	-	123.7	1.44	2.08	67.5	**0.85**	**1.15**	21.5
m=2																		
c100	**0.00**	0.27	28.2	**0.00**	0.07	22.7	0.47	0.72	44.7	**0.00**	-	24.0	0.17	0.20	30.6	**0.00**	**0.03**	3.6
r100	0.18	1.43	20.3	1.04	1.78	2.5	1.22	1.83	19.3	0.61	-	27.2	0.14	0.29	25.5	**0.00**	**0.04**	2.7
rc100	0.23	1.46	17.7	1.46	2.32	1.5	0.78	2.80	6.5	0.90	-	24.3	**0.00**	0.05	20.7	**0.00**	**0.05**	2.9
c200	0.59	0.95	174.6	0.17	0.34	9.4	0.34	0.66	10.8	0.76	-	80.2	**0.00**	0.20	63.5	**0.00**	**0.14**	16.8
r200	0.85	1.35	324.8	0.92	1.24	5.6	1.26	1.89	4.7	0.81	-	167.4	0.27	0.49	42.2	**0.08**	**0.26**	30.9
rc200	1.03	2.02	257.5	1.19	1.71	5.5	2.22	3.09	4.1	1.16	-	113.4	0.29	0.68	52.5	**0.15**	**0.38**	25.9
pr01-pr10	1.10	4.11	167.9	1.34	2.42	6.2	1.05	2.02	12.5	1.34	-	103.6	0.72	1.20	85.0	**0.10**	**0.43**	18.1
pr11-pr20	1.95	4.30	198.0	3.11	4.24	9.2	1.90	2.87	26.4	3.39	-	123.7	2.05	2.97	135.0	**0.84**	**1.29**	32.1
m=3																		
c100	0.11	0.73	27.4	0.24	0.56	27.8	0.45	0.95	52.8	0.11	-	82.6	0.11	0.49	40.5	**0.00**	**0.09**	5.1
r100	0.23	1.48	19.8	0.91	1.58	4.4	1.23	2.28	23.7	0.22	-	59.9	0.08	0.24	35.3	**0.00**	**0.06**	3.5
rc100	0.38	1.42	19.4	1.85	2.83	2.8	0.93	2.34	10.8	0.29	-	56.0	0.01	0.37	30.2	**0.00**	**0.25**	2.9
c200	0.29	1.01	63.0	0.58	0.92	8.6	0.77	1.29	17.7	0.16	-	381.2	0.22	0.42	27.7	**0.08**	**0.18**	5.2
r200	0.09	0.16	102.9	0.06	0.07	0.8	0.17	0.26	2.2	0.07	-	500.5	0.06	0.08	4.9	**0.03**	**0.06**	3.2
rc200	0.11	0.32	129.3	0.13	0.26	2.7	0.32	0.44	2.4	0.04	-	417.7	**0.06**	**0.11**	14.8	0.07	**0.11**	8.0
pr01-pr10	1.80	3.92	151.4	1.61	2.27	13.0	0.66	1.43	27.5	0.66	-	234.7	1.17	1.86	124.2	**0.23**	**0.63**	25.7
pr11-pr20	2.51	4.06	165.6	3.02	3.97	13.7	1.74	2.52	48.2	1.84	-	289.4	2.64	3.40	191.7	**0.48**	**0.94**	43.3
m=4																		
c100	0.38	1.34	26.2	0.79	1.12	27.1	1.14	1.72	42.6	0.20	-	180.7	0.77	1.26	48.5	**0.10**	**0.31**	7.6
r100	0.36	1.60	19.6	1.00	1.71	7.7	1.27	2.34	27.1	0.22	-	112.4	0.13	0.37	42.7	**0.06**	**0.17**	4.4
rc100	0.58	2.45	18.7	1.67	2.41	4.3	1.08	1.92	11.8	0.36	-	95.9	**0.10**	0.35	37.7	0.11	**0.33**	3.5
c200	**0.00**	**0.00**	33.5	**0.00**	**0.00**	0.0	**0.00**	**0.00**	0.2	**0.00**	-	158.2	**0.00**	0.00	0.0	**0.00**	**0.00**	0.1
r200	**0.00**	**0.00**	48.2	**0.00**	**0.00**	0.0	**0.00**	**0.00**	0.1	**0.00**	-	86.3	**0.00**	0.00	0.0	**0.00**	**0.00**	0.1
rc200	**0.00**	**0.00**	52.7	**0.00**	**0.00**	0.0	**0.00**	0.01	0.3	**0.00**	-	155.9	**0.00**	0.00	0.0	**0.00**	**0.00**	0.1
pr01-pr10	2.56	4.24	129.0	2.58	3.20	14.6	1.77	2.32	40.7	1.06	-	402.8	2.40	3.25	156.1	**0.35**	**0.77**	34.4
pr11-pr20	3.00	4.17	130.6	3.28	4.10	20.9	2.74	3.50	74.4	1.16	-	472.1	3.21	3.99	234.9	**0.44**	**1.03**	59.1
Average	0.66	1.59	118.9	1.04	1.55	7.5	1.18	1.93	20.8	0.87	-	156.7	0.51	0.80	54.9	**0.13**	**0.29**	13.3

tables corresponds to the sum of the values of the column. Note also that the authors of GRASP-ELS [153] and GVNS [152] only report the average value of solutions obtained over several runs. Most of the published metaheuristics reported their results using the average relative gap *arpe*, except for Hu and Lim [135] who used only the best relative gap *rpe*. For this reason, we split the comparisons into two: Table 2.4 displays comparisons based on the *arpe*, while Table 2.5 shows the comparisons based on the *rpe*. Even though the "OPT" data set is known to be difficult to solve, MS-ILS achieves both the smallest average relative gap and the second smallest relative gap, and is able to obtain 59 out of 66 optimal solutions in a reasonable amount of time. The I3CH is the only other algorithm that finds slightly better solutions than the MS-ILS, but it requires significantly higher computational times to do so. In some cases, it is up to twenty times slower than the MS-ILS.

Table 2.4 – Performance comparison based on *arpe* average for "OPT" data set.

Instances	Nb.	VNS			GRASP-ELS			GVNS			ELNS			MS-ILS		
		#OPT	*arpe*%	cpu	#OPT	*arpe*%	cpu	#OPT	*arpe*%	cpu	#OPT	*arpe*%	cpu	#OPT	*arpe*%	cpu
c100	9	**9**	0.02	6.4	-	**0.00**	0.4	-	0.47	2.5	**9**	**0.00**	1.5	**9**	**9**	0.5
r100	12	1	0.50	6.9	-	0.73	33.7	-	1.55	12.6	4	0.73	45.4	**5**	**0.47**	14.7
rc100	8	**4**	0.85	6.7	-	0.90	25.2	-	1.29	12.6	3	0.39	39.1	**4**	**0.37**	8.0
c200	8	-	-	-	-	0.00	0.0	-	0.00	0.2	**8**	**0.00**	0.0	**8**	**0.00**	0.1
r200	11	-	-	-	-	0.04	2.1	-	0.17	1.8	**11**	0.01	9.7	**11**	**0.00**	2.9
rc200	8	-	-	-	-	0.03	0.9	-	0.16	0.9	**8**	**0.00**	2.6	**8**	**0.00**	1.4
pr01-pr10	10	5	1.30	22.9	-	**0.92**	22.9	-	1.25	16.4	3	1.22	154.9	**5**	0.95	25.8
Avg./Total	66	19	0.67	10.7	-	0.37	12.2	-	0.70	6.7	46	0.34	36.2	**50**	**0.26**	7.6

Table 2.5 – Performance comparison based on *rpe* average for "OPT" data set.

Instances	Nb.	VNS			I3H			ELNS			MS-ILS		
		#OPT	*rpe*%	cpu	#OPT	*rpe*%	cpu	#OPT	*rpe*%	cpu	#OPT	*rpe*%	cpu
c100	9	**9**	**0.00**	6.4	**9**	**0.00**	45.2	**9**	**0.00**	1.5	**9**	**0.00**	0.5
r100	12	1	0.20	6.9	**8**	**0.07**	833.8	4	0.58	45.4	5	0.34	14.7
rc100	8	4	0.37	6.7	**8**	**0.00**	54.5	3	0.29	39.1	4	0.19	9.4
c200	8	-	-	-	**8**	**0.00**	0.6	**8**	**0.00**	0.0	**8**	**0.00**	0.1
r200	11	-	-	-	9	0.07	164.7	**11**	**0.00**	9.7	**11**	**0.00**	2.9
rc200	8	-	-	-	7	0.04	180.7	**8**	**0.00**	2.6	**8**	**0.00**	1.4
pr01-pr10	10	5	1.06	22.9	**6**	**0.78**	310.3	3	1.06	154.9	5	0.86	26.4
Avg./Total	66	19	0.41	10.7	**55**	**0.14**	227.1	46	0.27	36.2	50	0.20	7.9

For a more thorough comparison, we conducted additional experiments using other combinations of values for parameters $iter_{max}$, $iter_{ils}$ and $\mathcal{M}_{size}$ in order to achieve smaller and larger computation times than those presented above. We then compared the performance of every combination with those of the state-of-the-art algorithms, namely VNS, GRASP-ELS, GVNS, I3CH and ELNS in terms of computation time (CPU) and

in terms of either relative error(rpe) or average relative error ($arpe$). The combinations of values used for these experiments were chosen based on the results of the Pareto test described in Section 2.5.2. The results of these experiments are displayed in Figures 2.10 and 2.9, and Tables 2.6 and 2.7.

Figures 2.10 and 2.9 show the performance of the MS-ILS using different combinations of parameters compared to state-of-the-art algorithms; the first in terms of computation time and relative error (rpe), the second in terms of cpu and average relative error (arpe). As can be seen in both figures, the MS-ILS achieves better results compared to the remaining algorithms: not only does it achieve smaller deviations when computation times are equivalent, but each combination of parameter values we tested resulted in smaller deviations compared to the remaining algorithms.

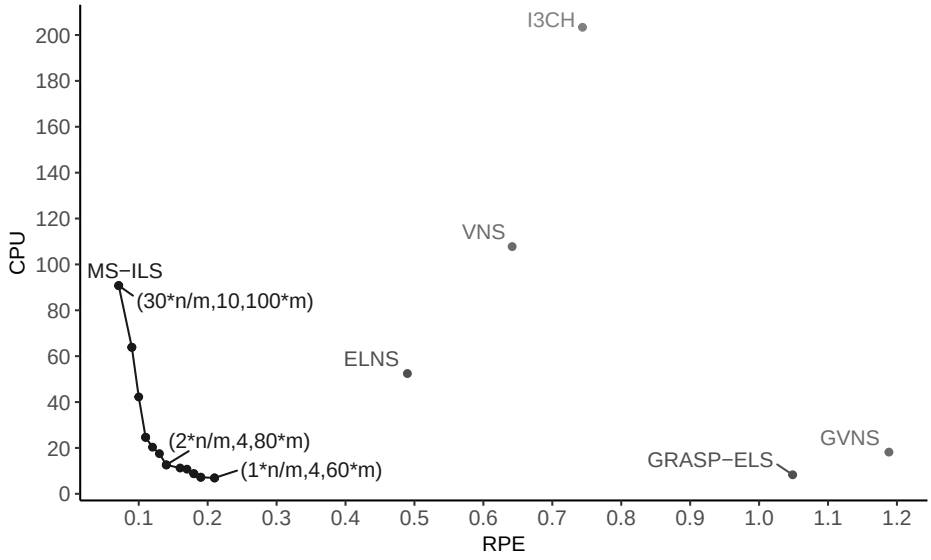


Figure 2.9 – Comparison of different MS-ILS settings with the literature based on CPU and RPE.

Tables 2.6 and 2.7 display the results of two runs of the MS-ILS using two different settings of $iter_{max}$, $iter_{ils}$ and $\mathcal{M}_{size}$. The first setting is the same as the one used in the above comparisons with the literature. The second one is a slower version of the algorithm obtained by setting $iter_{max} = 30n/m$, $iter_{ils} = 10$ and $\mathcal{M}_{size} = 100m$. The main observation here is that, if allowed to run longer, the MS-ILS is able to further improve solution quality. It is able to find the best known solutions for all of Solomon's instances at least one time out of five. On the "OPT" benchmark, it can obtain the optimal solutions for 59 out of 66 instances. Overall, it is able to find the best known solutions for 93% of the instances available in the literature.

During our experiments, including the tuning experiments, the MS-ILS found 61 new best-

Table 2.6 – Comparison of two different settings of MS-ILS on the standard benchmark.

Instances	MS-ILS (2,4,80)			MS-ILS (30,10,100)		
	rpe%	*arpe*%	cpu	*rpe*%	*arpe*%	cpu
m=1						
c100	0.00	0.00	2.0	0.00	0.00	32.5
r100	0.00	0.02	2.1	0.00	0.00	25.9
rc100	0.00	0.02	1.1	0.00	0.00	19.1
c200	0.00	0.08	9.9	0.00	0.00	80.4
r200	0.03	0.10	22.9	0.00	0.03	167.1
rc200	0.01	0.15	13.4	0.00	0.05	131.8
pr01-pr10	0.02	0.18	13.0	0.00	0.01	131.2
pr11-pr20	0.85	1.15	21.5	0.85	0.97	205.0
m=2						
c100	0.00	0.03	3.6	0.00	0.00	30.2
r100	0.00	0.04	2.7	0.00	0.01	23.8
rc100	0.00	0.05	2.9	0.00	0.01	20.8
c200	0.00	0.14	16.8	0.00	0.07	85.0
r200	0.08	0.26	30.9	0.01	0.19	187.9
rc200	0.15	0.38	25.9	0.02	0.21	171.9
pr01-pr10	0.10	0.43	18.1	0.03	0.20	124.4
pr11-pr20	0.84	1.29	32.1	0.64	0.91	254.1
m=3						
c100	0.00	0.09	5.1	0.00	0.04	39.8
r100	0.00	0.06	3.5	0.00	0.00	27.5
rc100	0.00	0.25	2.9	0.00	0.09	26.1
c200	0.08	0.18	5.2	0.02	0.11	32.1
r200	0.03	0.06	3.2	0.00	0.05	18.7
rc200	0.07	0.11	8.0	0.00	0.09	36.5
pr01-pr10	0.23	0.63	25.7	0.01	0.30	168.7
pr11-pr20	0.48	0.94	43.3	0.12	0.64	349.4
m=4						
c100	0.10	0.31	7.6	0.10	0.22	44.7
r100	0.06	0.17	4.4	0.00	0.08	34.9
rc100	0.11	0.33	3.5	0.01	0.08	31.8
c200	0.00	0.00	0.1	0.00	0.00	0.1
r200	0.00	0.00	0.1	0.00	0.00	0.1
rc200	0.00	0.00	0.1	0.00	0.00	0.1
pr01-pr10	0.35	0.77	34.4	0.00	0.51	241.2
pr11-pr20	0.44	1.03	59.1	0.12	0.69	415.7
Average	0.13	0.29	13.3	0.06	0.17	98.7

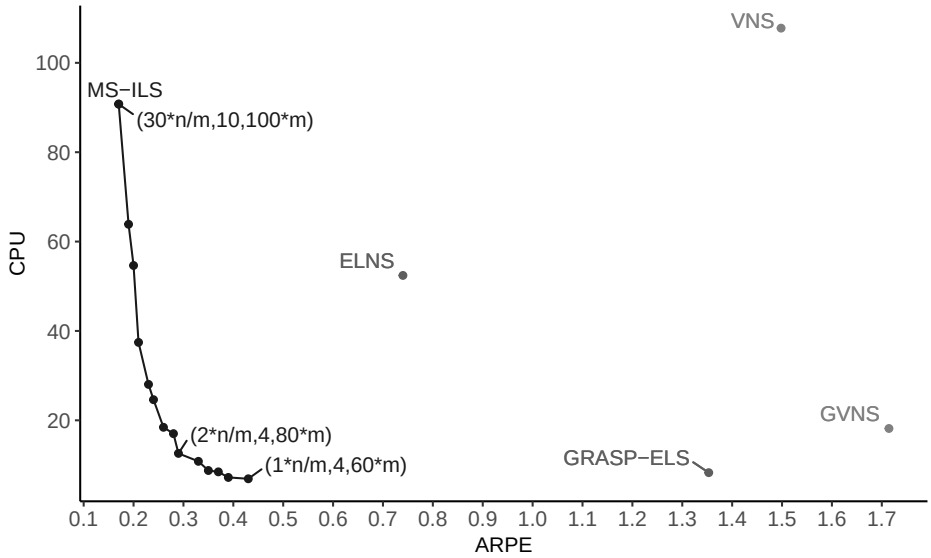


Figure 2.10 – Comparison of different MS-ILS settings with the literature based on CPU and ARPE.

Table 2.7 – Comparison of two different settings of MS-ILS on the "OPT".

Instances	Nb.	MS-ILS (2,4,80)				MS-ILS (30,10,100)			
		#OPT	*rpe*%	*arpe*%	cpu	#OPT	*rpe*%	*arpe*%	cpu
c100	9	9	0.00	0.00	0.5	9	0.00	0.00	0.6
r100	12	5	0.34	0.47	14.7	10	0.02	0.09	92.7
rc100	8	4	0.19	0.37	9.4	7	0.08	0.19	58.5
c200	8	8	0.00	0.00	0.1	8	0.00	0.00	0.1
r200	11	11	0.00	0.00	2.9	11	0.00	0.00	1.4
rc200	8	8	0.00	0.00	1.4	8	0.00	0.00	1.8
pr01-pr10	10	5	0.86	0.95	26.4	6	0.64	0.71	144.8
Avg./Total	66	50	0.20	0.26	7.9	59	0.10	0.14	42.8

known solution values for the standard benchmark instances: 23 for Solomon's instances and 38 for Cordeau's. Table 2.8 reports the new best-known solution values found by MS-ILS. For each instance, it indicates the number of vehicles and the new solution value.

Altogether, our MS-ILS is very competitive. It is able to find the current best-known solutions for 94% of the literature instances and was able to improve the best-known solutions for many of them. It achieves an average relative error (*arpe*) of only 0.17% on the standard benchmark and 0.14% on the "OPT" benchmark, and a relative percentage error (*rpe*) of 0.06% and 0.10%, on the two benchmarks, respectively, while still being faster than most state-of-the-art algorithms. In comparison, the previous best performing approach in the literature finds 65% of the current best-known solutions, and achieves an *arpe* of 0.80% on the standard benchmark and 0.34% on the "OPT" benchmark, and a *rpe* of 0.51% and 0.27%, on the two benchmarks, respectively. Finally, our experiments

Table 2.8 – New best-known solutions values found by MS-ILS.

Instance	m	Old BKS	New BKS	Instance	m	Old BKS	New BKS	Instance	m	Old BKS	New BKS
r207	1	1077	1078	c108	4	1130	1140	pr14	3	1372	1375
r208	1	1117	1118	r104	4	974	975	pr15	3	1654	1694
r209	1	961	962	r112	4	972	974	pr18	3	1281	1289
r210	1	1000	1002					pr19	3	1417	1428
rc204	1	1140	1143	pr04	2	926	928	pr20	3	1684	1722
rc208	1	1057	1058	pr05	2	1101	1103				
				pr10	2	1134	1145	pr02	4	1079	1083
r201	2	1256	1260	pr13	2	843	845	pr03	4	1232	1247
r202	2	1350	1353	pr15	2	1220	1238	pr04	4	1585	1595
r203	2	1420	1431	pr18	2	953	955	pr05	4	1838	1858
r205	2	1395	1402	pr19	2	1034	1041	pr06	4	1860	1894
r206	2	1447	1452	pr20	2	1237	1251	pr08	4	1382	1390
r209	2	1419	1423					pr09	4	1619	1622
r210	2	1430	1438	pr02	3	943	945	pr10	4	1943	1965
rc201	2	1385	1386	pr03	3	1010	1014	pr12	4	1132	1133
rc202	2	1520	1523	pr04	3	1294	1296	pr13	4	1386	1392
rc203	2	1637	1639	pr05	3	1482	1500	pr14	4	1670	1688
rc204	2	1716	1718	pr06	3	1514	1515	pr15	4	2065	2085
rc207	2	1601	1609	pr08	3	1139	1141	pr17	4	934	936
rc208	2	1691	1705	pr09	3	1275	1277	pr18	4	1539	1554
				pr10	3	1573	1582	pr19	4	1750	1780
r201	3	1441	1450	pr13	3	1145	1159	pr20	4	2062	2115

show that the MS-ILS can be tuned to either favor speed or solution quality and remain relatively better than other approaches.

2.5.4 Performance analysis

In the following, we discuss the results of the experiments conducted to evaluate the impact of each of the key components of our algorithm, namely the combination of two different search spaces and the adaptive memory. To do this, we derive alternative versions of the MS-ILS by disabling search spaces one at a time, or by disabling the adaptive memory. The considered configurations are as follows:

(a) **Standard**: the standard algorithm described in Section 2.4.

(b) **No Routes**: an alternate version of MS-ILS where the route search space operators are disabled.

(c) **No Tours**: an alternate version of MS-ILS where the giant tour search space operators and the splitting procedures are disabled. To construct solutions from the memory, we simply select m routes that do not share customers.

(d) No Memory: version of the standard algorithm where the adaptive memory and giant tour construction are disabled. Each iteration of the main loop starts from a randomly generated solution.

All of the above configurations were run ten times on each instance of both the standard and the "OPT" benchmark using the parameter values given in Section 2.5.2. Table 2.9 shows the average gap relative to the best know solutions, and the average CPU time achieved by each of the configurations from (a) to (d).

Table 2.9 – Performance analysis of MS-ILS components.

		(a) Standard		(b) No Routes		(c) No Tours		(d) No Memory	
	Instances	arpe	cpu	arpe	cpu	arpe	cpu	arpe	cpu
m=1	Solomon 100	0.02 %	2.3	0.92 %	1.0	0.06 %	2.9	0.07 %	3.1
	Solomon 200	0.09 %	21.1	1.55 %	8.1	0.15 %	18.0	0.45 %	8.6
	Cordeau	0.57 %	21.8	4.50 %	6.8	0.69 %	19.5	1.03 %	21.6
	avg	0.23 %	15.1	2.32 %	5.3	0.30 %	13.5	0.52 %	11.1
m=2	Solomon 100	0.03 %	3.7	1.17 %	1.6	0.04 %	3.4	0.37 %	2.4
	Solomon 200	0.31 %	26.8	1.65 %	46.0	0.42 %	12.6	0.98 %	6.5
	Cordeau	0.83 %	31.5	4.98 %	8.8	1.05 %	21.0	2.66 %	14.9
	avg	0.39 %	20.7	2.60 %	18.8	0.50 %	12.3	1.33 %	8.0
m=3	Solomon 100	0.12 %	4.5	1.30 %	6.5	0.12 %	3.7	0.85 %	2.3
	Solomon 200	0.28 %	6.6	1.57 %	4.4	0.32 %	4.7	1.39 %	1.9
	Cordeau	0.83 %	42.3	5.03 %	13.7	1.00 %	24.8	3.37 %	13.2
	avg	0.41 %	17.8	2.63 %	8.2	0.48 %	11.1	1.87 %	5.8
m=4	Solomon 100	0.28 %	6.7	1.72 %	4.3	0.33 %	4.7	1.49 %	2.0
	Solomon 200	0.00 %	0.1	0.00 %	0.1	0.00 %	0.1	0.00 %	0.1
	Cordeau	0.84 %	54.2	5.44 %	22.2	1.07 %	24.8	3.97 %	12.8
	avg	0.37 %	20.3	2.39 %	8.9	0.47 %	9.9	1.82 %	5.0
OPT	Solomon 100	0.31 %	11.2	0.97 %	10.3	0.55 %	2.5	0.70 %	2.3
	Solomon 200	0.00 %	1.7	0.10 %	16.7	0.00 %	0.9	0.02 %	1.1
	Cordeau	0.99 %	28.2	1.76 %	70.3	1.10 %	6.7	1.42 %	10.3
	avg	0.44 %	13.7	0.94 %	32.5	0.55 %	3.3	0.72 %	4.6

Comparisons between the configurations (a) to (c) highlight the impact of alternating between route and giant tour search spaces instead of only using one of them. Disabling either one of the search spaces translates into a decrease of solution quality compared to configuration (a), but removing the route operators has a bigger impact on solution quality than removing giant tour operators. We still decided to keep both spaces in the design of the algorithm because both of them help improve solution quality; besides, the giant tour search spaces is still needed to find good solutions for some instances.

Comparisons between configurations (a) and (d) show the contribution of including the

adaptive memory into the multi-start framework. As we can see, adding the adaptive memory significantly improves the average gap on the two benchmarks, with a reasonable impact on CPU time. In terms of solution quality, the use of the adaptive memory has a more significant impact when solving instances of the standard benchmark with $m \geq 2$ than when solving instances with $m = 1$. This is because when $m = 1$, the algorithm chooses a previous local optimum and tries to improve it, whereas when $m \geq 2$, the algorithm constructs new solutions using several from previous local optima.

Most of the time, disabling one component results in a decrease in computation times. However, depending on the instance, the opposite can happen, and disabling one component may result in bigger computation times. For example, in the standard benchmark on Solomon's instances with $m = 1$; disabling the route space local search operators hinders the progress of the algorithm towards good solutions, which explains the increase in CPU time.

2.6 Conclusion

In this chapter, we introduced a simple yet very effective Multi-Start Iterated Local Search (MS-ILS) for the Team Orienteering Problem with Time Windows. Our algorithm is based on a local search that alternates between two different search spaces: the *route search space* that corresponds to actual solutions to the TOPTW, and the *giant tour search space* that makes it possible to explore the solution space without being limited by time windows and length constraints. In order to improve solution quality, several local search operators are included to be used in each search space. Additionally, the algorithm integrates an adaptive memory mechanism to further improve performance by making use of previous local optima to build better solutions.

Computational results have shown that our approach performs very well compared to state-of-the-art algorithms and is able to outperform them in terms of overall solution quality and computation times. In particular, the MS-ILS is able to find the current best-known solutions, or better ones, for 94% of the benchmark instances within reasonable runtimes, and achieves an overall average relative gap of 0.17% and 0.14% on the two benchmarks of the literature, respectively. Our approach was also able to find new best solutions for 61 instances for which no optimal solution has yet been found.

Finally, the method proposed herein is flexible in the sense that it can be easily adapted to accommodate new constraints or to address other variants of the problem. One potential

direction for future research would be to extend our algorithm to more realistic versions of orienteering problems with time windows.

A Neighborhood Search and Set Cover Hybrid Heuristic for the Two-Echelon Vehicle Routing Problem

The *Two-Echelon Vehicle Routing Problem* (*2E-VRP*) is a variant of the classical vehicle routing problem arising in the context of city logistics. In the 2E-VRP, freight from a *main depot* is delivered to final customers using intermediate facilities, called *satellites*. In this paper, we propose a new hybrid heuristic method for solving the 2E-VRP that relies on two components. The first component effectively explores the search space in order to discover a set of interesting routes. The second recombines the discovered routes into high-quality solutions. Experimentations on benchmark instances show the performance of our approach: our algorithm achieves high-quality solutions in short computational times and improves the current best known solutions for several large scale instances.

3.1 Introduction

Freight transportation is a key factor underpinning economic growth. However, it is also a major nuisance, especially in urban areas where congestion and environmental effects disturb people's well-being. As demand for freight transportation increases, new transport policies and better traffic management become essential to limit its effects. The concept of city logistics is one approach to solving the problem. It aims to optimize freight transportation within city areas while considering traffic congestion and environmental issues as well as costs and benefits to the freight shippers [237]. Some of the most used models in city logistics are multi-echelon distribution systems, especially two-echelon systems.

In a two-echelon distribution system, delivery from one or more depots to the customers is managed by shipping and consolidating freight through intermediate depots called *satellites*. Freight is first moved from the depots to the satellites using large trucks. Then, freight is delivered from the satellites to the customers using smaller vehicles. Proceeding like this allows to shape more conveniently the fleet of vehicles to be used, as larger trucks are more cost efficient whereas smaller ones are preferable in city centers. Because the flow of freight in each echelon depends on that in the other echelon, routing problems arising in two-echelon distribution systems must be studied as a whole; they cannot be merely decomposed into two separate sub-problems. The problem that studies how to efficiently route freight in such systems is known as the *Two-Echelon Vehicle Routing Problem (2E-VRP)*.

In this work, we consider the basic version of the 2E-VRP. It is characterized by a single depot and a set of satellites. A fleet of homogeneous vehicles of known size is available at each echelon. Vehicle capacities are limited. Only one type of product is to be shipped and split deliveries are only allowed at the first level. The objective is to minimize the total routing cost in both levels.

To address this problem, we propose a hybrid heuristic that relies on two components embedded in an iterative framework. The first component aims to generate a set of promising routes using destroy and repair operators combined with an efficient local search procedure. The second component recombines the generated routes by solving a set covering problem to obtain a high quality solution. Computational experiments conducted on the test instances of the literature show the performances of our approach, as it reached high quality solutions in short computing times, and was able to improve the current best known solution for several large instances.

The remainder of this paper is organized as follows. In Section 3.2, an overview of the related literature is given. The problem is described in Section 3.3, and the proposed approach is explained in Section 3.4. Section 3.5 presents computational results and compares them to the best known solutions of the literature. Finally, Section 3.6 concludes and discusses possible directions for future research.

3.2 Literature review

One of the first studies on the optimisation of two-echelon distribution systems was presented by Jacobsen and Madsen [138, 177] who considered the problem of daily

newspaper distribution involving two editors who wanted to share their facilities and transportation means to reduce costs. In their problem, they considered several transfer points to transfer newspapers from one vehicle to another, but unlike the 2E-VRP they didn't allow for split deliveries and retailers could be served directly from the printing office.

Crainic et al. [68] studied a new organizational and technological framework for urban freight management in congested areas. They used data from the city of Rome to design a two-tier distribution system. They introduced intermediate facilities to consolidate freight and redistribute it among small vehicles, as larger ones cannot pass through the narrow streets of the city centre. They also proposed a mathematical model for the problem of locating the satellite facilities. Later, Crainic et al. [69] considered a two-echelon, synchronized, scheduled, multi-depot, multiple-tour, heterogeneous vehicle routing problem with time windows for which they presented a general mathematical model and promising algorithmic avenues. Subsequently, Crainic et al. [67] compared the performance of a analysed the impact of various parameters on the total cost of the 2E-VRP. They studied the impact of depot location, satellite numbers and locations, and customer distribution. They conclude that the 2E-VRP yields better results than the VRP when the depot is located outside the customers area. In [66], the authors provide a complementary study where they considere a more general cost that includes infrastructure costs, operational costs, and environmental cost.

Perboli et al. [193] introduced a flow-based formulation for the 2E-VRP and generated three sets of instances with up to fifty customers and four satellites, based on CVRP instances. The authors also proposed valid inequalities for the 2E-VRP and two math-based heuristics. Their Branch-&-Cut algorithm (B-&-C) was able to optimally solve instances with 21 customers. By using new families of valid inequalities, Perboli et al. [192] were able to optimally solve all the instances with 21 customers and to reduce the gap on the larger ones.

Crainic et al. [64] developed two heuristics for the 2E-VRP. Both algorithms proceed by separating the first and second echelon, and solving them sequentially. However, one solves the second level routing problem by decomposing it into a set of independent VRPs using a clustering heuristic, while the other treats it as multi-depot VRP.

A multi-start heuristic for solving the 2E-VRP was proposed in [65]. The algorithm starts by assigning the customers to the satellites using a heuristic criterion then proceeds to the resolution of the $m+1$ resulting VRPs, where m is the number of satellites, using an exact

method. After that, the solution is perturbed by randomized changes to the customer-to-satellite assignment, and the problem is solved again until a maximum number of iterations is reached. Components of the multi-start heuristic were used in a hybrid GRASP with path re-linking in [63]. The algorithm first generates solutions using GRASP. Some of the generated solutions may be infeasible, a feasibility search is then applied. Feasible solutions are improved using a local search. Finally, using path re-linking, the algorithm tries to further improve the solutions.

Hemmelmayr et al. [132] designed an Adaptive Large Neighbourhood Search (ALNS) for the 2E-VRP. The ALNS algorithm proceeds by removing, at each iteration, a subset of customers from the current solution using a destroy operator and, then, re-inserting the removed customers in different positions using a repair operator. The authors presented 4 repair operators and 8 destroy operators divided into two sets : those that change the satellite configuration of the current solution by opening/closing satellites, and those that keep the satellite configuration unchanged and have a smaller impact on the structure of the solution. New larger instances with up to 200 customers were also introduced. The ALNS improved many of the best solutions previously published.

Jepsen et al. [140] presented a Branch-&-Cut algorithm for the 2E-VRP based on a MILP formulation. Said formulation is a relaxation of the 2E-VRP that provides lower bounds for the problem but not necessarily feasible solutions. Therefore, the authors also present a feasibility problem to test if the produced integer solutions are valid solutions for the 2E-VRP, and a branching scheme to branch on infeasible ones. The B&C algorithm was able to solve to optimality instances with up to 50 customers.

Santos et al. [214] developed two Branch-&-Price (B&P) algorithms to solve the 2E-VRP : one only considers routes that satisfy elementary constraints while the other relaxes such conditions when pricing. In a later work [215], they proposed a reformulation of the problem that overcomes symmetry issues observed in their previous formulations, and implemented a Brand-&-Cut-&-Price algorithm by incorporating valid inequalities into their B&P. The new algorithm performed well in comparison to other exact methods.

The current best exact method for the 2E-VRP was introduced by Baldacci et al. [21]. Its main idea is to decompose the 2E-VRP into a set of MDVRPs with side constraints then solve the generated set of subproblems to obtain an optimal solution for the 2E-VRP. The proposed algorithm relies on a new ILP formulation that is used to derive a relaxation, and a bounding procedure based on dynamic programming and a dual ascent method. The authors provide detailed results on instances from previous literature, as well as a

newly generated set of instances with up to 100 customers.

Zeng et al. [272] presented a hybrid two phase heuristic for the 2E-VRP composed of a GRASP and Variable Neighbourhood Descent (VND). First, the GRASP generates a feasible solution using a route-first cluster-second procedure, then, the VND tries to improve it. The process is repeated until a maximum number of iterations is reached. The algorithm provides good results but, unfortunately, only instances comprising up to 50 customers were used for the tests.

Breunig et al. [39] published a Large Neighbourhood Search (LNS) for the 2E-VRP. Their algorithm is based on the destroy-and-repair approach. At each iteration, a destroy operator is used to remove customers from the current solution, then a repair operator is used to re-insert them at different positions, and a local search is performed to improve the new solution. The authors use 5 different destroy operators and one repair operator. While its principle resembles that of the ALNS of [132], the algorithm of [39] is faster and conceptually simpler. The authors also resolved inconsistencies between different versions of benchmark instances, and made them available online in a unified format. The LNS-2E was able to improve the best known solutions for several instances of the literature.

Recent years saw a growing interest for the 2E-VRP with the introduction of different variants of the problem. The *Time Dependant 2E-VRP with Environmental Considerations* was introduced in [226]. Contrary to the standard 2E-VRP, this variant acounts for accounts for vehicle type, traveled distance, vehicle speed, load, multiple time zones, and CO_2 emissions. The authors proposed a MILP formulation of the problem and some valid inequalities to strenghten it, and did a case study on a Dutch supermarket chain to show the applicability of their model to a real-life problem. Wang et al. [267] proposed an algorithm for the *2E-VRP with Environmental Considerations (2E-CVRP-E)* that closely resembles the prévious variant. Their algorithm comprises of a Variable Neighborhood Search (VNS) followed by the resolution of a linear programm to further improve the obtained solution. They tested their algorithm on newly proposed instances for the 2E-VRP-E and on instances for the standard 2E-VRP, and were able to improve several best known solutions for 2E-VRP instances.

Grangier et al. [122] proposed an adaptive large neighborhood search (ALNS) for the *Two-Echelon Multiple-Trip Vehicle Routing Problem with Satellite Synchronization (2E-MTVRP-SS)*. This variant of the problem was first discussed in [69] and [193] but neither a resolution algorithm nor instances were proposed since then. The proposed ALNS uses a set of custom repair and destroy heuristics, and an efficient feasability check to solve

the problem. Results are reported on instances obtained from the extension of standard 2E-VRP instnaces.

In [222], the *Adaptive Two-Echelon Capacitated Vehicle Routing Problem (A2E-CVRP)* was proposed. Compared to 2E-CVRP, A2E-CVRP considers multiple depots and allows direct shipping from the depot to the customers. The authors introduced a mathematical formulation and lower bound for A2E-CVRP that is also used for deriving an upper bound. They performed computational experiments on the small instances of the 2E-VRP and showed that the A2E-CVRP yields lower route costs compared to the classical 2E-VRP.

Finally, Zhou et al. [273] introduced a *Multi-Depot Two-Echelon Vehicle Routing Problem with Delivery Options (MD-TEVRP-DO)* for last mile distribution. This variant of the problem considers multiple depots from which satellites can be served, and more importantly, each customer is associated with a particular depot and can be served in two different ways. Vehicles can serve customers either by direclty visiting the customer (Home Delivery) or by leaving the shipment at a pickup facility from which the customer can retrieve it (Customer Pickup). The authors proposed a Hybrid Multi-Population Genetic Algorithm to solve the problem and tested it on an instance generated from real world data and on several randomly generated instances.

3.3 Problem definition

The 2E-VRP is defined on a weighted undirected graph $G = (V, A)$, where V is the set of nodes and A the set of arcs. Set V is partitioned as $V = \{v_0\} \cup V_{sat} \cup V_{cust}$. Node v_0 represents the depot, subset V_{sat} contains n_{sat} satellites and subset V_{cust} contains n_{cust} customers. Set $A = A_1 \cup A_2$ is divided into two subsets. $A_1 = \{(i,j) : i,j \in \{v_0\} \cup V_{sat}, i \neq j\}$ contains the arcs that can be taken by first level vehicles: trips between the depot and the satellites and trips between pairs of satellites. $A_2 = \{(i,j) : i,j \in V_{sat} \cup V_{cust}, (i,j) \notin V_{sat} \times V_{sat}, i \neq j\}$ contains the arcs that can be taken by second level vehicles: trips between customers and satellites and trips between pairs of customers. A travel cost $c_{ij},\ (i,j) \in A$, is associated with each arc. We assume that the matrix (c_{ij}) satisfies the triangle inequality.

Each customer $i \in V_{cust}$ demands d_i units of freight to be delivered. The demand of a customer cannot be split among several vehicles, that is, a customer must be served exactly once. Moreover, customer demands cannot be delivered by direct shipping from the depot and must be consolidated at a satellite. Satellite demands are not explicitly

given but considered to be the sum of all the customer demands that are served trough the satellite. We assume that it can exceed vehicle capacity and thus, we allow for it to be split among different vehicles e.i. a satellite can be served by more than one vehicle. A satellite may also have a demand equal to zero and, in this case, not be visited by any vehicle. Consolidating shipments at satellite $s \in V_{sat}$ incurs handling costs equal to h_s times the quantity of handled goods.

A fleet f_1 of m_1 identical vehicles of capacity Q_1 is located at the depot v_0 and is used to deliver goods to the satellites. Additionally, a fleet f_2 of m_2 identical vehicles of capacity Q_2 is available for serving the customers. Each of the m_2 vehicles can be located at any satellite $s \in V_{sat}$ as long as the number of vehicles at one satellite does not exceed a limit k_s.

We define a first-level route as a route performed by a first-level vehicle that starts at the depot, visits one or several satellites then returns to the depot. In a same way, we define a second-level route as a route run by a second-level vehicle that starts at satellite $s \in V_{sat}$, visits a subset of customers before returning to s. Routes must respect vehicle capacities, that is, the sum of deliveries made by a first-level route to the satellites it visits must not exceed Q_1 and the total demand of the customers visited by a second-level route must not exceed Q_2. Each vehicle performs only one tour, and each route has a cost equal to the sum of the costs of the arcs used.

The objective of the 2E-VRP is to find a set of routes at both levels such that each costumer is visited exactly once, the capacity constraints are respected, the quantity delivered to costumers from each satellite is equal to the quantity received from the depot, and the total routing and handling costs are minimized. Figure 3.1 shows a solution example for the 2E-VRP.

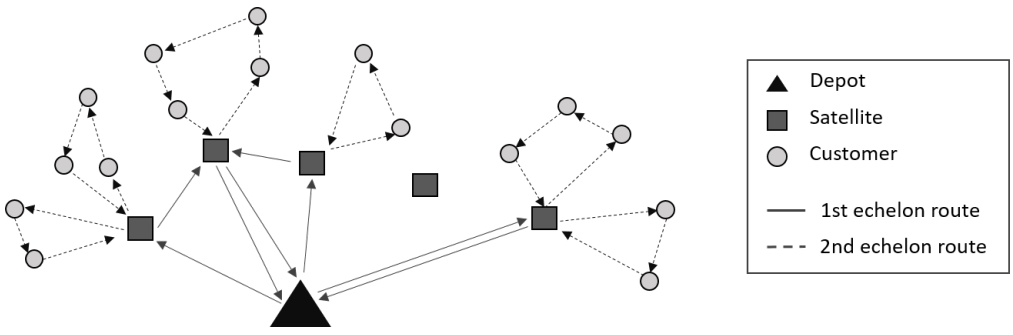


Figure 3.1 – Example of a 2E-VRP solution.

3.4 Solution method

We propose a hybrid heuristic that relies on a neighborhood search to generate good feasible solutions, and a integer programming (IP) method to recombine the routes from those solutions into a better one. Algorithm 3 summarizes the steps of our method.

Algorithm 3: Neighborhood Search and Set Cover hybrid heuristic for the 2E-VRP.

Data: 2E-VRP instance
Result: S_{best} solution for the 2E-VRP instance

```
1  begin
2      S ← S_0
3      S_best ← S_0
4      S_best ← BestInsertionHeuristic()
5      S ← S_best
6      pool ← {}
7      while !Stopping criteria do
8          S ← RouteGenerationHeuristic(S, pool)
9          if cost(S) < cost(S_best) then S_best ← S
10         S ← RouteRecombination(pool)
11         if cost(S) < cost(S_best) then
12             S_best ← S
13         else
14             S ← GreedyInsertionHeuristic()          // Restart
15     return S_best
```

At each iteration of the algorithm, the route generation heuristic takes an initial solution S and tries to improve it while exploring the solution space and storing new routes in the pool. After that, the recombination component uses the discovered routes to construct a better solution by solving a Set Cover based formulation of the 2E-VRP. If the recombination fails to produce a better solution, the algorithm constructs a different one from scratch and uses it as initial solution for the route generation heuristic during the next iteration. The idea of the approach is to use the integer program as a mean to find better quality solutions missed by the route generation heuristic while guiding the search process towards different regions of the solution space.

A comparable approach was proposed by Rochat et al. [210] for the vehicle routing problem with time windows. However, they used the exact model as a post-optimization method after the completion of their heuristic. This is generally the case due to the exponential worst case performance of the model. What is new in our proposal is that we iteratively apply a Set Cover (SC) based formulation of the problem as a refinement technique rather

Algorithm 4: Route generation heuristic for the 2E-VRP.

Data: 2E-VRP instance
Initial Solution S_0
Result: S_{best} solution for the 2E-VRP instance

```
1  begin
2      S ← S_0
3      S_best ← S_0
4      repeat
5          i ← 1
6          while i < i_max do
7              S_tmp ← destroy(S, τ)
8              S_tmp ← localSearch(repair(S_tmp))
9              S_tmp ← firstLevelReconstruction(S_tmp)
10             pool ← update(pool, S_tmp)          // Add routes to pool
11             if cost(S_tmp) < cost(S) then
12                 S ← S_tmp
13                 i ← 1
14                 τ ← τ_m in
15             else
16                 i ← i + 1
17                 increment(τ)
18         cost(S) < cost(S_best)  S_best ← S
19         iter ← 0
20         iter ← iter + 1
21         S ← perturb(S)
22     until iter = iter_repeat
23     return S_best
```

than focusing on the local search results. We show that is possible to combine efficiently heuristic and exact algorithms to explore the search space within short runtimes.

3.4.1 Route generation heuristic

The neighborhood search we use to explore the solution space is based on the destroy-and-repair principle. At each iteration, a part of the solution is destroyed by removing a limited number of customers using a *destroy operator*. The removed customers are then re-inserted into the solution with a *repair operator*. The structure of this heuristic is described in Algorithm 4.

Starting from an initial solution, a random number $\eta \in [1, \tau]$ of customers is removed from the second echelon. The maximum number of customers to be removed τ is first initialized

to τ_{min} and then increased after each non-improving iteration until it reaches τ_{max}. As soon as an improvement is found, τ is reset to τ_{min}. Slowly varying the value of τ during the execution allows to intensify the search around promising solutions and then to slowly increase diversification as the search converges toward a local optima. Once the solution is destroyed, the removed customers are reinserted using a *repair operator* and the obtained solution is passed to a *local search* to improve the second echelon routes. After that, the satellite demands are computed and the first echelon routes are constructed to obtain a complete solution. If the new solution has a better objective value than S, it is accepted as the new incumbent. Moreover, after i_{max} consecutive iterations without improving the incumbent solution, the best-known solution is updated and the configuration of the available satellites is modified using $perturb(S)$ to allow the search procedure to explore a different region of the solution space. The solution obtained after the perturbation becomes the new incumbent. The algorithm ends after $iter_{repeat}$ consecutive iterations have been performed without improving the best-found solution.

3.4.1.1 Destruction

The destroy procedure only considers the second level routes. At each iteration, it randomly chooses one of the following operators and removes a random number of customers η in $[1, \tau]$.

a. **Random removal operator:** removes η randomly chosen customers from the solution.

b. **Worst removal operator:** removes the customers with the highest increase in solution cost. More precisely, it calculates for each customer k located between i and j a saving value $c_{ik} + c_{kj} - c_{ij}$. Savings are then normalized by the average cost of the incident arcs of the corresponding customer and altered by a random factor between 0.8 and 1.2 as in [132]. Finally, customers are sorted in decreasing order of their normalized savings and the η first customers are removed from the solution. Normalizing the savings serves to avoid repeatedly removing the customers that are isolated from the others.

c. **Sequence removal operator:** removes a sequence of η consecutive customers from a randomly chosen route. If η is larger than the chosen route, the whole route is destroyed and the remaining number of customers is removed from a second route.

3.4.1.2 Repair and first level reconstruction

Repair is performed by using two heuristics : *Best Insertion Heuristic* (BIH) and *Greedy Insertion Heuristic* (GIH). When repairing an incomplete solution, we first use BIH. This constructive heuristic identifies among all the unrouted customers the one that increases the least the total solution cost and inserts it at its best position. It repeats the process until all customers are routed. If one or more customers remain unrouted because their demands are higher than the largest remaining capacity of any vehicle, the repair process is restarted using GIH. The *Greedy Insertion Heuristic* inserts customers in a random order one after the other at their cheapest possible position in the solution. If the GIH fails, the customers are randomly reordered and the heuristic restarts. We observed that proceeding this way is sufficient to achieve feasible solutions after a small number of tries. These repair heuristics consider feasible insertions in already existing routes. If the maximum number of vehicles is not yet reached, the creation of new empty routes from open satellites is also tested.

The construction of the first-echelon routes is achieved by means of a heuristic similar to GIH. The heuristic starts by creating for each satellite with a demand greater than Q_1 enough back-and-forth trips so that its remaining demand becomes smaller than Q_1. Once it is done, the heuristic proceeds to insert of the remaining demands the same way as GIH.

3.4.1.3 Local search

The local search procedure consists of the following operators : $2-opt$, $2-opt^*$, $Relocate(\lambda)$, and $Swap(\lambda_1, \lambda_2)$ with $\lambda, \lambda_1, \lambda_2 \in \{1, 2\}$. The $2-opt$ operator [169] is performed on each route and $2-opt^*$ [197] is performed on routes originating from the same satellite. *Relocate* moves sequences of λ customers to their best positions in the solution. Finally, *Swap* exchanges the positions of two sequences of λ_1 and λ_2 customers from the same route or from two different routes. At each iteration, the local search procedure randomly applies one of the above operators. If the chosen operator does not improve the solution, it is discarded, otherwise the set of operators is reset. The process continues until all operators have been discarded. Moves from each operator are performed in a first-improvement manner until no improving move can be found in the neighborhood.

3.4.1.4 Perturbation

In order to explore different regions of the search space, we temporarily close satellites and reopen them using the *Close Satellites* and *Open Satellites* operators.

a. **Close Satellites:** randomly chooses one satellite among the open ones having at least one route originating from them and closes it. The routes of the chosen satellite are reassigned to an open satellite that keeps their cost to a minimum. When the number of open satellites becomes less than the minimum required to serve all customers, the operator chooses a random satellite among the closed ones and opens it.

b. **Open Satellites:** chooses a random number of satellites among those that are closed and opens them. In order to allow the number of open satellites to decrease, especially at the beginning when most of them are open, the number of satellites to be opened can be nil.

3.4.2 Recombination method

The route recombination component uses a pool of routes collected during the search process and recombines them to obtain a high-quality solution by solving a set cover based formulation of the problem. In the following, we introduce the notations used in the IP model.

Let $\mathcal{M}$ be the set of all the possible first level routes, and $\mathcal{M}_s \subseteq \mathcal{M}$ the subset of first-level routes that serve satellite $s \in V_{sat}$. We note g_r the cost of route $r \in \mathcal{M}$. Let $\mathcal{R}$ be the set of all the possible second-level routes, and $\mathcal{R}_s$ the subset or routes passing through $s \in V_{sat}$, thus $\mathcal{R} = \bigcup_{s \in V_{sat}} \mathcal{R}_s$. We associate to each route $r \in \mathcal{R}$ a cost c_r, and a load $w_r = \sum_{c \in r} d_c$ equal to the total demand of customers visited in route r. The binary parameter δ_{ri} is equal to 1 if and only if route $r \in \mathcal{R}$ visits customer $i \in V_{cust}$, and 0 otherwise. The second-level routes having been extracted from valid solutions, they all satisfy the vehicle capacity constraints.

Let $y_r \in \{0, 1\}$ be a binary decision variable equal to 1 if and only if first-level route $r \in \mathcal{M}$ is in the solution, $x_r \in \{0, 1\}$ a binary decision variable equal to 1 if and only if second-level route $r \in \mathcal{R}$ is in the solution, and q_{sr} a non-negative variable representing the amount of goods delivered by route $r \in \mathcal{M}$ to satellite $s \in V_{sat}$. We assume that

$q_{sr} = 0$ if satellite s is not visited in route r. Parameter h_s represents handling costs at satellite $s \in V_{sat}$. The route recombination model can be formulated as follows:

$$\min \; z = \sum_{r \in \mathcal{R}} c_r \cdot x_r + \sum_{r \in \mathcal{M}} g_r \cdot y_r + \sum_{s \in V_{sat}} \sum_{r \in \mathcal{M}_s} h_s \cdot q_{sr} \tag{3.1}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}} \delta_{ri} \cdot x_r \geq 1 \quad , \quad \forall i \in V_{cust} \tag{3.2}$$

$$\sum_{r \in \mathcal{R}_s} x_r \leq k_s \quad , \quad \forall s \in V_{sat} \tag{3.3}$$

$$\sum_{r \in \mathcal{R}} x_r \leq m_2 \tag{3.4}$$

$$\sum_{r \in \mathcal{M}} y_r \leq m_1 \tag{3.5}$$

$$\sum_{r \in \mathcal{M}_s} q_{sr} = \sum_{r \in \mathcal{R}_s} w_r \cdot x_r \quad , \quad \forall s \in V_{sat} \tag{3.6}$$

$$\sum_{s \in V_{sat}} q_{sr} \leq Q^1 \cdot y_r \quad , \quad \forall r \in \mathcal{M} \tag{3.7}$$

$$x_r \in \{0, 1\}, \quad r \in \mathcal{R} \tag{3.8}$$

$$y_r \in \{0, 1\}, \quad r \in \mathcal{M} \tag{3.9}$$

$$q_{sr} \in \mathbb{R}_+, \quad s \in V_{sat}, \; r \in \mathcal{M} \tag{3.10}$$

The objective function (3.1) states to minimize routing costs on both levels plus handling costs at each satellite. Constraints (3.2) ensure that each customer is visited at least once. Constraints (3.3) limit the number of second-level vehicles per satellite. Constraints (3.4) and (3.5) impose upper bounds on the number of vehicles used to implement first and second level routes. Balance between the quantity delivered by first-level routes to a satellite and the customer demands supplied from said satellite is imposed by constraints (3.6). Constraints (3.7) ensure that the capacity of first-level vehicles in not exceeded. Because the total amount of goods that need to be supplied to each satellite is not known beforehand, we cannot assume that capacity constraints are respected by first-level routes like we did for second-level routes. We need to explicitly state them in the formulation. Finally, constraints (3.8), (3.9), and (3.10) define the values domain for the decision variables.

Note that the model we use in our recombination component is a relaxation of the 2E-VRP. Constraints (3.2) require that each customer is visited at least once, instead of exactly once. However, since the distance matrix satisfies the triangle inequality, the

two formulations remain equivalent as the resolution process will naturally lean towards solutions with the least possible amount of visits to a same customer. If the pool contains all the possible routes, solving the formulation with the relaxed model will still result in an optimal solution where each customer is visited exactly once. The idea of relaxing the problem stems from the fact that the recombination pool only contains a limited subset of routes, thus the solutions it finds may be few. To increase the number of combinations that can be made, we choose to allow combining routes that share common customers, as it can lead to better objective values. Even though the resulting combination may not be a valid solution to the 2E-VRP, removing the extra visits to each customer makes it feasible while producing new routes and further lowering the objective value.

3.4.2.1 Pool management and initialization

The performance of the route recombination component strongly depends on the size of the pool of routes. A larger size increases the chances of finding high-quality solutions but also induces higher computation times, whereas a small size reduces computation times but makes finding improved solutions less likely. Thus, pool size must be fixed in order to offer a good trade-off between solution quality and computation efforts. Furthermore, to account for the lesser number of available routes, it is better to keep inside the pool only routes that are more likely to be in high-quality solutions. To this end, we assign each route a priority based on the cost of the solution it was extracted from, thus favoring routes that belong to the best found solutions. When the pool capacity is reached, routes with lower priority are removed and replaced by the new ones. If a route already exists inside the pool, its priority is updated if it is extracted from a better solution.

The pool is initialized with the routes of x different solutions generated by the *Greedy Insertion Heuristic* described in Section 3.4.1.2 and improved with the local search procedure described in Section 3.4.1.3. Furthermore, for each satellite s we add m^1 copies of round trip routes to s from the depot to account for the possibility of it being served more than once.

3.4.2.2 Correcting heuristic

When the route recombination model is solved, some customers might be visited more than once. In this case, we use a correcting heuristic to remove the extra visits and produce a valid solution. The algorithm starts by establishing the set V_{cm} of customers that are

visited more than once. It then computes for each visit v of each customer $i \in V_{cm}$ its removal gain δ_{iv}, removes the visit with the highest gain and updates the gains for the remaining ones. When the number of visits to a customer drops to one, it is removed from V_{cm}. The procedure is repeated until V_{cm} becomes empty. During our tests, we observed that only a few customers tend to be visited multiple times. Thus, this simple heuristic proves to be enough to provide good results with limited computational effort.

3.5 Computational results

Our algorithm was coded in C++ using the Standard Template Library (STL) for data structures, and IBM ILOG CPLEX 12.6.3 to solve the IP. The algorithm is compiled with the GNU GCC compiler in a Linux environment and tested on an Intel Xeon E5-2670v2 CPU at 2.50GHz with similar performance to the ones used in the literature.

We conducted extensive computational experiments on the benchmark instances for the 2E-VRP. There are currently six instance sets available. The size of the instances ranges from 12 customers and 2 satellites, to 200 customers and 10 satellites. The main characteristics of the benchmark instances are listed in Table **??** of Appendix **??**. Note that the small instances of *Set 1* are no longer used for testing, thus they are not included. For our tests we used the files provided by Breunig et al. [39].

3.5.1 Parameter tuning

The proposed approach has six parameters: (1) i_{max}, $iter_{repeat}$, τ_{min} and τ_{max} in the route generation heuristic; (2) the size of the pool (S_{pool}) in the route recombination component; and (3) the stopping criterion of the iterative framework. We carried out a series of preliminary experiments to set the parameter values: we tested our algorithm on a subset of instances while varying parameter values, and kept those that offered the best trade-off between solution quality and runtime. The stopping criteria is set according to previous literature. Breunig et al. [39] set the maximum runtime of their algorithm to 60s for small instances and 900s for larger ones. Wang et al. [267] use the maximum runtime and the maximum number of iterations N_{algo} without improving the best found solution as stopping rules. They set them so that the maximum runtime of their algorithm does not exceed 1500s. To show the performance of our method we restrict our runtime to

60s and 900s as do Breunig et al. [39]. The remaining parameter settings are given in Table 3.1.

Parameter	Description	Value
i_{max}	max. nb. of non-improving iterations before perturbing the solution	$0.2n$
$iter_{repeat}$	max. nb. of non-improving iterations for the route generation	10
τ_{min}, τ_{max}	max. nb. of customers to be removed	$0.15n, 0.45n$
S_{pool}	size of the pool	$\frac{\sum d_c}{m^2} * 15$
N_{algo}	max. nb. of non-improving iterations in the global algorithm	n

Table 3.1 – Parameter settings.

3.5.2 Comparison with the literature

In order to investigate the effectiveness of the proposed algorithm, we compare its performance, when applicable, with that of the ALNS by Hemmelmayr et al. [132], the LNS by Breunig et al. [39] and the VNS by Wang et al. [267] as well as the current best-known solution for each instance from the literature. All the results were obtained through five independent runs of the algorithm and are summarized in Table 3.2. The results of our Neighborhood Search and Set Cover Hybrid Heuristic are listed in column "NS-SC". The columns "ALNS", "LNS", and "VNS" show the results of the methods proposed by Hemmelmayr et al. [132], Breunig et al. [39], and Wang et al. [267], respectively. The average and the best objective value of the five runs are given in columns "Avg. 5" and "Best 5", respectively. Column "CPU" shows the average runtime of the algorithm in seconds. The column "BKS" refers to the best-known solution of that set of instances. As was observed by Breunig et al. [39], there exist some small differences in objective values that can be explained by a different rounding convention or the small optimality gap of CPLEX. Table 3.3 summarizes the gaps obtained by each algorithm on each benchmark. Columns "Avg. %" and "Best %" show the average and best gap, respectively, expressed as a percentage. The overall gap is calculated by considering the number of instances in each benchmark. The detailed results obtained by our NS-SC are presented in Appendix B, where they are compared to the best-known solutions in the literature.

Instances in Sets 2 and 3, are relatively easy to solve and all algorithms are able to find the

Table 3.2 – Computational results for 2E-VRP instances.

Instances		$\|V_{cust}\|$	ALNS			LNS			VNS			NS-SC			BKS
			Best 5	Avg 5	CPU	Best 5	Avg 5	CPU	Best 5	Avg 5	CPU	Best 5	Avg	CPU	
Set 2	a	21	410.69	410.69	35	410.69	410.69	60	410.69	410.69	1	410.69	410.69	1	410.69
		32	744.49	744.49	69	744.49	744.49	60	744.49	744.49	4	744.49	744.49	6	744.49
	b	50	559.20	559.20	147	559.20	559.20	60	559.20	559.20	18	559.20	559.20	22	559.20
		50	530.10	530.10	150	530.10	530.10	60	530.10	530.10	13	530.10	530.10	19	530.10
	c	50				628.65	628.65	60	628.65	628.65	42	628.65	628.65	24	628.65
		50				566.52	566.52	60	566.52	566.52	25	566.52	566.52	22	566.52
Set 3	a	21	512.61	512.61	40	512.61	512.61	60	512.61	512.61	1	512.61	512.61	1	512.61
		32	669.99	669.99	78	669.99	669.99	60	669.99	669.99	6	669.99	669.99	5	669.99
	b	50	714.75	714.75	161	714.75	714.75	60	714.75	714.75	37	714.75	714.75	28	714.75
	c	50				668.40	668.40	60	668.40	668.93	70	668.40	668.40	27	668.40
Set 4	a	50				1526.86	1527.13	60	1527.55	1527.80	38	1526.87	1527.23	25	1526.86
		50				1415.66	1415.66	60	1415.85	1416.69	67	1415.70	1416.12	23	1415.65
		50				1317.34	1317.36	60	1317.48	1318.34	85	1317.34	1318.16	31	1317.33
	b	50	1524.70	1524.85	248	1515.61	1516.01	60	1516.27	1516.70	36	1515.61	1516.18	33	1515.60
		50	1377.85	1378.16	139	1375.68	1375.68	60	1375.79	1376.37	36	1375.68	1375.82	29	1375.67
		50	1300.33	1301.17	121	1299.88	1299.93	60	1299.96	1300.23	63	1299.91	1300.28	37	1299.87
Set 5	5_1	100	1058.27	1067.11	373	1057.58	1058.94	900	1058.26	1059.37	953	1056.75	1059.32	582	1056.74
	5_2	100	950.01	960.87	421	956.64	961.41	900	949.47	951.70	1452	951.45	955.21	557	946.82
	5_3	200	1357.14	1386.45	974	1357.79	1375.70	900	1340.87	1343.60	1500	1348.62	1355.42	900	1338.35
Set 6	A_50	50				639.64	639.64	60	640.06	640.29	155	639.64	639.89	35	639.64
	A_75	75				946.35	947.85	900	946.32	946.54	730	946.86	946.86	235	946.32
	A_100	100				1147.34	1150.05	900	1146.18	1147.05	1216	1146.32	1147.31	560	1146.18
	B_50	50				826.48	826.48	60	826.48	826.59	141	826.59	826.59	33	826.48
	B_75	75				1461.54	1461.94	900	1461.29	1461.33	578	1461.29	1461.53	363	1461.29
	B_100	100				1738.57	1741.67	900	1734.02	1736.80	1322	1733.92	1736.53	712	1733.36
Avg.			900.78	904.65	227	983.53	984.83	295	982.45	983.01	344	982.72	983.51	172	982.06

	ALNS		LNS		VNS		NS-SC	
	Avg. %	Best %	Avg. %	Best %	Avg. %	Best %	Avg. %	Best %
Set 2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Set 3	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00
Set 4a			0.01	0.00	0.07	0.02	0.04	0.00
Set 4b	0.30	0.26	0.01	0.00	0.05	0.02	0.03	0.00
Set 5	2.00	0.63	1.51	0.86	0.39	0.20	0.80	0.42
Set 6 A			0.16	0.04	0.06	0.02	0.07	0.02
Set 6 B			0.17	0.11	0.07	0.01	0.11	0.03
Overall	1.27	1.20	0.18	0.09	0.08	0.03	0.10	0.04

Table 3.3 – Summary of average and best gaps on 2E-VRP benchmarks.

best known solutions at least one time out of five. Instances in Set 4, while not bigger than some instances of Sets 2 and 3, are more difficult to solve due to customer distribution. The LNS algorithm is the only one that finds the current best known solutions for all the instances in Set 4. Our "NS-SC" only misses four of them, and still achieves high quality solutions with gaps less than 0.04%. Instances of Set 5 are the largest of the literature and those where the gaps and the runtimes are more important. On these instances, all the algorithms fail to achieve the best known solutions for several instances, mainly due to the bigger numbers of customers ans satellites that constitute the instances. The ALNS and the LNS can achieve an average relative gap of 2.00% and 1.51%, respectively. The VNS achieves an average relative gap of 0.39%, but is slower than the other algorithms. Our algorithm, on the other hand, offers good compromise between solutions quality and runtime, as it achieves an average relative gap of 0.80% while being significantly faster than both the LNS and the VNS. It was also able to improve the current best known solutions for a total of seven instances from Set 5 during our experiments. Only Breunig et al. [39] and Wang et al. [267] report results on the instances of Set 6. The LNS, the VNS, and our NS-SC are all able to obtain very low average relative gaps on both Set 6a and Set 6b, but once again our algorithm has a smaller runtime. Overall, our algorithm is able to achieve the current best known solutions for 216 out of 234 instances with an overall average relative gap of 0.10% and running times smaller than those of the literature. Based on these results, our approach is very effective in solving the 2E-VRP.

3.6 Conclusions

In this work, we presented a hybrid heuristic for the 2E-VRP. The algorithm uses an effective neighborhood search to explore the solution space and discover high quality

solutions. By keeping trace of the exploration steps, the heuristic generates a set of routes which are then recombined using an integer programming model. Solving this model serves as way to find better solutions that were missed by the neighborhood search procedure and to faster lead the algorithm towards promising regions of the solution space. Computational experiments on the standard benchmark instances demonstrate the competitiveness of our approach. Our algorithm consistently achieves high quality solutions with an overall average relative gap of 0.10%, while requiring less running time than other algorithms, and improves the current best known solutions for seven instances for which no optimal solution is known.

In summary, the results presented in this work are encouraging for the application of our approach to optimize other two-echelon routing problems. Its components can be adapted and additional ones can be integrated to account for different constraints. Future work will primarily focus on the extension of the algorithm to variants of the 2E-VRP and similar routing problems, mainly to accommodate more practical constraints and more realistic cost structures.

A Branch-&-Cut Algorithm for the Orienteering Problem with Hotel Selection

The Orienteering Problem with Hotel Selection (OPHS) is a recent extension of the orienteering problem (OP) where intermediate facilities, called "hotels" are introduced. Given a number of days D, a set of hotels and a set of POIs, the goal of the OPHS is to find a tour of D connected trips, that visits a subset of POIs and maximizes the total collected profit. Each trip must start and end at one of the available hotels. In this chapter, we present a new mathematical formulation of the problem several valid inequalities to enhance it. We also introduce a fast heuristic based on the *order-first split-second* approach for the OPHS, and use it to generate lower bounds for the exact algorithm to solve our model. Finally, we conduct several computational experiments to evaluate the performance of our method on benchmark instances of the literature. Results demonstrate the effectiveness of our approach.

Introduction

The Orienteering Problem with Hotel Selection (OPHS) is a recent extension of the orienteering problem (OP) where intermediate facilities, called "hotels" are introduced. In the OPHS, we consider set of N points of interest (POIs) and a set of H hotels. To each POI we assign a non-negative score, while hotels have no scores. The time need to travel from point i ot point j (whether hotel or POI) is known. It is supposed fixed and verifies the triangle inequality. Given a planning horizon of D days and a fixed time budget for each day, the goal of the OPHS is to determine a *tour* that maximizes the total collected profit from visiting a subset of POIs. The profit from visiting a POI can be collected at most once. The tour is composed of D connected *trips*, one for each day. A *trip* is an

ordered list of POIs that starts and ends at one of the available hotels. The departure and the arrival of a trip are not necessarily the same. The departure hotel and the arriving hotel of the *tour* are fixed in advance, and can be used as intermediate hotels during the tour.

Several real-life applications can be modeled using an OPHS: a traveling salesperson planning a multiple day business trip to visit potential clients who needs to select which clients to visit and must choose appropriately where to spend the night at the end of each day, the design of a multi-day tourist trip across an attractive region, or the planning of submarine surveillance activities [85].

In this chapter, we address the exact solution of the OPHS using a Branch-&-Cut approach. In Section 4.1, we provide a comprehensive review of solution methods for the OPHS, as well as a review on some related problems. In Section 4.2, we introduce a new integer linear programming model for the OPHS and in Section 4.3 we present several valid inequalities to reinforce it. Then, in Section 4.4 we describe a fast heuristic to solve the problem based on the *order-first split-second* approach, and introduce novel splitting procedure specific to the OPHS. The overall structure of our exact algorithm is described in Section 4.5. Experimental results on literature benchmarks are presented in Section 4.6 and Section 4.8 concludes the chapter.

4.1 Related work

The OPHS was introduced by Divsalar et al. [85] who, presented an integer linear programming model for the problem and a Skewed Variable Neighborhood Search (SVNS). The SVNS is comprised of two phases: an *initialization phase*, and an *improvement phase*. In the *initialization*, the algorithm builds a list of feasible hotel sequences which, will be used to construct solutions during the improvement phase. To this effect, the SVNS constructs the list of all feasible hotel sequences, calculates a potential score for each pair of hotels by solving an OP with a simple local search method, and then uses that score to sort the list of feasible sequences and keep only a subset of it. Afterwards, it constructs the initial solution. During the *improvement phase*, the SVNS tries to identify an optimal combination of hotels while simultaneously looking for the best selection of POIs for a given combination of hotels. The improvement phases relies on two "shaking" functions: one for the POIs and one for the hotels, and a local search algorithm composed of nine local search moves. Furthermore, the authors introduced 229 benchmark instances of varying

sizes to evaluate their algorithm. These instances are generated based on instances for the OP, and 224 of them have known optimal solutions. The performance of the SVNS depends on the total number of feasible hotel combinations and produces good solutions in a reasonable time when the number of hotels and the number of trips are small.

Later, Divsalar et al. [86] proposed a Memetic Algorithm for the OPHS. Like the SVNS, the MA is also structured into two main parts: *the initialization* and the *main-loop*. During the initialization, the MA computes a potential score for each pair of hotels by solving an OP and generates an initial population. To generate the initial population, the algorithm composes feasible hotel sequences starting from the departure and selecting the successive hotels in a probabilistic fashion. Afterwards, it inserts POIs into each trip of every feasible sequence to obtain complete OPHS solutions. The *main-loop* aims at improving the solutions. It includes two crossover operators and one mutation operator that are used to populate a solution pool by generating new solutions with new feasible hotel sequences. The population is then updated by selecting individuals from the pool. While they follow the same two steps, the MA and the SVNS differ in how they deal with feasible hotel sequences. In the SVNS all the feasible hotel sequences are created in advance when, in the MA new sequences are obtained by the genetic operators by combining the previously created sequences. This causes the MA to be more efficient that the SVNS and perform better when the total number of feasible sequences gets larger. To evaluate their MA, the authors also introduced 176 more complex instances with known optimal solutions.

The OPHS is closely related to the Traveling Salesperson Problem with Hotel Selection (TSPHS) which, was originally discussed by Vansteenwegen et al. [258]. Like in the OPHS, in the TSPHS, the salesperson has to visit a number of customers, they should select a hotel at the end of each day to rest, and continue their visits the next day starting from the same hotel. However, unlike in the OPHS, the salesperson must visit all their customers and the number of days allowed for the visits is not limited. The objective of the problem becomes to, first, minimize the number of days needed to visit all the customers, and second, to minimize the total travel length.

To the best of our knowledge, most of the research on the TSPHS focuses on heuristic solution methods. Vansteenwegen et al. [258] proposed a two-index formulation of the TSPHS, together with a set of benchmark instances of varying sizes. Their formulation can be optimally solved for instance with up to 40 customers. Additionally, the authors solve the TSPHS using a heuristic algorithm that relies on two initialization methods and an improvement phase. The first initialization method generates an initial solution based

on the "nearest neighbor" principle, while the second starts by solving a TSP over the set of customers and, then, inserts hotels into the resulting tour to make it a feasible TSPHS solution. The improvement phase consists of several neighborhood from the literature as well as search operators designed specifically for the TSPHS. Castro et al. [44] modified the formulation of Vansteenwegen et al. [258] by using a weighted objective function instead of the lexicographical ordering of the two objectives, and the Dantzig–Fulkerson–Johnson sub-tour elimination constraints instead of the Miller–Tucker–Zemlin constraints. They also proposed a Memetic Algorithm (MA) with an embedded Tabu Search (TS) to solve the problem. In their framework, the Memetic Algorithm is concerned with the hotel selection decision, while the routing decision is delegated to the Tabu Search. The MA uses a population that comprises individuals made only of hotel sequences, and a one-point crossover operator based on hotel exchange. It generates TSPHS tours by using two construction heuristics, one that inserts customers into a hotel sequence and another one based on the "Order-First Split-Second" principle. The TS is composed of several local search operators, on of which is specifically designed for the problem. The MA of Castro et al. [44] achieves very competitive results compared to other algorithms of the literature. It is, however, rather complex and can be slow on large instances. Castro et al. [43] introduced a Set-partitioning formulation and a fast heuristic algorithm for the TSPHS. Their approach uses an "Order-First Split-Second" method to generate an initial solution which, is then improved using a Variable Neighborhood Descent (VND) embedded into an Iterated Local Search (ILS). The VND uses several neighborhood search operators, some of which focus on customer routing and others on hotel selection. Baltz et al. [24] introduced the Traveling Salesperson Problem with Multiple Time Windows and Hotel Selection (TSP-MTWHS) for which, they proposed a Mixed Integer Linear Programming model and a randomized heuristic. More Recently, Lu et al. [174] presented a hybrid metaheuristic that combines a Memetic Algorithm with dynamic programming to solve the TSPHS. Their hybrid MA is built on the use of three dedicated crossover operators for solution recombination, an adaptive rule for crossover selection, and a two-phase local search procedure which alternates between feasible and infeasible search spaces. It also uses a dynamic programming approach to find an optimal hotel sequence for a given TSP tour and transform it into a TSPHS solution. This DP algorithm is used during the initialization phase which, is similar to that of Castro et al. [44], and at the end of the local search procedure to re-optimize the hotel sequence of the solution.

In Vehicle Routing problems with Intermediate Facilities (VRP-IF), the constraint that requires routes to start and end at the same depot is relaxed. Thus, similar to the OPHS, they allow routes to be split into trips that may start from and end at a different facility.

Several problems involving intermediate facilities (IFs) have been studied in the literature, among which the Periodic Vehicle Routing Problem with Intermediate Facilities (PVRP-IF) and the Vehicle Routing Problem with Intermediate Replenishment Facilities (VRP-IF), Even though these problems present similarities to the OPHS, they also differ from it in several ways. There are three differences with the OPHS that are common to all these problems: unlike in the OPHS, all the customer vertices have to be served, the number of trips is not limited, and instead of maximizing a score, the objective of these problems is the minimization of the total cost (usually, the travel cost) of the vehicle routes.

In the PVRP-IF [4, 131], a set of customers have to be visited one or several times over a time horizon using a fleet of vehicles. Each vehicle leaves the depot, serves a subset of customers, and when its work shift is over, returns to the depot. Intermediate facilities are made available so that vehicles can stop and renew their capacity when it is reached, to avoid returning to the depot. The vehicles may visit the intermediate facilities as many times as needed. A vehicle route in the PVRP-IF corresponds to a tour in the OPHS and is divided into "simple routes" which correspond to OPHS trips. Compared to the OPHS, where each trip is limited by a time budget, in the PVRP-IF, the time limit is applied on the whole tour and the number of customers that can be visited during each trip is only limited by vehicle capacity.

The Vehicle Routing Problem with Intermediate Replenishment Facilities (VRPIRF) [241] was first introduced by Crevier et al. [70] under the name "Multi-Depot Vehicle Routing Problem with Inter-Depot Routes". In the VRPIRF, intermediate facilities act as replenishment depots along vehicle routes. It is supposed that the total demand of the customer set exceeds the total capacity of the vehicle fleet. Thus, during their route, vehicles may need to stop at intermediate facilities to renew their load, in order to be able to satisfy customer demands. Similar to the PVRP-IF, the number of customers visited in the parts of the vehicle route between two intermediate facilities is limited by vehicle capacity, and an upper bound is imposed on the duration of the whole route. A VRPIRF route corresponds to an OPHS tour, and the part of route between two consecutive facilities is called a "route segment". Unlike the OPHS, the total duration of routes must not exceed an upper-bound. Furthermore, the duration of "routes segments" is not explicitly limited, but vehicle capacities imply a certain limitation. Solution methods for the problem were proposed by Crevier et al. [70], Tarantilis et al. [241] and Hemmelmayr et al. [131]

Finally, the Waste Collection Vehicle Routing Problem with Time Windows (WCVRPTW) [149, 29] is a variant of the VRP-IF where customer nodes represent

sites at which waste is collected and intermediate facilities are used as disposal sites. In the WCVRPTW, empty vehicles depart from the central depot, collect the waste from the customer sites, and when full, visit a disposal site to be emptied. At the end of their work shift, the vehicles must return empty to the depot. Compared to VRP-IF, the WCVRPTW considers further practical constraints, namely: time windows, routing time limit per vehicle, vehicle capacity, route capacity, and driver breaks. The route capacity includes maximum number of stops and lifts, and maximum volume and weight that a driver can handle per day.

4.2 Problem description

Given a non-empty set $V_h = \{0, \ldots, H-1\}$ of $H \geq 2$ hotels, and a set $V_c = \{H, \ldots, H+n-1\}$ of n points of interest (POIs), the OPHS is defined on a complete graph $G = (V, A)$ where $V = V_h \cup V_c$, and $A = \{(i,j) : i \neq j,\ i,j \in V\}$. Each POI $i \in V_c$ is associated with a non-negative profit (score) p_i. Each arc $(i,j) \in A$ is associated with a non-negative travel cost $c_{i,j}$. Travel costs are assumed symmetric and satisfy the triangle inequality. The departure and the arrival depots are represented by hotels 0 and 1, while the remaining hotels are called "extra hotels". A "trip" designates a sequence of POIs starting and ending at an available hotel. The length of each trip $d \in [1, D]$ is limited by a given time budget L_d. A "tour" is an ordered set of connected trips (*i.e.* each trip starts where the previous one ends) of which, the first starts at the departure hotel, and the last ends at the arrival hotel. Both the departure and the arrival can be used as intermediate hotels during the tour. Since there is no limit on the number of visits to a hotel, a hotel can be selected more than once during the tour. Thus, a tour is not necessarily a single cycle. The objective of the OPHS is to determine a tour of connected trips that maximizes the sum of collected profits where each POI is visited at most once and the time budget of each trip is respected.

Using the above notation, the OPHS can be formulated as an Integer Linear Program (ILP). Let y_i^d be a binary decision variable that takes the value 1 if POI i is visited in trip d, and 0 otherwise. Let w_h^d be a binary decision variable that takes the value 1 if trip d ends at hotel h, and 0 otherwise. Because two successive trips end and start at the same hotel, variables w_h^d also indicate the starting hotel for trip $d+1$. Thus, we use variables w_h^0 to indicate if a hotel h is used as the departure hotel for trip 1. Finally, let $x_{i,j}^d$ be a binary decision variable which takes the value 1 if, in trip d, a visit to vertex i (hotel or POI) is followed by a visit to vertex j, 0 otherwise. We propose the following mathematical model

$$\max\ z = \sum_{d=1}^{D} \sum_{i \in V_c} p_i y_i^d \tag{4.1}$$

$$\text{s.t.} \sum_{d=1}^{D} y_i^d \leq 1 \quad , \quad \forall i \in V_c \tag{4.2}$$

$$\sum_{j \in V} x_{i,j}^d = y_i^d \quad , \quad \forall i \in V_c,\ d \in [1, D] \tag{4.3}$$

$$\sum_{j \in V} x_{j,i}^d = y_i^d \quad , \quad \forall i \in V_c,\ d \in [1, D] \tag{4.4}$$

$$\sum_{h \in V_h} w_h^d = 1 \quad , \quad \forall\ d \in [0, D] \tag{4.5}$$

$$\sum_{i \in V} x_{i,h}^d = w_h^d \quad , \quad \forall h \in V_h,\ d \in [1, D] \tag{4.6}$$

$$\sum_{i \in V} x_{h,i}^d = w_h^{d-1} \quad , \quad \forall h \in V_h,\ d \in [1, D] \tag{4.7}$$

$$w_0^0 = 1 \tag{4.8}$$

$$w_1^D = 1 \tag{4.9}$$

$$\sum_{i \in V} \sum_{j \in V} c_{i,j} x_{i,j}^d \leq L_d \quad , \quad \forall d \in [1, D] \tag{4.10}$$

$$\sum_{i \in U} \sum_{j \in U \setminus \{i\}} x_{i,j}^d \leq |U| - 1 \quad , \quad \forall U \subseteq V_c,\ |U| \geq 2,\ d \in [1, D] \tag{4.11}$$

$$y_i^d \in \{0, 1\}, \quad \forall i \in V_c,\ d \in [1, D] \tag{4.12}$$

$$w_h^d \in \{0, 1\}, \quad \forall h \in V_h,\ d \in [1, D-1] \tag{4.13}$$

$$x_{i,j}^d \in \{0, 1\}, \quad \forall i \in V,\ j \in V,\ d \in [1, D] \tag{4.14}$$

for the OPHS, denoted hereafter *ILP*:

The objective function (4.1) maximizes the sum of collected profits from the visited POIs. Constraints (4.2) impose that each POI is visited at most once. Constraints (4.3) and (4.4) ensure the connectivity of each trip. Constraints (4.5) guarantee that each trip starts and ends at one of the available hotels. Constraints (4.6) and (4.7) ensure tour connectivity; that is, if a trip ends at a given hotel h, the following trip starts at the same hotel. Constraints (4.8) and (4.9) ensure that the tour starts at the departure hotel and ends at the the arrival hotel. The upper bound on the length of each trip is imposed by constraints (4.10). Constraints (4.11) are the classical Dantzig–Fulkerson–Johnson subtour elimination constraints. Note that the subsets U in constraints (4.11) only involve POI vertices, since the trips in an OPHS solution might be cycles that start and end at the same hotel. Constraints (4.12), (4.13) and (4.14) define the value domains for the decision

variables.

Enumerating all the subtour elimination constraints produces a formulation with an exponential number of constraints. In practice, when solving routing problems, the problem is first relaxed by removing the subtour elimination constraints from the model, and then introducing them back into the model only when needed.

Constraints (4.11) can be replaced with the stronger *Generalized Subtour Elimination Constraints* (GSECs) [157] which, enhance both the elimination of specific subtours and the connectivity in the solution. GSECs were first applied to solve the Orienteering Problem (OP) by Fischetti et al. [98] and were later used to solve the Team Orienteering Problem (TOP) by El-Hajj et al. [94].

Given a subset S of POIs, we note $\delta(S)$ the set of arcs in A with exactly one end-vertex in S, and $\gamma(S)$ the set of arcs with both end-vertices in S. The GSECs are formulated as follows:

$$\sum_{(u,v)\in\delta(U)} x^d_{u,v} \geq 2y^d_i \quad , \quad \forall U \subseteq V_c,\ |U| \geq 2,\ \forall i \in U,\ \forall d\ \in [1, D] \tag{4.15}$$

These constraints state that any set of at least two POIs has to be connected to its complement. Thus, they ensure that each visited POI i is reachable from one or two hotel vertices.

Furthermore, due to the degree constraints (4.3), (4.4), (4.7) and (4.8), the GSECs can be written as:

$$\sum_{(u,v)\in\gamma(U)} x^d_{u,v} \leq \sum_{i\in U} y^d_i - y^d_j \quad , \quad \forall U \subseteq V_c,\ |U| \geq 2,\ \forall j \in U,\ \forall d\ \in [1, D] \tag{4.16}$$

$$\sum_{(u,v)\in\gamma(S)} x^d_{u,v} \leq \sum_{i\in S\cap V_c} y^d_i - y^d_j + 1 \quad , \quad \forall S \subseteq V, S \cap V_h \neq \emptyset,\ |S| \geq 2,\ \forall j \in V_c \setminus S,\ \forall d\ \in [1, D] \tag{4.17}$$

Constraints (4.16) impose an upper bound on the number of internal arcs of the subtour U to avoid the formation of a cycle. On the other hand, constraints (4.17) state that if a POI is visited outside of the main trip S, one arc of the trip has to be removed in order to connect the external POI.

4.3 Valid inequalities

4.3.1 Irrelevant components

Finding and removing POIs that cannot be visited in any feasible solution is a simple way of reducing the size of the problem search space. We deem a POI i "unreachable" during day d if, given any pair of hotels $(h_1, h_2) \in V_h^2$, the travel cost of the trip (h_1, i, h_2) exceeds the allotted time budget L_d.

This reasoning can be extended to reduce the number of available arcs by defining "irrelevant" arcs. An arc $(i, j) \in V^2$ is "irrelevant" to day d, if the travel cost of trip (h_1, i, j, h_2), is greater than L_d for every pair of hotels $(h_1, h_2) \in V_h^2$. Unreachable POIs and irrelevant arcs are removed from the problem by adding the following constraints, where $\mathcal{D}^-$ is the set of days where POI i (resp. arc (i, j)) is unreachable (resp. irrelevant).

$$\sum_{d \in \mathcal{D}^-} y_i^d = 0 \tag{4.18}$$

$$\sum_{d \in \mathcal{D}^-} x_{i,j}^d = 0 \tag{4.19}$$

4.3.2 Bounds on trip profits

El-Hajj et al. [94] proposed a set of valid inequalities for the Team Orienteering Problem (TOP) by imposing bounds on the characteristics of a subset of tours, mainly the collected profit and the number of visited POIs. They derive smaller instances from the original problem and either solve or bind them to gain useful information for the construction of the tours of the original problem.

Let $\mathcal{T} = \bigcup_{n=1}^{D-1} \{(d_1, d_2, \ldots, d_n) \mid d_i \in [1, D],\ d_{i+1} = d_i + 1\}$ be the set of all tuples of at most $D-1$ successive trips. For each $T \in \mathcal{T}$, let $\bar{T}$ be the set of trips $d \in [1, D]$ such that $d \notin T$. For each instance X of the problem and $T \in \mathcal{T}$, we note X^T the instance derived by only keeping the trips of T and relaxing the constraints (4.8) and (4.9) of the base model. Let $LB(X)$ be a lower bound of a given instance X and $UB(X)$ its upper bound. Clearly, constraints (4.20) are valid inequalities for the OPHS. The total collected profit of any subset of successive trips cannot exceed the upper bound of an instance constituted of only this trips with no constraints on the departure nor the arrival.

$$\sum_{d \in T} \sum_{i \in V_c} p_i y_i^d \leq UB(X^T) \quad , \quad \forall T \ \subset \mathcal{T} \tag{4.20}$$

The use of $UB(X^T)$ can be extended to impose a lower bound on the profit collected during the tours in $\bar{T}$; that is the profit collected outside of T. Indeed, given a lower bound $LB(X)$ of the problem, the total profit collected during the tours in T and the tours in $\bar{T}$ has to be greater or equal to $LB(X)$. Thus, given an upper bound on the profit collected in T, we can deduce a lower bound on the profit that has to be collected in $\bar{T}$. This can be expressed as follows.

$$\sum_{d \in \bar{T}} \sum_{i \in V_c} p_i y_i^d \geq LB(X) - UB(X^T) \quad , \quad \forall \ T \subset \mathcal{T} \tag{4.21}$$

To compute the upper and lower bounds required in inequalities (4.20) and (4.21), we start by solving the instances X^T such that T contains only one trip, i.e. $T \in \{(d) \mid d \in [1, D]\}$. The obtained bounds are then used in cuts to solve instances containing more trips: to solve X^T and $T = (d_1, \ldots, d_n)$ with $n < D$, we first solve $X^{T'}$ with $T' = (d_1, \ldots, d_{n-1})$.

4.3.3 Bounds on the number of POIs per trip

Similarly to profits, it is also possible to bound the number of visited POIs in each trip. To find an upper bound UB_{POI} on the number of visited POIs in a subset of trips, we first derive an instance X^T_{min} from the previously defined instance X^T, by setting the profit of each POI to $p_{min} = \min\{p_i, i \in V_c\}$. We then modify our model to maximize the number of visited customers while satisfying constraints (4.2) to (4.7), (4.10) to (**??**), (4.20), and (4.21). The resulting upper bound is used to bound the number of POIs.

Next, in order to get a lower bound LB_{POI} on the number of POIs per trip, we consider the one trip instances X^T such that $T \in \{(d) \mid d \in [1, D]\}$ and, for each one of them, minimize the number of visited POIs while satisfying constraints (4.2) to (4.7), (4.10) to (4.14), (4.20), and (4.21). Note that these inequalities are limited to single trips only and do not concern subsets of more than one trip. Furthermore, for the purpose of computing LB_{POI}, constraints (4.21) are rewritten as:

$$\sum_{d \in T} \sum_{i \in V_c} p_i y_i^d \geq LB(X) - UB(X^{\bar{T}}) \quad , \quad \forall\ T \subset \mathcal{T} \tag{4.22}$$

The obtained values of $LB_{POI}(X^T)$ and $UB_{POI}(X^T)$ are then used in the following valid inequalities to restrict the number of POIs visited in each trip or subset of successive trips:

$$\sum_{d \in T} \sum_{i \in V_c} y_i^d \leq UB_{POI}(X^T_{min}) \quad , \quad \forall T \ \subset \mathcal{T} \tag{4.23}$$

$$\sum_{i \in V_c} y_i^d \geq LB_{POI}(X^{(d)}) \quad , \quad \forall d \in [1, D] \tag{4.24}$$

To compute the upper bounds on the number of visited POIs, we proceed in the same manner as with the bounds on trip profits, i.e., solving the instances X^T that contain a single trip only, and then using the obtained bounds to solve instances containing more trips.

4.3.4 Incompatibility cuts

Incompatibility cuts stem from the observation that if two given locations, either hotels or POIs, are located too far away from one another, they are very unlikely to belong into the same trip, due to time budget considerations. This observation can be extended to pairs of arcs (if traversing two given arcs takes too much time, they are unlikely to be used in the same trip), and eventually to pairs of arcs and vertices. In the following, we only consider incompatibilities between vertices, incompatibilities between arcs, and to some extent, incompatibilities between arcs and hotels.

More formally, we consider two components of an instance X, arcs or vertices, *incompatible during day d* if they cannot be together in the corresponding trip d, in any optimal solution of the problem. Based on this, we define an incompatibility graph between vertices $\mathcal{G}_V^d(V, Inc_V^d)$, and another one for incompatibilities between arcs $\mathcal{G}_A^d(A, Inc_A^d)$, for each day $d \in [1, D]$. The arc sets of these graphs are defined as:

1. $Inc_V^d = \{(i,j) \mid i,j \in V,\ UB(X[i \overset{d}{\sim} j]) < LB(X)\}$;
2. $Inc_A^d = \{((i_1,j_1),(i_2,j_2)) \mid (i_1,j_1),(i_2,j_2) \in A,\ UB(X[(i_1,j_1) \overset{d}{\sim} (i_2,j_2)]) < LB(X)\}$;

where $X[i \overset{d}{\sim} j]$ denotes an instance of the modified problem where i and j must both be visited during day d, and $X[(i_1,j_1) \overset{d}{\sim} (i_2,j_2)]$ denotes an instance of the modified problem where arcs (i_1,j_1) and (i_2,j_2) must be traversed both during day d.

Finally, note that the construction of the full set of incompatibilities can be difficult, depending on the instance. Nevertheless, it is still possible to initialize the incompatibility graphs using some simple rules, and afterwards, as the solution process progresses, add new edges using the previous definition of the graphs and relying on new cuts.

4.3.4.1 Simple incompatibilities

We can easily find incompatibilities between hotels and POIs by following the same reasoning used in Section 4.3.1:

r1. If the travel cost between a pair of hotels (h_1,h_2) is greater than the time budget of a given day d, then the two hotels cannot be selected in the corresponding trip.

r2. If the travel cost between a POI i and a hotel h is greater than the time budget for a given day d, then i cannot be visited in a trip where hotel h is used as either the departure or the arrival.

r3. Given a pair of POIs (i,j), if for every pair of hotels (h_1,h_2), the travel cost of the trip (h_1,i,j,h_2) exceeds the allotted time budget, i and j cannot be visited during the same trip.

r4. Given a pair of arcs $u = (i_1,j_1)$ and $v = (i_2,j_2)$, if for every pair of hotels (h_1,h_2), the cost of the shortest path from h_1 to h_2 containing u and v exceeds the time budget, then u and v cannot be used together during the corresponding trip.

r5. If i and j are two incompatible vertices, then arcs $u \in (\{i\} \times V) \cup (V \times \{i\})$ and $v \in (\{j\} \times V) \cup (V \times \{j\})$ cannot be used together in one trip.

The above properties are mainly used to initialize the incompatibility graphs $\mathcal{G}_V^d$ and $\mathcal{G}_A^d$ which, then, will be used to derive incompatibility cuts as explained in Section 4.3.4.2.

However, we can also use them to formulate simpler inequalities to express incompatibilities between hotels, POIs and hotels, and arcs and hotels as follows.

$$w_{h_1}^{d-1} + w_{h_2}^{d} \leq 1 \quad , \quad \forall (h_1, h_2) \in V_h^2,\ d \in [1, D], \quad c_{h_1,h_2} > L_d \tag{4.25}$$

$$y_i^d \leq 1 - w_h^d \quad , \quad \forall i \in V_c,\ \forall h \in V_h,\ d \in [1, D], \quad c_{i,h} > L_d \tag{4.26}$$

$$y_i^d \leq 1 - w_h^{d-1} \quad , \quad \forall i \in V_c,\ \forall h \in V_h,\ d \in [1, D], \quad c_{i,h} > L_d \tag{4.27}$$

$$x_{i,j}^d \leq 1 - w_h^d \quad , \quad \forall i \in V,\ \forall j \in V,\ \forall h \in V_h,\ d \in [1, D], \quad c_{h,i} + c_{i,j} > L_d \tag{4.28}$$

$$x_{i,j}^d \leq 1 - w_h^{d-1} \quad , \quad \forall i \in V,\ \forall j \in V,\ \forall h \in V_h,\ d \in [1, D], \quad c_{i,j} + c_{j,h} > L_d \tag{4.29}$$

Constraints (4.25) express incompatibilities between hotels, constraints (4.26) and (4.27) express incompatibilities between a hotel and POI, and constraints (4.28) and (4.29) are for incompatibilities between arcs and hotels. Note that the incompatibilities between pairs of POIs, resp. arcs, are not expressed here; the reason being that inequalities of the form

$$y_i^d + y_j^d \leq 1 \quad , \quad \forall (i, j) \in Inc_V^d, d \in [1, D] \tag{4.30}$$

are dominated by the *clique cuts* described in Section 4.3.4.2. Thus we use those cuts to implement these incompatibilities.

Furthermore, the previous incompatibility constraints can be extended to express incompatibilities between arcs or POIs and pairs of hotels. Given a pair of hotels (h_1, h_2), if the total distance from $h1$ to a POI i and from i to $h2$ is greater than time budget, then i cannot be visited in a trip where the two hotels are used as departure and arrival respectively. Likewise, in the case of an arc, we simply need to consider the travel cost from the h_1 to the first extremity of the arc, the arc's cost, and the travel cost from the second extremity of the arc to h_2. The resulting constraints are expressed as follows.

$$y_i^d \leq 2 - (w_{h_1}^{d-1} + w_{h_2}^{d}) \quad , \quad \forall i \in V,\ \forall (h_1, h_2) \in V_h^2,\ d \in [1, D], \quad c_{h_1,i} + c_{i,h_2} > L_d \tag{4.31}$$

$$x_{i,j}^d \leq 2 - (w_{h_1}^{d-1} + w_{h_2}^{d}) \quad , \quad \forall i \in V,\ \forall (h_1, h_2) \in V_h^2,\ d \in [1, D], \quad c_{h_1,i} + c_{i,j} + c_{j,h_2} > L_d \tag{4.32}$$

Finally, it would be possible to establish incompatibilities between POIs or between arcs with regards to a fixed hotel or pair of hotels. However, we choose not to, because of the additional computational effort that it requires.

4.3.4.2 Clique cuts

As stated before, incompatibility cuts between vertices and between arcs are computed by extracting *cliques* from the incompatibility graphs $\mathcal{G}_V^d$ and $\mathcal{G}_A^d$. A clique is a subset of vertices of an undirected graph such that every two vertices in the clique are adjacent. Thus, a clique of $\mathcal{G}_V^d$ (resp. $\mathcal{G}_A^d$) is a subset of vertices (resp. arcs) which are pairwise incompatible, and therefore, visiting any vertex (resp. traveling through an arc) of the clique during a given trip excludes the remaining elements from being visited (resp. traversed) during that same trip; that is, a trip can only contain one element of the clique. Clique cuts are formulated using the following inequalities, where K and Q represent clique of $\mathcal{G}_V^d$ and $\mathcal{G}_A^d$, respectively.

$$\sum_{i \in K} y_i^d + \sum_{h \in K} w_h^d \leq 1 \quad , \quad \forall d \in [1, D] \tag{4.33}$$

$$\sum_{(u,v) \in Q} x_{i,j}^d \leq 1 \quad , \quad \forall d \in [1, D] \tag{4.34}$$

Because they produce stronger cuts, large and maximal cliques are more desirable for inequalities (4.33) and (4.34). A clique is said to be maximal if it cannot be extended by including more adjacent vertices, and a maximal clique is maximum if it is the largest clique in the graph. In general, the number of maximal cliques in a graph is an exponential function of the number of its vertices, and finding the maximum clique is an NP-Hard problem. However, efficient algorithms to find maximal cliques or a subset of them are available in the literature. The details concerning the generation of cliques are discussed in Section 4.5.

4.3.5 Equivalence breaking cuts

In the OPHS, the trips that form a solution are asymmetric: they have each a different time budget, and different departures and arrivals. However, given an OPHS tour, it is possible to derive an equivalent tour with the same objective value while maintaining the order of visits of POIs, by simply shifting the first POIs from one trip to the last positions of the preceding one and vice versa, as long as the time limit constraint is satisfied. Fig. 4.1 shows an example of such equivalent tours.

In order to remove these equivalent solutions from the search space, we only consider tours where it is not possible to shift the first customer of any trip to the end of the trip

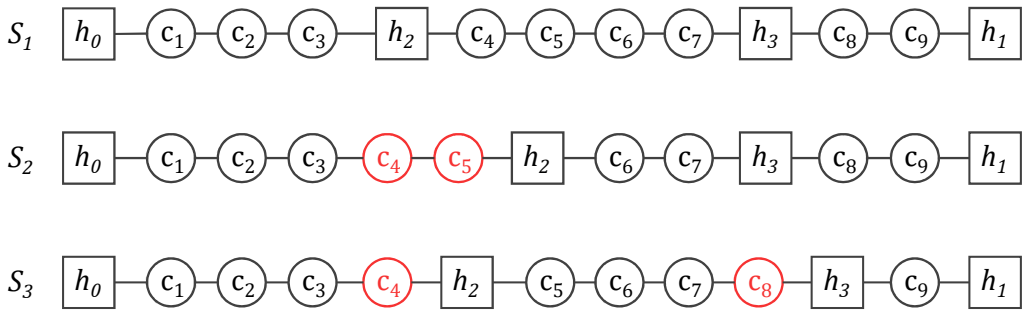


Figure 4.1 – Equivalent OPHS tours.

preceding it. This is achieved by adding the following constraints to our model.

$$\begin{aligned} &\sum_{(h_1,i)\in V_h\times V_c} c_{h_1,i}x^d_{h_1,i} + \sum_{(i,j)\in V_c^2} c_{i,j}x^d_{i,j} \\ &+ \sum_{(i,h_2)\in V_c\times V_h} c_{i,v}x^d_{i,h_2} + \sum_{h_2\in V_h} c_{v,h_2}x^{d+1}_{h_2,v} \\ &\qquad > L_d \sum_{h_2\in V_h} x^{d+1}_{h_2,v} \quad , \quad \forall v \in V_c,\ \forall d \in [1, D-1] \end{aligned} \tag{4.35}$$

4.4 Computing an initial solution

To obtain an initial feasible solution we have implemented a heuristic approach for the OPHS. Algorithm 5 details the steps of this heuristic. The algorithm is a Multi-Start Heuristic (MS-OPHS) which performs $i_{restart}$ iterations in total. At each iteration, the heuristic generates a random permutation π of POIs from which, it extracts a solution using an *order-first split-second* algorithm. Afterwards the extracted solution is improved using a local search procedure. It starts by removing d randomly chosen POIs from the solution, with d a random number in $[1, d_{max}]$. Then it applies the $2-Opt$ operator on the partial solution to decrease the duration of each trip. After that, new POIs are inserted in the solution using a Best Insertion Heuristic (BIH). The implemented BIH constructs a feasible solution by successively inserting POIs in their best possible position. At each iteration, the procedure evaluates all the feasible insertions of unrouted POIs and selects the insertion with the minimum cost $cost(u) = Shift_u/(p_u)^2$ where, $Shift_u = min(c_{i,u} + c_{u,j} - c_{i,j})$, i and j being two consecutive nodes in the solution. The local search then repeats until it reaches $iter_{max}$ iterations without improving the current solution. In the following, we present a new splitting procedure, specific for the OPHS.

Algorithm 5: Multi-start Local Search for the OPHS.

Data: S_{best} empty solution
V_h and V_c vectors of h hotels and n POIs, respectively.
Result: S_{best} best solution found

```
1  begin
2      for i ← 0 to i_restarts do
3          π ← RandomPermutation(V_c)
4          S ← Split(π, V_h)
5          while iter < iter_max do                    // Local search loop
6              S' ← S
7              d ← rand(1, d_dmax)
8              RandomReomval(S', d)
9              Apply2Opt(S')
10             U ← getUnroutedPOIs(V, S')
11             ApplyBestInsertion(S', U)
12             if f(S') ≥ f(S) then
13                 S ← S'
14                 iter ← 0
15             else
16                 iter ← iter + 1
17         if f(S) ≥ f(S_best) then
18             S_best ← S
19     return S_best
```

4.4.1 Splitting algorithm for the OPHS

In our approach, a giant tour is a permutation $\Pi = (\pi_1, \pi_2, ..., \pi_n)$ of all the accessible POIs in V_c with no route delimiters and no hotels. The aim of the splitting procedure is to select which sub-sequence of POIs to visit during each trip, and to select the hotels to use between each trip, in order to maximize the total collected profit, while satisfying time budget constraints. This problem can be reduced to a longest path problem on an auxiliary directed acyclic graph $G_{aux} = (V_{aux}, A_{aux})$ defined as follows. The set of nodes is $V_{aux} = \{v_{i,h,p} \mid i \in [1, n], h \in V_h, p \in [1, D]\} \cup \{d, a\}$ where, each node $v_{i,h,p}$ represents a visit to POI π_i after leaving hotel h on day p. The nodes d and a are dummy nodes that act as the departure and the arrival of the shortest path.

The construction of the arc set A_{aux} is based on the definition of *saturated trips*. Starting from position i of Π, the corresponding *saturated trip* is maximum length trip obtained by including all the subsequent customers as long as all the time budget constraints are satisfied, or until the end of the tour is reached. As such, POIs that remain unrouted after

splitting can only be located between saturated trips in Π. The splitting problem consists in identifying D sub-sequences in Π and the hotels connecting them, such that the total collected profit is maximized. We can show that for any permutation Π, there exists an optimal solution for this problem where all the trips are saturated. This is formalized in Proposition 4.1. Therefore, the optimal splitting can be found by considering only saturated trips.

Proposition 4.1. *For any instance of the OPHS where D is number of days, for any sequence Π of POIs of this instance, and for any optimal solution S_π to the splitting problem, there exists a solution S'_π where all the trips are saturated and such that S_π and S'_π have the same profit.*

Proof. Let $\Pi = (\pi_1, \pi_2, ..., \pi_n)$ be a sequence of OPHS POIs. We note $\langle i, l_i \rangle_\Pi$ a sub-sequence of Π of length l_i starting from position i, i.e. $(\pi_i, \pi_{i+1}, ..., \pi_{i+l_i})$. Let $S_\Pi = (\langle i_1, l_{i_1} \rangle_\Pi, \ldots, \langle i_D, l_{i_D} \rangle_\Pi)$ be the optimal solution of the splitting problem of Π. Suppose $\langle i_k, l_{i_k} \rangle_\Pi \in S_\Pi$ an unsaturated trip.

- If $\langle i_{k+1}, l_{i_{k+1}} \rangle_\Pi$ and $\langle i_k, l_{i_k} \rangle_\Pi$ are such that $i_{k+1} > i_k + l_{i_k} + 1$, i.e. there are unrouted customers between them, then it is possible to extend $\langle i_k, l_{i_k} \rangle_\Pi$ by at least one customer and increase the collected profit. Thus S_Π is not optimal.

- Else, if $\langle i_k, l_{i_k} \rangle_\Pi$ and $\langle i_{k+1}, l_{i_{k+1}} \rangle_\Pi$ are such that $i_{k+1} = i_k + l_{i_k} + 1$, it is possible to derive a solution S'_Π with the same profit as S_Π by shifting POIs from the start of $\langle i_{k+1}, l_{i_{k+1}} \rangle_\Pi$ to $\langle i_k, l_{i_k} \rangle_\Pi$ until it becomes saturated. This is possible because the distances verify the triangle inequalities. Consequently, the obtained solution $S_\Pi = (\langle i_1, l_{i_1} \rangle_\Pi, \ldots, \langle i_k, l'_{i_k} \rangle_\Pi, \langle i'_{k+1}, l_{i'_{k+1}} \rangle_\Pi, \ldots, \langle i_D, l_{i_D} \rangle_\Pi)$ is optimal. If $\langle i'_{k+1}, l_{i'_{k+1}} \rangle_\Pi$ not saturated, we apply the same logic to it. By repeating this process, we can replace each trip of a solution by a saturated one, until we derive an optimal solution where all the trips are saturated.

□

To construct the set of arc A_{aux}, for each POI π_i in Π, we identify a saturated route $r_i(h_1, h_2, p)$ for every possible pair of hotels (h_1, h_2), and for each day $p \in [1, D]$. Each arc of the graph is associated with a weight equal to the profit of the trip it represents. Thus, A_{aux} is constructed as follows:

- Arc $(v_{i,h_1,p}, v_{j,h_2,p+1})$ with $i < j$ represent a saturated trip $r_i(h_1, h_2, p)$. It is included in A_{aux} if and only if POI j is not part of $r_i(h_1, h_2, p)$. The weight of these arcs is equal to the total profit of the saturated route they represent.
- Arcs $(v_{i,h_1,p}, v_{i,h_2,p+1})$ with $h_1 \neq h_2$, represent an empty trip that starts at hotel h_1 and ends at h_2. Their weight is equal to 0.
- Arcs $(d, v_{i,1,0})$ are used to select the POI in Π from which the visits start. They have a weight equal to 0.
- Arcs $(v_{i,h_1,D}, a)$ represent the last saturated trip $r_i(h_1, 1, D)$ that ends at the final hotel (hotel 1). Their weight is equal to the profit of the saturated trip.

Figure 4.2 depicts the auxiliary graph associated with a giant tour $\Pi = \{1, 2, \ldots, n\}$, in an instance with 2 hotels and 3 days. The nodes $v_{i,h,d}$ corresponding to visits to a same customer i are regrouped within the same gray rectangle. Note that only a subset of arcs are represented in order not to overload the illustration.

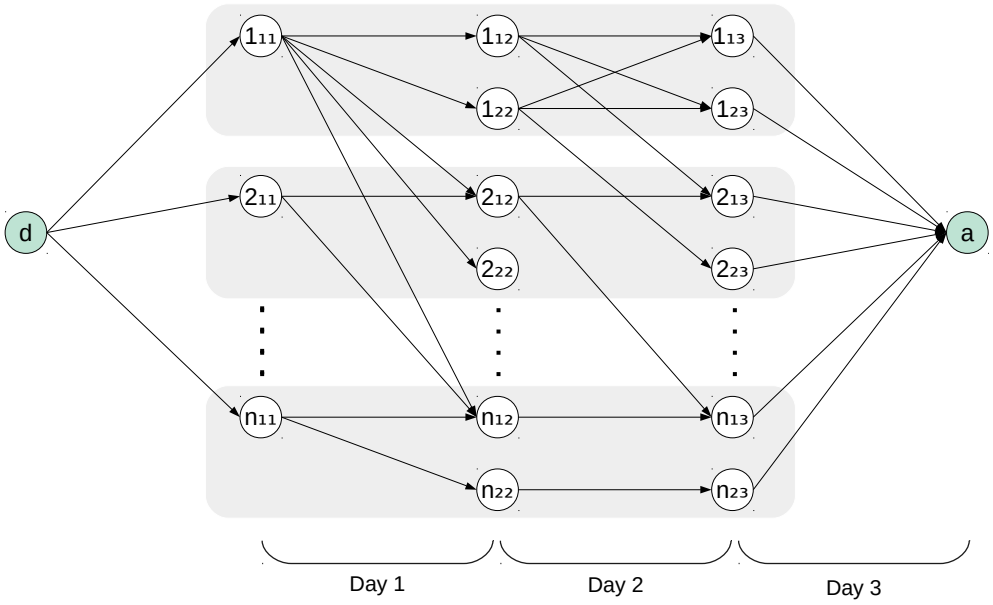


Figure 4.2 – Auxiliary graph for a permutation of n POIs to be split into 3 trips using 2 hotels.

An OPHS solution corresponds to a path from node d to a in G_{aux}. Thus, the solution with maximum profit corresponds to the longest path between d and a. The auxiliary graph G_{aux} is acyclic by construction. Moreover, it is layered in such a way that every path from d to a contains exactly $D + 1$ arcs, and that no customer can be visited twice in the same path. As such, the longest path can be computed in $\mathcal{O}(n^2m^2D)$ complexity using Bellman's algorithm for directed acyclic graphs.

4.5 Overall algorithm

Our mathematical model is solved by means of a Branch-&-Cut algorithm where GSECs are separated. At the beginning, the model contains constraints (4.2)-(4.10), constraints (4.12)-(4.14) and some initial cuts. During the pre-processing phase, the irrelevant components are detected and removed from the model. Then, an initial feasible solution is computed using the heuristic described in Section 4.4. Afterwards, the cuts described above, namely incompatibility cuts, equivalence breaking cuts and bounds on scores and on visited POIs are generated and stored in a *cut pool* to allow for faster separation during the solution process. For the incompatibility cuts, we first generate the simple constraints (i.e. constraints (4.25) to (4.29)) and construct the incompatibility graphs for the clique cuts, then we determine for each node $i \in V$ a maximal clique that contains it, using the metaheuristic of Rossi et al. [211]. The generation of bounds on trip scores and visited POIs is achieved by using the same Branch-&-Cut algorithm to solve the associated sub-problems, but without including the bound-based cuts, and with a time limitation to keep the pre-processing phase short.

Our algorithm uses two different separation procedures, one for the GSECs and the other for the remaining cuts. The separation of GSECs is exclusively performed on candidate solutions in integral nodes. When a candidate solution is detected, the algorithm checks if it contains any subtour. If it does, it is rejected and the violated GSECs are added to the model. To identify subtours in integral solutions, this procedure looks for strong components in the directed graph. Recall that a digraph is strongly connected if, for every pair of vertices (i, j), it contains a path from i to j. A strongly connected component of a digraph is a subset of its vertices such that the induced subgraph is strongly connected. As such, subtours in integral solutions correspond to non-trivial strongly connected components. As for the remaining cuts, our separation strategy is very simple, and consists in checking if any of the cuts in the *cut pool* are violated by the current LP relaxation. All violated inequalities found, regardless of their type, are added to the model.

In order to perform better, the overall algorithm needs to be able to detect "good" feasible solutions early. To this end, we use two heuristic procedures to derive a feasible solution from a relaxation of the problem. The first heuristic (H1) uses the linear programming relaxation of ILP to construct a solution to the problem. Given the fractional values of variables y_i^d and w_h^d, H1 proceeds one trip at a time. For each trip d, it constructs an OP instance by rounding the fractional values of each y_i^d variable to the nearest integer to the determine the list of potential POIs, and rounding the variables $w_{h_1}^{d-1}$ and $w_{h_2}^d$ such

that $w_{h_1}^{d-1} = \max_{h \in V_h}(w_h^{d-1})$ and $w_{h_2}^{d} = \max_{h \in V_h}(w_h^{d})$ to determine the departure and the arrival of the OP instance, respectively. Afterwards, it uses the best insertion heuristic (BIH) described in Section 4.4 to solve the resulting OP instance. In the end, the OP solutions are concatenated to form a solution for the OPHS.

The second heuristic (H2) works in a similar fashion, but is only applied on candidate solutions. Recall, that because the subtour elimination constraints are relaxed from ILP, candidate solutions might be infeasible due to the presence of subtours. If it is the case, H2 uses the integer values of variables y_i^d and w_h^d to construct OP instances for each trip d and then, solves each OP instance using a dynamic programming (DP) procedure [95]. Using DP within H1 is rather inconvenient, sees as DP is slower than BIH, especially in instances where the number of POIs per trip is significant. Furthermore, the OP instances resulting from candidate solutions are usually smaller.

4.6 Computational experiments

Our algorithm was implemented in C++ using the Standard Template Library (STL) and the IBM ILOG CPLEX 12.6.3 C++ API to solve the IP model, and compiled using the GNU GCC compiler in a Linux environment. All the experiments were conducted on a single thread on an Intel Xeon X7542 CPU at 2.67GHz processor. In the following, we present and discuss the results of our computational experiments that were carried out in order to evaluate the performance of our model relative to the state-of-the-art, and to assess the contribution of each of the proposed valid inequalities.

4.6.1 Benchmark instances

Our algorithm was tested on the OPHS benchmark instances of Divsalar et al. [85] and Divsalar et al. [86] which, are available online at `http://www.mech.kuleuven.be/en/cib/op`.

The OPHS benchmark instances are organized into five sets which were all created using the benchmark instances of Tsiligirides [252] and Chao et al. [45] for the Orienteering Problem. Table 4.1 summarizes the main characteristics of each set of instances. Column *nb.* reports the number of instances in the set, column D reports the number of trips in the instance, and columns $|V_h|$ and $[V_c|$ show the number of hotels and the number of POIs in the instance, respectively.

Instances	*nb.*	D	$\lvert V_h \rvert$	$\lvert V_c \rvert$
Set 1	35	2	3	$[30, 100]$
	35	3	4	$[30, 100]$
	35	4	5	$[30, 100]$
Set 2	35	3	7	$[30, 100]$
	35	4	8	$[30, 100]$
Set 3	22	4	12	$[62, 100]$
	22	5	14	$[62, 100]$
Set 4	5	2	5	$[98, 100]$
	5	3	5	$[98, 100]$
Set 5	22	4	14	$[62, 100]$
	22	4	17	$[62, 100]$
	22	5	12	$[62, 100]$
	22	5	17	$[62, 100]$
	22	6	12	$[62, 100]$
	22	6	14	$[62, 100]$
	22	6	17	$[62, 100]$
	13	8	17	$[98, 100]$
	9	10	17	$[98, 100]$

Table 4.1 – Characteristics of the OPHS benchmark instances.

The instances in sets 1, 2, 3, and 5 were all generated from OP instances for which optimal solutions are known, and in a way such that the optimal OPHS solution corresponds to the optimal OP solution. To create an OPHS instance with D trips, the authors start from an optimal solution to a regular OP instance. First, they divide the length of the solution by D to have an estimate of the time budget per trip, and divide the OP tour into D sub-paths such that the length of each sub-path, apart from the last one, is greater or equal to the estimated one. The hotels are then added to the instance as follows: (1) the departure hotel and the arrival hotel are inserted, respectively, at the location of the starting and the ending vertex of the optimal OP tour, (2) one intermediate hotel is inserted at the location of the last vertex of each sub-path apart from the last one, and (3) additional intermediate hotels are added at the location of randomly selected POIs that are not in the optimal OP path.

For the instances in Set 4, the authors used large OP instances with no known optimal solutions from Chao et al. [45]. They used the departure and the arrival of the instance as the departure hotel and the arrival hotel of the OPHS instance, and replaced three

randomly selected vertices with intermediate hotels. As for the trips, they imposed equal time budgets on every trip for each instance.

4.6.2 Evaluation of the lower bound heuristic

To evaluate the performance of our lower bound heuristic (MS-OPHS), we tested it on the OPHS benchmark and compare its results to the Memetic Algorithm of Divsalar et al. [86]. The comparison is made in terms of solution quality and average computation time. Table 4.2 reports the results obtained by our method (MS-OPHS) and those obtained by the MA of Divsalar on literature benchmarks. Our results were obtained through five independent runs of the MS-OPHS on each instance. The first half of the table shows the characteristics of the benchmark instances. Column "*Ex. hotels*" indicates the number of intermediary hotels in the instance, column "*Trips*" the number of trips, and column "*Nb. inst.*" the number of instances that compose the set. The second half of the table displays the results of each model. Column "Best%" displays the gap obtained during the best run of the algorithm, column "AVG%" the average gap of all the runs, and CPU indicates the average computation time in seconds. The gap here is the percentage difference between the obtained solution and the optimal solution. Note that for Set 4, the optimal solutions are not necessarily known, thus we use the best known solution. The gaps are calculated as follows: $gap = ((Optimal - Result)/Optimal) * 100$, where result is the value of the obtained solution.

The results show that our method performs worse than the MA, but that is understandable, given the simplicity of the MS-OPHS. However, its performance is rather satisfactory. On small instances (few hotels and trips), it is on par with MA, and it was even able to improve some best known solutions in Set 4.

4.6.3 Comparison between mathematical models

In this section, we present a comparison between our algorithm and exact methods available in the literature. As mentioned before, Divsalar et al. [85] are the only ones we know of, who proposed a mathematical formulation for the OPHS. However, they only tested their model on a small subset of instances. For the sake of comparison, we implemented their model and tested it on all the benchmark instances available in the literature. We then compared it with the base ILP model presented in Section 4.2.

Set of instances				MA [86]			MS-OPHS		
Name	Ex. hotels	Trips	Nb.inst.	Best %	AVG %	CPU	Best %	AVG %	CPU
Set 1	1	2	35	1.29	1.44	2.59	0.32	0.5	0.83
	2	3	35	0.43	0.88	1.84	0.48	1.27	1.17
	3	4	35	0.46	0.6	1.09	0.68	2.69	1.44
Set 2	5	3	35	0.41	0.81	1.41	0.35	1.23	1.39
	6	4	35	0.39	0.64	1.15	1.31	3.58	1.58
Set 3	10	4	22	0.96	1.72	6.59	4.54	6.56	3.56
	12	5	22	1.43	1.98	5.44	4.65	7.81	4.21
Set 4	3	2	10	0.76	0.76	1.17	-0.23	-0.12	2.28
Set 5	12	4	22	1.24	1.78	6.5	3.88	6.7	3.88
	15	4	22	1.32	1.92	6.63	3.89	6.52	2.04
	10	5	22	1.3	1.99	5.43	5.1	7.76	3.99
	15	5	22	1.4	2.22	5.41	4.65	7.81	4.21
	10	6	22	1.24	2.3	4.66	6.4	10.12	4.36
	12	6	22	1.63	2.28	3.49	6.51	10.3	4.77
	15	6	22	1.39	2.55	4.78	6.46	10.48	4.49
	15	8	13	2.95	3.66	5.16	12.83	16.15	4.36
	15	10	9	3.78	5.03	5.04	4.89	6.92	8.33
Avg. / Total	-	-	405	1.32	1.92	4.02	3.92	6.25	3.35

Table 4.2 – Comparison of the MS-OPHS with the MA of Divsalar et al. [86].

Table 4.3 shows the comparison between results obtained by the model of Divsalar et al. [85] and the results obtained by ours. The first half of the table shows the characteristics of the benchmark instances. Column "*Ex. hotels*" indicates the number of intermediary hotels in the instance, column "*Trips*" the number of trips, and column "*Nb. inst.*" the number of instances that compose the set. The second half of the table displays the results of each model. Column "$\#OPT$" displays the number of instances that were solved to optimality whithin a one hour time limit. Column "$GAP\%$" is the average gap between the best upper bound (UB) and the best lower bound (LB) obtained for unsolved instances. It is calculated as $GAP\% = 100 * (UB - LB)/UB$. Finally, column CPU reports the average solution time expressed in seconds.

The results presented in Table 4.3 show that our model performs better than that of Divsalar et al. [85]. It is able to solve much more instances to optimality and is also faster. Note that, the model of Divsalar et al. [85], when it cannot solve an instance, achieves a better GAP than our model. This is due to subtour elimination constraints. Indeed, when using MTZ constraints, the integral solutions obtained during the solution process are all feasible solutions for the OPHS. On the other hand, when using GSECs, integral solutions in the search tree might not be valid solutions, due to the presence of

subtours. As such, Divsalar's model has better chances of finding feasible lower bounds than our model. Hence, the better gaps. However, this flaw can be easily corrected by introducing an efficient repair heuristic in the Branch-&-Cut process.

Set of instances				Divsalar et al. [85]			Base model (ILP)		
Name	Ex. hotels	Trips	Nb.inst.	#OPT	GAP%	CPU	#OPT	GAP%	CPU
Set 1	1	2	35	20	5,69	360	35	0	430
	2	3	35	18	9,82	877	32	2,63	397
	3	4	35	13	11,25	789	28	8,11	657
Set 2	5	3	35	17	8,44	781	34	0,24	525
	6	4	35	14	12,02	1043	24	15,51	396
Set 3	10	4	22	1	31,09	2403	3	65,41	389
	12	5	22	0	46,58	-	2	80,05	966
Set 4	3	2	5	2	24,99	490	2	26,26	139
	3	3	5	0	25,56	-	0	73,05	-
Set 5	12	4	22	1	30,91	2302	4	51,92	1123
	15	4	22	0	34,21	-	4	48,26	1643
	10	5	22	0	48,56	-	1	82,51	1809
	15	5	22	0	44,1	-	1	88,74	924
	10	6	22	0	49,28	-	2	81,63	1401
	12	6	22	1	46,93	2048	3	81,36	737
	15	6	22	0	47,2	-	2	87,53	2016
	15	8	13	0	65,04	-	0	98,3	-
	15	10	9	0	83,52	-	0	99,79	-
Avg. / Total	-	-	405	87	34,73	1233	177	55,07	903

Table 4.3 – Comparison between the base ILP model and the model of Divsalar et al. [85].

4.6.4 Evaluation of components

To study the impact of the proposed valid inequalities, we conducted several experiments. During each of them, we solved the base model plus one set of valid inequalities. Table 4.4 shows the impact of the additional cuts added to the basic model. The table is devided into 5 similar parts of three columns each. Each part reports the number of instances being solved to optimality (#Opt) whithin a one hour time limit, the average gap between the best upper bound and the best lower bound (GAP%), and the average solution time expressed in seconds (CPU). Columns "Base model" displays the results obtained with the base model, columns "Heuristics" display the results obtained by using the heuristics discribed in Section 4.5, and each of the remaining columns shows the results obtained by using the model plus the heuristics and the set of valid inequalities displayed in the

header of the column.

Based on the results shown in Table 4.4, we notice a significant increase in the number of solved instances once the heuristics are included in the algorithm. The introduction of valid inequalities further improves the number of solved instances. However, some perform better than others depending on the set of instances. For example, on the instances of set 2 with 5 extra hotels and 3 trips, incompatibility cuts perform better than cuts based on bounds on the number of POIs. But, POI cuts perform better on instances of set 1 with 2 extra hotels and 3 trips. Overall, the results show that using the proposed heuristics and valid inequalities improves the performance of our algorithm.

4.7 Numerical results

In this section, we present the results obtained by our algorithm on the OPHS instances. Table 4.5 show the results of our algorithm on the OPHS instances. Columns "$|V_c|$" and "OPT" in these tables indicate the number of POIs of the instance, and the value of the optimal solution. Columns "LB" and "UB" show the best lower bound and the best upper bound obtained by our algorithm, respectively, and column "Gap" represents the gap between the lower and the upper bound. Column "CPU" displays the computation time needed to solve the instance. A dash "−" indicates that the instances could not solved within the one-hours time limit.

We observe that our algorithm is able to solve most of the instances with less than 6 extra hotels and less than 4 trips. However, it struggles with instances that contain more than 10 extra hotels or more than 4 trips. On the other hand, the number of POIs per instance does not seem to have much effect on the solution process, compared to the number of hotels and the number of trips which, affect the number of possible hotel permutations in an OPHS tour. These results appear to show that the latter has a greater impact on the difficulty of solving the OPHS than the number of POIs.

Furthermore, note that there are instances where the B-&-C could not solve an instance to optimality, while other configurations that incorporate less cuts are able to do so. This may be due to the fact that adding several cuts might slow down the algorithm due to the additional effort required for cut separation. Furthermore, we also noticed that solutions with higher objective values are not always good initial solutions, and that sometimes slightly worse solutions cause CPLEX to converge more rapidly.

Set of instances				Heuristics			Base model (ILP)			Equivalency cuts			Incompatibility Cuts			Score bounds			POI bounds		
Name	Ex.	hotels	Trips	Nb.inst	#OPT	GAP	CPU	#OPT	GAP	CPU	#OPT	GAP	CPU	#OPT	GAP	CPU	#OPT	GAP	CPU	#OPT	GAP	CPU
Set 1	1	2	35	35	0	546	31	0.31	430	35	0	426	35	0	407	35	0	571	35	0	603
	2	3	35	27	2.25	1006	28	0.42	656	33	0.84	550	31	1.66	420	32	0.54	515	33	0.77	570
	3	4	35	27	2.78	1196	23	1.4	1207	27	2.42	1131	31	1.79	546	28	2.18	1196	30	1.87	923
Set 2	5	3	35	31	1.18	815	27	0.7	608	31	0.97	708	32	0.97	439	29	1.62	885	30	1.69	797
	6	4	35	27	2.1	1209	23	1.07	1344	29	2.39	1155	28	2.81	606	26	2.27	1212	28	2.45	1196
Set 3	10	4	22	3	14.79	3068	3	9.05	3011	5	13.25	3081	5	13.28	2395	5	12.82	2988	5	13.18	2994
	12	5	22	2	17.67	3307	1	13.78	3229	2	17.12	3387	1	19.2	3096	2	17.12	3377	2	17.21	3376
Set 4	3	2	5	2	16.93	2184	2	24.36	2104	2	16.54	2215	2	15.34	2062	2	16.54	2201	2	16.56	2205
	3	3	5	0	24.3	3515	0	11.38	3419	0	17.63	3595	0	12.47	3412	0	18.01	3575	0	18.03	3575
Set 5	12	4	22	5	13.62	2912	2	10.23	3005	5	13.09	2953	5	12.75	2550	5	12.98	3103	5	12.64	3110
	15	4	22	5	12.98	2892	1	9.97	3108	5	12.72	2986	5	12.57	2570	4	13.13	3102	4	13.66	3135
	10	5	22	1	17.45	3372	0	11.94	3358	2	17.74	3406	3	17.86	2744	2	17.62	3435	3	17.07	3370
	15	5	22	1	18.76	3339	0	15.37	3222	2	17.77	3394	2	19.16	3197	2	17.93	3516	2	17.7	3454
	10	6	22	3	19.99	3165	0	20.45	3287	3	19.53	3322	3	20.76	3153	2	20.48	3328	3	18.79	3297
	12	6	22	3	21.56	3119	0	21.06	2922	3	20.14	3198	3	25.15	2956	3	21.32	3227	3	21	3291
	15	6	22	1	21.76	3321	0	20.48	3418	3	18.87	3343	2	23.2	3269	3	19.88	3397	3	19.96	3386
	15	8	13	0	34.58	3526	0	47.12	3579	0	30.78	3630	0	34.18	3711	0	32.09	3628	0	32.7	3632
	15	10	9	0	39.14	3516	0	50.11	3490	0	36.94	3639	0	40.15	3633	0	36.75	3625	0	36.59	3621
Avg. / Total	-	-	405	173	15.66	2556	141	14.96	2522	187	14.37	2562	188	15.18	2287	179	14.63	2605	188	14.55	2585

Table 4.4 – Impact of additional cuts.

Set of instances				Branch-&-Cut		
Name	Ex. hotels	Trips	Nb.inst.	#OPT	GAP%	CPU
Set 1	1	2	35	35	0	130
	2	3	35	31	0.31	289
	3	4	35	29	0.47	316
Set 2	5	3	35	30	0.24	256
	6	4	35	29	0.64	525
Set 3	10	4	22	6	6.9	809
	12	5	22	3	11.45	441
Set 4	3	2	5	2	22.74	50
	3	3	5	0	10	-
Set 5	12	4	22	5	7.94	984
	15	4	22	5	8.24	555
	10	5	22	5	10.68	1282
	15	5	22	4	10.85	1118
	10	6	22	5	12.2	1010
	12	6	22	4	13.58	892
	15	6	22	3	14.76	537
	15	8	13	0	27.88	-
	15	10	9	0	49.51	-
Avg. / Total	-	-	405	196	11.58	613

Table 4.5 – Results of the B-&-C algorithm on OPHS benchmarks.

4.8 Conclusion

The OPHS is a variant of the Orienteering Problem where intermediate facilities, called "hotels" are introduced. Given a number of days D, a set of hotels and a set of POIs, the goal of the OPHS is to find a tour of D connected trips, that visits a subset of POIs and maximizes the total collected profit. Each trip must start and end at one of the available hotels. In this chapter, we introduced a new mathematical model for the OPHS and an exact method to solve it. We presented several valid inequalities to reinforce our mathematical model and improve the performance of our algorithm, as well as a fast heuristic based on the *order-first split-second* approach. The results obtained on the OPHS benchmark instances show the effectiveness of our method.

Future work will focus on the improvement of our exact algorithm and the development of effective meta-heuristics based on our *order-fist, split-second approach*. The introduction of well-thought upper-bounds, branching rules, and cut management strategies can potentially lead to significant improvements in the performance of our exact algorithm. Decomposition approaches are also a direction to be explored. Additionally, several

extensions of the OPHS can be considered for future work, including time windows, service times at the POIs, scores for the hotels, visiting and lodging fees or even multi-modal variant of the problem.

Conclusion and future work

With the ever increasing urbanization and the growing trend that is e-commerce, the share of urban freight distribution in energy consumption and environmental pollution, is bound to increase in the future. In recent years, City Logistics and its various concepts, such as co-modality and collaborative distribution, emerged as viable solutions to mitigate the negative impacts of urban distribution. In this thesis, we took a closer look at how vehicle routing problems can contribute to the operation of City Logistics systems, and propose several innovative methods to solve selective and multi-echelon variants of the vehicle routing problem.

We started Chapter 1 with an overview on freight distribution in urban areas, its importance, and its challenges with regards to environmental issues. We then presented City Logistics as potential source of solutions to the challenges of urban transportation. We focused on concepts such as co-modality: what it is, what it aims for and how it is implemented in various contexts. Throughout this part of the chapter, we gave examples on how vehicle routing problems contribute to City Logistics. In the second part of the chapter, we presented in more details the Vehicle Routing Problem. We gave an overview of solution methods that were developed to solve VRPs, and discussed variants of the problem, especially selective variants.

In Chapter 2, we presented an effective meta-heuristic for the Team Orienteering Problem with Time Windows (TOPTW). The TOPTW is a selective variant of the VRP where the goal is to plan a set of routes over a subset of locations, that maximizes profits while taking into consideration time limitations on the routes and customer availability. Our algorithm is based on a neighborhood search that alternates between a *route search space*, and a *giant tour search space* to explore the solution space without being limited by time windows and length constraints. In order to benefit from information gathered while exploring the search space, our method integrates an adaptive memory mechanism that allows to use local optima to build high quality solutions. Computational results on literature benchmark shown the competitiveness of our approach. In particular, it is able to find the current best-known solutions, or better ones, for 94% of the benchmark

instances within reasonable runtimes, and was able to find new best solutions for 61 instances for which no optimal solution has yet been found.

After the TOPTW, we tackled the Two-Echelon Vehicle Routing Problem (2E-VRP), a variant of the VRP where, freight from a *main depot* is delivered to final customers using intermediate facilities, called *satellites*. The 2E-VRP is a variant that developed in the context of City Logistics. To solve the 2E-VRP, we designed a hybrid algorithm that combines heuristic solution methods with mathematical programming. It relies on a neighborhood search heuristic to explore the search space and generate a set of promising routes. These routes are then combined into high quality solution using a set covering formulation of the 2E-VRP. This aims to identify high quality solutions that might have been missed by the neighborhood search procedure, and lead the algorithm towards promising regions of the solution space. This approach proved very effective, as demonstrated by computational experiments on the standard benchmark instances. It consistently achieves high quality solutions, on par with state of the art-algorithms, yet requires significantly less computational time.

The OPHS is a routing problem that shares aspects with both the Orienteering Problem and the 2E-VRP. In the OPHS, given a number of days D, a set of intermediate facilities called "hotels", and a set of "Points Of Interest", the goal is to find a maximal profit tour of D connected trips which, must start and end at one of the available hotels, and visit each POI at most once. In Chapter 4, we presented a new mathematical model for the OPHS with an exponential number of constraints and solved it using an exact method. We presented several valid inequalities to enhance the performance of our mathematical model, as well as heuristics to obtain feasible solution from the LP relaxation of the problem. Furthermore, we introduced a fast multi-start heuristic for the OPHS based on the *order-first split-second* approach. To that effect, we designed a split procedure specifically for the OPHS. The results obtained on the OPHS benchmark instances show the effectiveness of our method.

The contributions of this thesis and the empirical findings they led to are very encouraging and open up multiple research perspectives. At first, we should solidify our work on the solution of the OPHS. Several ways to improve our exact method can be investigated. The introduction of additional valid inequalities, whether adaptations of cuts designed for similar problems or cuts specific to the OPHS is planned. Similarly, studying ways to reduce the size of the problem, either through effective pre-processing procedures or by using decomposition algorithms should be done. We observed that the number of hotel combinations might have a bigger effect on difficulty than the number of nodes

in the instance, as such, extensive experiments should be carried out to confirm it. If it is confirmed, design choices to improve the algorithm need to take that into account. Furthermore, we plan on proposing a new more effective meta-heuristic approach for the OPHS. The results obtained by both our simple OPHS heuristic using a splitting procedure, and the performance of our MS-ILS in solving the TOPTW encourage us to explore this avenue.

Regarding the 2E-VRP, our solution approach, while very competitive, still has room for improvement. Solving the set covering formulation during the recombination step is very time consuming, and needs to be sped up. In our implementation, we used CPLEX to solve the MILP as is, but using ideas from column generation methods is a viable way to reduce the computational effort necessary to find the optimal solution of the partial model, or at least to significantly improve the upper bound. Using an effective heuristic approach to solve the set covering problem instead of an exact algorithm, and using more elaborate strategies for pool management are also ideas worth exploring. Furthermore, in order to improve solution quality, more diversification is needed in the adaptive memory. The introduction of new components adapted for route generation and better pool management strategies might allow us to achieve that.

In our numerous computational experiments, we noticed that different components of the algorithm have different impacts on the solution process, depending on the instance being treated. A single component could improve the performance of the algorithm on one instance of the problem, and degrade it on another one. For example, the route recombination component in our hybrid algorithm for the 2E-VRP improves the performance of the algorithm on large instances like those of set 5, but rarely finds improvements when used on smaller instances, yet still constitutes a major overhead in computation time. Thus, knowing which characteristics of the instance influence the effect of a component or even if a parameter can be a determinant factor in improving the design and the performance of our heuristics. By using these characteristics, it would be possible to allow algorithms to automatically adapt to each instance, on the spot, by considering or discarding components to suit its specifics. Identifying the features that characterize an instance of the problem to predict which heuristic suits them best, constitutes another relevant direction for future work.

Finally, future work should also include the extension of our algorithms to other more general variants of the routing problems studied herein. These so-called *Rich VRPs* or *Multi-attribute VRPs* consider more complex constraints and objectives arising in real life applications. The consideration of environmental costs, risks and uncertainty,

varying travel times, pick-up and delivery practices, transfers and transshipments, and synchronisation are all of relevance to the optimization of City Logistics.

References

[1] Aleman, R. E. and Hill, R. R. (2010). A tabu search with vocabulary building approach for the vehicle routing problem with split demands. *International Journal of Metaheuristics*, 1(1):55–80.

[2] ALICE / ERTRAC Urban mobility WG (2014). Urban freight research roadmap. Available at: `https://www.smartcities.at/assets/01-Foerderungen/SUL/SUL-infocorner/ERTRAC-ALICE-WG5-Urban-freight-Research-Innovation-roadmap-DEF.pdf`(accessed 15 October 2019).

[3] Allen, J., Thorne, G., and Browne, M. (2007). Bestufs good practice guide on urban freight transport. *BESTUFS EU Thematic Network, Brussels*.

[4] Angelelli, E. and Speranza, M. G. (2002). The periodic vehicle routing problem with intermediate facilities. *European Journal of Operational Research*, 137(2):233 – 247. Graphs and Scheduling.

[5] Archetti, C., Bianchessi, N., and Speranza, M. (2013). Optimal solutions for routing problems with profits. *Discrete Applied Mathematics*, 161(4):547 – 557. Seventh International Conference on Graphs and Optimization 2010.

[6] Archetti, C., Bianchessi, N., and Speranza, M. G. (2014a). Branch-and-cut algorithms for the split delivery vehicle routing problem. *European Journal of Operational Research*, 238(3):685 – 698.

[7] Archetti, C., Hertz, A., and Speranza, M. G. (2007). Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1):49–76.

[8] Archetti, C., Ángel Corberán, Plana, I., Sanchis, J. M., and Speranza, M. G. (2015). A matheuristic for the team orienteering arc routing problem. *European Journal of Operational Research*, 245(2):392 – 401.

[9] Archetti, C., Savelsbergh, M. W. P., and Speranza, M. G. (2006). Worst-case analysis for split delivery vehicle routing problems. *Transportation Science*, 40(2):226–234.

[10] Archetti, C. and Speranza, M. G. (2012). Vehicle routing problems with split deliveries. *International Transactions in Operational Research*, 19(1-2):3–22.

[11] Archetti, C. and Speranza, M. G. (2014). *Arc routing problems with profits*, pages 257–284. MOS-SIAM Series on Optimization. SIAM, Philadelphia.

[12] Archetti, C., Speranza, M. G., and Savelsbergh, M. W. P. (2008). An optimization-based heuristic for the split delivery vehicle routing problem. *Transportation Science*, 42(1):22–31.

[13] Archetti, C., Speranza, M. G., and Vigo, D. (2014b). *Chapter 10: Vehicle Routing Problems with Profits*, chapter 10, pages 273–297. Number 18 in MOS-SIAM Series on Optimization. SIAM, Philadelphia.

[14] Awerbuch, B., Azar, Y., Blum, A., and Vempala, S. (1998). New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM Journal on Computing*, 28(1):254–262.

[15] Balas, E. (1989). The prize collecting traveling salesman problem. *Networks*, 19(6):621–636.

[16] Balas, E. (1995). The prize collecting traveling salesman problem: Ii. polyhedral results. *Networks*, 25(4):199–216.

[17] Balas, E. (2007). *The Prize Collecting Traveling Salesman Problem and its Applications*, pages 663–695. Springer US, Boston, MA.

[18] Baldacci, R., Hadjiconstantinou, E., and Mingozzi, A. (2004). An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, 52(5):723–738.

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

[20] Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283.

[21] Baldacci, R., Mingozzi, A., Roberti, R., and Calvo, R. W. (2013). An exact algorithm for the two-echelon capacitated vehicle routing problem. *Operations research*, 61(2):298–314.

[22] Baldacci, R., Toth, P., and Vigo, D. (2007). Recent advances in vehicle routing exact algorithms. *4OR*, 5(4):269–298.

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

[24] Baltz, A., Ouali, M. E., Jäger, G., Sauerland, V., and Srivastav, A. (2015). Exact and heuristic algorithms for the travelling salesman problem with multiple time windows and hotel selection. *Journal of the Operational Research Society*, 66(4):615–626.

[25] Beasley, J. E. (1983). Route first—cluster second methods for vehicle routing. *Omega*, 11(4):403 – 408.

[26] Belenguer, J. M., Martinez, M. C., and Mota, E. (2000). A lower bound for the split delivery vehicle routing problem. *Operations Research*, 48(5):801–810.

[27] Bell, J. E. and McMullen, P. R. (2004). Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18(1):41–48.

[28] Ben-Said, A., El-Hajj, R., and Moukrim, A. (2019). A variable space search heuristic for the capacitated team orienteering problem. *Journal of Heuristics*, 25(2):273–303.

[29] Benjamin, A. and Beasley, J. (2010). Metaheuristics for the waste collection vehicle routing problem with time windows, driver rest period and multiple disposal facilities. *Computers & Operations Research*, 37(12):2270 – 2280.

[30] Berbotto, L., García, S., and Nogales, F. J. (2014). A randomized granular tabu search heuristic for the split delivery vehicle routing problem. *Annals of Operations Research*, 222(1):153–173.

[31] Blocho, M. and Czech, Z. J. (2013). A parallel memetic algorithm for the vehicle routing problem with time windows. In *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 144–151.

[32] Boudia, M., Prins, C., and Reghioui, M. (2007). An effective memetic algorithm with population management for the split delivery vehicle routing problem. In *Hybrid Metaheuristics. HM 2007. Lecture Notes on Computer Science*, volume 4771, pages 16–30, Berlin, Heidelberg. Springer Berlin Heidelberg.

[33] Bouly, H., Dang, D.-C., and Moukrim, A. (2010a). A memetic algorithm for the team orienteering problem. *4OR*, 8(1):49–70.

[34] Bouly, H., Dang, D.-C., and Moukrim, A. (2010b). A memetic algorithm for the team orienteering problem. *4OR*, 8(1):49–70.

[35] Boussier, S., Feillet, D., and Gendreau, M. (2007). An exact algorithm for team orienteering problems. *4OR*, 5(3):211–230.

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

[37] Bräysy, O. and Gendreau, M. (2005a). Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118.

[38] Bräysy, O. and Gendreau, M. (2005b). Vehicle routing problem with time windows, Part II: Metaheuristics. *Transportation Science*, 39(1):119–139.

[39] Breunig, U., Schmid, V., Hartl, R. F., and Vidal, T. (2016). A large neighbourhood based heuristic for two-echelon routing problems. *Computers and Operations Research*, 76:208–225.

[40] Bullnheimer, B., Hartl, R. F., and Strauss, C. (1999). An improved ant System algorithm for thevehicle Routing Problem. *Annals of Operations Research*, 89:319–328.

[41] Butt, S. E. and Cavalier, T. M. (1994). A heuristic for the multiple tour maximum collection problem. *Computers & Operations Research*, 21(1):101 – 111.

[42] Butt, S. E. and Ryan, D. M. (1999). An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers & Operations Research*, 26(4):427 – 441.

[43] Castro, M., Sörensen, K., Vansteenwegen, P., and Goos, P. (2015). A fast metaheuristic for the travelling salesperson problem with hotel selection. *4OR*, 13(1):15–34.

[44] Castro, M., Sörensen, K., Vansteenwegen, P., and Goos, P. (2013). A memetic algorithm for the travelling salesperson problem with hotel selection. *Computers & Operations Research*, 40(7):1716 – 1728.

[45] Chao, I.-M., Golden, B., and Wasil, E. A. (1996a). The team orienteering problem. *European Journal of Operational Research*, 88:464–474.

[46] Chao, I.-M., Golden, B. L., and Wasil, E. A. (1996b). A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3):475 – 489.

[47] Chao, I.-M., Golden, B. L., and Wasil, E. A. (1996c). The team orienteering problem. *European Journal of Operational Research*, 88(3):464 – 474.

[48] Chen, P., Golden, B., Wang, X., and Wasil, E. (2017). A novel approach to solve the split delivery vehicle routing problem. *International Transactions in Operational Research*, 24(1-2):27–41.

[49] Christofides, N. and Eilon, S. (1969). An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, 20(3):309–318.

[50] Christofides, N., Mingozzi, A., and Toth, P. (1981a). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical programming*, 20(1):255–282.

[51] Christofides, N., Mingozzi, A., and Toth, P. (1981b). State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11(2):145–164.

[52] CITEPA (2019).

[53] Civitas, W. T. (2015). Policy note - smart choices for cities. making urban freight logistics more sustainable.

[54] Clarke, G. and Wright, J. W. (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations research*, 12(4):568–581.

[55] Commission of the European Communities (2006). Keep Europe moving: sustainable mobility for our continent: mid-term review of the European Commission's 2001 Transport White Paper.

[56] Commission of the European Communities (2007). Green Paper - Towards a new culture for urban mobility. *European Economy*.

[57] Contardo, C. and Martinelli, R. (2014). A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129 – 146.

[58] Cordeau, J. F., Gendreau, M., and Laporte, G. (1997a). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30:105–119.

[59] Cordeau, J.-F., Gendreau, M., and Laporte, G. (1997b). A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119.

[60] Cordeau, J.-F., Laporte, G., and Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52(8):928–936.

[61] Cordeau, J.-F., Laporte, G., Savelsbergh, M. W., and Vigo, D. (2007). Chapter 6 vehicle routing. In *Transportation*, volume 14 of *Handbooks in Operations Research and Management Science*, pages 367 – 428. Elsevier.

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

[63] Crainic, T., Mancini, S., Perboli, G., and Tadei, R. (2011a). A GRASP with Path-Relinking Metaheuristic for the Two-Echelon Vehicle Routing Problem. Technical report, Cirrelt, Montreal.

[64] Crainic, T. G., Mancini, S., Perboli, G., and Tadei, R. (2008). Clustering-based heuristics for the two-echelon vehicle routing problem. *CIRRELT-2008-46*.

[65] Crainic, T. G., Mancini, S., Perboli, G., and Tadei, R. (2011b). Multi-start heuristics for the two-echelon vehicle routing problem. In Merz, P. and Hao, J.-K., editors, *Evolutionary Computation in Combinatorial Optimization*, pages 179–190, Berlin, Heidelberg. Springer Berlin Heidelberg.

[66] Crainic, T. G., Mancini, S., Perboli, G., and Tadei, R. (2012). Impact of Generalized Travel Costs on Satellite Location in the Two-Echelon Vehicle Routing Problem. *Procedia - Social and Behavioral Sciences*, 39:195–204.

[67] Crainic, T. G., Perboli, G., Mancini, S., and Tadei, R. (2010). Two-Echelon Vehicle Routing Problem: A satellite location analysis. *Procedia - Social and Behavioral Sciences*, 2(3):5944–5955.

[68] Crainic, T. G., Ricciardi, N., and Storchi, G. (2004). Advanced freight transportation systems for congested urban areas. *Transportation Research Part C: Emerging Technologies*, 12(2):119–137.

[69] Crainic, T. G., Ricciardi, N., and Storchi, G. (2009). Models for evaluating and planning city logistics systems. *Transportation Science*, 43(4):432–454.

[70] Crevier, B., Cordeau, J.-F., and Laporte, G. (2007). The multi-depot vehicle routing problem with inter-depot routes. *European Journal of Operational Research*, 176(2):756 – 773.

[71] Cruijssen, F., Bräysy, O., Dullaert, W., Fleuren, H., and Salomon, M. (2007). Joint route planning under varying market conditions. *International Journal of Physical Distribution & Logistics Management*, 37(4):287–304.

[72] Cura, T. (2014). An artificial bee colony algorithm approach for the team orienteering problem with time windows. *Computers & Industrial Engineering*, 74:270 – 290.

[73] Dablanc, L. (2009). Freight transport for development toolkit: Urban freight - freight transport, a key for the new urban economy. *World Bank, Department for International Development*.

[74] Dahl, S. and Derigs, U. (2011). Cooperative planning in express carrier networks — an empirical study on the effectiveness of a real-time decision support system. *Decision Support Systems*, 51(3):620 – 626.

[75] Dang, D.-C., El-Hajj, R., and Moukrim, A. (2013a). A branch-and-cut algorithm for solving the team orienteering problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 332–339, Berlin, Heidelberg. Springer Berlin Heidelberg.

[76] Dang, D.-C., Guibadj, R. N., and Moukrim, A. (2011). A pso-based memetic algorithm for the team orienteering problem. In *Applications of Evolutionary Computation*, pages 471–480, Berlin, Heidelberg. Springer Berlin Heidelberg.

[77] Dang, D.-C., Guibadj, R. N., and Moukrim, A. (2013b). An effective pso-inspired algorithm for the team orienteering problem. *European Journal of Operational Research*, 229(2):332–344.

[78] Dantzig, G., Fulkerson, R., and Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410.

[79] Defryn, C., Sörensen, K., and Cornelissens, T. (2016). The selective vehicle routing problem in a collaborative environment. *European Journal of Operational Research*, 250(2):400 – 411.

[80] Dell'Amico, M., Maffioli, F., and Värbrand, P. (1995). On prize-collecting tours and the asymmetric travelling salesman problem. *International Transactions in Operational Research*, 2(3):297–308.

[81] Derigs, U., Li, B., and Vogel, U. (2010). Local search-based metaheuristics for the split delivery vehicle routing problem. *Journal of the Operational Research Society*, 61(9):1356–1364.

[82] Desaulniers, G., Lessard, F., and Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404.

[83] Desaulniers, G., Madsen, O. B., and Ropke, S. (2014). *Chapter 5: The Vehicle Routing Problem with Time Windows*, chapter 5, pages 119–159. Number 18 in MOS-SIAM Series on Optimization. SIAM, Philadelphia.

[84] Desrosiers, J., Dumas, Y., and Soumis, F. (1986). A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4):301–325.

[85] Divsalar, A., Vansteenwegen, P., and Cattrysse, D. (2013). A variable neighborhood search method for the orienteering problem with hotel selection. *International Journal of Production Economics*, 145(1):150 – 160.

[86] Divsalar, A., Vansteenwegen, P., Sörensen, K., and Cattrysse, D. (2014). A memetic algorithm for the orienteering problem with hotel selection. *European Journal of Operational Research*, 237(1):29 – 49.

[87] Dorigo, M. and Stützle, T. (2019). *Ant Colony Optimization: Overview and Recent Advances*, pages 311–351. Springer International Publishing, Cham.

[88] Dror, M. and Trudeau, P. (1990). Split delivery routing. *Naval Research Logistics (NRL)*, 37(3):383–402.

[89] Dueck, G. (1993). New optimization heuristics: the great deluge algorithm and the record-to-record travel. *Journal of Computational physics*, 104(1):86–92.

[90] Duque, D., Lozano, L., and Medaglia, A. L. (2015). Solving the orienteering problem with time windows via the pulse framework. *Computers & Operations Research*, 54:168 – 176.

[91] Eiichi TANIGUCHI, R. G. T. and QURESHI, A. G. (2018). City logistics 1: New opportunities and challenges. In Taniguchi, E. and Thompson, R. G., editors, *City Logistics 1: New Opportunities and Challenges*, chapter 1- Recent Developments and Prospects for Modeling City Logistics, pages 1–28. John wiley & sons edition.

[92] Eilon, S., Watson-Gandy, C. D. T., and Christofides, N. (1971). *Distribution management*. Griffin London.

[93] Eksioglu, B., Vural, A. V., and Reisman, A. (2009). The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483.

[94] El-Hajj, R., Dang, D.-C., and Moukrim, A. (2016). Solving the team orienteering problem with cutting planes. *Computers & Operations Research*, 74:21 – 30.

[95] El-Hajj (El-Jajj), R. (2015). *Vehicle routing problems with profits, exact and heuristic approaches*. PhD thesis. Thèse de doctorat dirigée par Chebaro, Bilal et Moukrim, Aziz Technologies de l'Information et des Systèmes Compiègne 2015.

[96] Escobar, J. W., Linfati, R., Toth, P., and Baldoquin, M. G. (2014). A hybrid granular tabu search algorithm for the multi-depot vehicle routing problem. *Journal of Heuristics*, 20(5):483–509.

[97] Ferreira, J., Quintas, A., Oliveira, J. A., Pereira, G. A. B., and Dias, L. (2014). Solving the team orienteering problem: Developing a solution tool using a genetic algorithm approach. In Snášel, V., Krömer, P., Köppen, M., and Schaefer, G., editors, *Soft Computing in Industrial Applications*, pages 365–375, Cham. Springer International Publishing.

[98] Fischetti, M., Gonzalez, J. J. S., and Toth, P. (1998). Solving the orienteering problem through branch-and-cut. *INFORMS Journal on Computing*, 10(2):133–148.

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

[100] Foster, B. A. and Ryan, D. M. (1976). An integer programming approach to the vehicle scheduling problem. *Operational Research Quarterly*, pages 367–384.

[101] Gansterer, M., Küçüktepe, M., and Hartl, R. F. (2017). The multi-vehicle profitable pickup and delivery problem. *OR Spectrum*, 39(1):303–319.

[102] Garcia, A., Vansteenwegen, P., Arbelaitz, O., Souffriau, W., and Linaza, M. T. (2013). Integrating public transportation in personalised electronic tourist guides. *Computers & Operations Research*, 40(3):758 – 774. Transport Scheduling.

[103] Gaskell, T. J. (1967). Bases for vehicle fleet scheduling. *Or*, pages 281–295.

[104] Gavalas, D., Konstantopoulos, C., Mastakas, K., and Pantziou, G. (2014a). A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics*, 20(3):291–328.

[105] Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., and Vathis, N. (2014b). Efficient heuristics for the time dependent team orienteering problem with time windows. In Gupta, P. and Zaroliagis, C., editors, *Applied Algorithms*, pages 152–163, Cham. Springer International Publishing.

[106] Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., and Vathis, N. (2015). Approximation algorithms for the arc orienteering problem. *Information Processing Letters*, 115(2):313 – 315.

[107] Gavish, B. and Graves, S. C. (1978). The travelling salesman problem and related problems.

[108] Gedik, R., Kirac, E., Milburn, A. B., and Rainwater, C. (2017). A constraint programming approach for the team orienteering problem with time windows. *Computers & Industrial Engineering*, 107:178 – 195.

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

[110] Gendreau, M., Laporte, G., and Semet, F. (1998a). A branch-and-cut algorithm for the undirected selective traveling salesman problem. *Networks*, 32(4):263–273.

[111] Gendreau, M., Laporte, G., and Semet, F. (1998b). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, 106(2):539 – 545.

[112] Gendreau, M., Laporte, G., and Semet, F. (1998c). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, 106(2):539 – 545.

[113] Gendreau, M., Potvin, J.-Y., Bräumlaysy, O., Hasle, G., and Løkketangen, A. (2008a). *Metaheuristics for the Vehicle Routing Problem and Its Extensions: A Categorized Bibliography*, pages 143–169. Springer US, Boston, MA.

[114] Gendreau, M., Potvin, J.-Y., Bräumlaysy, O., Hasle, G., and Løkketangen, A. (2008b). *Metaheuristics for the Vehicle Routing Problem and Its Extensions: A Categorized Bibliography*, pages 143–169. Springer US, Boston, MA.

[115] Gendreau, M. and Tarantilis, C. D. (2010). Solving large-scale vehicle routing problems with time windows: The state-of-the-art. Technical report, Cirrelt, Montreal.

[116] Ghilas, V., Demir, E., and Van Woensel, T. (2013). Integrating passenger and freight transportation: Model formulation and insights. In *Proceedings of the 2013 Beta Working Papers (WP)*, volume 441. Technische Universiteit Eindhoven.

[117] Ghilas, V., Demir, E., and Van Woensel, T. (2016). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. *Computers & Operations Research*, 72:12–30.

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

[119] Giovanni, R. and Matteo, S. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51(3):155–170.

[120] Golden, B., Assad, A., and Dahl, R. (1984). Analysis of a large scale vehicle routing problem with an inventory component. *Large scale systems*, 7(2-3):181–190.

[121] Golden, B. L., Levy, L., and Vohra, R. (1987). The orienteering problem. *Naval Research Logistics (NRL)*, 34(3):307–318.

[122] 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.

[123] Groër, C., Golden, B., and Wasil, E. (2011). A parallel algorithm for the vehicle routing problem. *INFORMS Journal on Computing*, 23(2):315–330.

[124] Guibadj, R. N. and Moukrim, A. (2014). Memetic algorithm with an efficient split procedure for the team orienteering problem with time windows. In *International Conference on Artificial Evolution. EA 2013. Lecture Notes in Computer Science, vol 8752*, pages 183–194, Cham. Springer International Publishing.

[125] Gunawan, A., Lau, H. C., and Lu, K. (2015a). An iterated local search algorithm for solving the orienteering problem with time windows. In *Evolutionary Computation in Combinatorial Optimization*, pages 61–73, Cham. Springer International Publishing.

[126] Gunawan, A., Lau, H. C., and Lu, K. (2015b). Sails: hybrid algorithm for the team orienteering problem with time windows. In *Proceedings of the 10th international conference of the practice and theory of automated timetabling (patat 2014)*, pages 202–217, York, United Kingdom. MISTA.

[127] Gunawan, A., Lau, H. C., and Lu, K. (2015c). Well-tuned ils for extended team orienteering problem with time windows. In *LARC Technical Report Series*. Singapore Management University.

[128] Gunawan, A., Lau, H. C., and Vansteenwegen, P. (2016). Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315 – 332.

[129] Hadjiconstantinou, E., Christofides, N., and Mingozzi, A. (1995). A new exact algorithm for the vehicle routing problem based on q-paths and k-shortest paths relaxations. *Annals of Operations Research*, 61.

[130] Hashimoto, H. and Yagiura, M. (2008). A path relinking approach with an adaptive mechanism to control parameters for the vehicle routing problem with time windows. In van Hemert, J. and Cotta, C., editors, *Evolutionary Computation in Combinatorial Optimization*, pages 254–265, Berlin, Heidelberg. Springer Berlin Heidelberg.

[131] Hemmelmayr, V., Doerner, K. F., Hartl, R. F., and Rath, S. (2013). A heuristic solution method for node routing based solid waste collection problems. *Journal of Heuristics*, 19(2):129–156.

[132] 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.

[133] Ho, S. C. and Gendreau, M. (2006). Path relinking for the vehicle routing problem. *Journal of Heuristics*, 12(1):55–72.

[134] Holland, J. (1975). Adaptation in natural and artificial systems: an introductory analysis with application to biology. *Control and artificial intelligence*.

[135] Hu, Q. and Lim, A. (2014). An iterative three-component heuristic for the team orienteering problem with time windows. *European Journal of Operational Research*, 232(2):276–286.

[136] Irnich, S., Funke, B., and Grünert, T. (2006). Sequential search and its application to vehicle-routing problems. *Computers & Operations Research*, 33(8):2405 – 2429.

[137] Irnich, S., Schneider, M., and Vigo, D. (2014). *Chapter 9: Four Variants of the Vehicle Routing Problem*, chapter 9, pages 241–271. Number 18 in MOS-SIAM Series on Optimization. SIAM, Philadelphia.

[138] Jacobsen, S. K. and Madsen, O. B. G. (1980). A comparative study of heuristics for a two-level routing-location problem. *European Journal of Operational Research*, 5(6):378–387.

[139] Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511.

[140] Jepsen, M., Spoorendonk, S., and Ropke, S. (2013). A branch-and-cut algorithm for the symmetric two-echelon capacitated vehicle routing problem. *Transportation Science*, 47(1):23–37.

[141] Jin, M., Liu, K., and Bowden, R. O. (2007). A two-stage algorithm with valid inequalities for the split delivery vehicle routing problem. *International Journal of Production Economics*, 105(1):228 – 242.

[142] Jin, M., Liu, K., and Eksioglu, B. (2008). A column generation approach for the split delivery vehicle routing problem. *Operations Research Letters*, 36(2):265 – 270.

[143] Kallehauge, B. (2008). Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, 35(7):2307 – 2330. Part Special Issue: Includes selected papers presented at the ECCO'04 European Conference on combinatorial Optimization.

[144] Karakatič, S. and Podgorelec, V. (2015). A survey of genetic algorithms for solving multi depot vehicle routing problem. *Applied Soft Computing*, 27:519 – 532.

[145] Ke, L., Archetti, C., and Feng, Z. (2008). Ants can solve the team orienteering problem. *Computers & Industrial Engineering*, 54(3):648 – 665.

[146] Ke, L., Zhai, L., Li, J., and Chan, F. T. (2016). Pareto mimic algorithm: An approach to the team orienteering problem. *Omega*, 61:155 – 166.

[147] Keshtkaran, M., Ziarati, K., Bettinelli, A., and Vigo, D. (2016). Enhanced exact solution methods for the team orienteering problem. *International Journal of Production Research*, 54(2):591–601.

[148] Kikuta, J., Ito, T., Tomiyama, I., Yamamoto, S., and Yamada, T. (2012). New subway-integrated city logistics system. *Procedia - Social and Behavioral Sciences*, 39:476 – 489. Seventh International Conference on City Logistics which was held on June 7- 9,2011, Mallorca, Spain.

[149] Kim, B.-I., Kim, S., and Sahoo, S. (2006). Waste collection vehicle routing problem with time windows. *Computers & Operations Research*, 33(12):3624 – 3642. Part Special Issue: Recent Algorithmic Advances for Arc Routing Problems.

[150] Krajewska, M. A., Kopfer, H., Laporte, G., Ropke, S., and Zaccour, G. (2008). Horizontal cooperation among freight carriers: request allocation and profit sharing. *Journal of the Operational Research Society*, 59(11):1483–1491.

[151] Kytöjoki, J., Nuortio, T., Bräysy, O., and Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, 34(9):2743 – 2757.

[152] Labadie, N., Mansini, R., Melechovský, J., and Wolfler-Calvo, R. (2012). The team orienteering problem with time windows: An lp-based granular variable neighborhood search. *European Journal of Operational Research*, 220(1):15–27.

[153] Labadie, N., Melechovský, J., and Wolfler-Calvo, R. (2011). Hybridized evolutionary local search algorithm for the team orienteering problem with time windows. *Journal of Heuristics*, 17(6):729–753.

[154] Laporte, G. (2009). Fifty Years of Vehicle Routing. *Transportation Science*, 43:408–416.

[155] Laporte, G., Desrochers, M., and Nobert, Y. (1984). Two exact algorithms for the distance-constrained vehicle routing problem. *Networks*, 14(1):161–172.

[156] Laporte, G. and Martello, S. (1990). The selective travelling salesman problem. *Discrete Applied Mathematics*, 26(2):193 – 207.

[157] Laporte, G., Mercure, H., and Nobert, Y. (1986). An exact algorithm for the asymmetrical capacited vehicle routing problem. *Networks*, 16:33–46.

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

[159] Laporte, G., Nobert, Y., and Desrochers, M. (1985). Optimal routing under capacity and distance restrictions. *Operations research*, 33(5):1050–1073.

[160] Laporte, G., Nobert, Y., and Taillefer, S. (1988). Solving a Family of Multi-Depot Vehicle Routing and Location-Routing Problems. *Transportation Science*, 22(3):161–172.

[161] Leifer, A. C. and Rosenwein, M. B. (1994). Strong linear programming relaxations for the orienteering problem. *European Journal of Operational Research*, 73(3):517 – 523.

[162] Leonardi, J., Browne, M., and Allen, J. (2012). Before-after assessment of a logistics trial with clean urban freight vehicles: A case study in london. *Procedia - Social and Behavioral Sciences*, 39:146 – 157. Seventh International Conference on City Logistics which was held on June 7- 9,2011, Mallorca, Spain.

[163] Li, B., Krushinsky, D., Reijers, H. A., and Van Woensel, T. (2014). The Share-A-Ride Problem: People and parcels sharing taxis. *European Journal of Operational Research*, 238(1):31–40.

[164] Li, B., Krushinsky, D., Woensel, T. V., and Reijers, H. A. (2016a). An adaptive large neighborhood search heuristic for the share-a-ride problem. *Computers & Operations Research*, 66:170 – 180.

[165] Li, B., Krushinsky, D., Woensel, T. V., and Reijers, H. A. (2016b). The share-a-ride problem with stochastic travel times and stochastic delivery locations. *Transportation Research Part C: Emerging Technologies*, 67:95 – 108.

[166] Li, F., Golden, B., and Wasil, E. (2005). Very large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, 32(5):1165–1179.

[167] Li, Y., Chen, H., and Prins, C. (2016c). 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.

[168] Lin, S. (1965). Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269.

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

[170] Lin, S.-W. (2013). Solving the team orienteering problem using effective multi-start simulated annealing. *Applied Soft Computing*, 13(2):1064 – 1073.

[171] Lin, S.-W. and Yu, V. F. (2012). A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research*, 217(1):94–107.

[172] Liu, R., Jiang, Z., Fung, R. Y., Chen, F., and Liu, X. (2010). Two-phase heuristic algorithms for full truckloads multi-depot capacitated vehicle routing problem in carrier collaboration. *Computers & Operations Research*, 37(5):950 – 959. Disruption Management.

[173] Lozano, L., Duque, D., and Medaglia, A. L. (2016). An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science*, 50(1):348–357.

[174] Lu, Y., Benlic, U., and Wu, Q. (2018). A hybrid dynamic programming and memetic algorithm to the traveling salesman problem with hotel selection. *Computers & Operations Research*, 90:193 – 207.

[175] Lv, X., Wang, N., Zhen, Y., and Chen, H. (2016). Shipper collaboration with pickup and delivery requests in reverse logistics. *IFAC-PapersOnLine*, 49(12):1868 – 1873. 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016.

[176] Lyu, X., Wang, N., and andHaoxun Chen, Z. Y. (2017). Shipper collaboration in forward and reverse logistics. *Journal of Industrial & Management Optimization*, 13(1547-5816-2017-5-226):1.

[177] Madsen, O. B. G. (1983). Methods for solving combined two level location-routing problems of realistic dimensions. *European Journal of Operational Research*, 12(3):295–301.

[178] Masson, R., Trentini, A., Lehuédé, F., Malhéné, N., Péton, O., and Tlahig, H. (2017). Optimization of a city logistics transportation system with mixed passengers and goods. *EURO Journal on Transportation and Logistics*, 6(1):81–109.

[179] Mester, D. and Bräysy, O. (2005). Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers & Operations Research*, 32(6):1593 – 1614.

[180] Montemanni, R. and Gambardella, L. M. (2009). Ant colony system for team orienteering problems with time windows. *Foundations of Computing and Decision Sciences*, 34(4):287–306.

[181] Montemanni, R., Weyland, D., and Gambardella, L. M. (2011). An enhanced ant colony system for the team orienteering problem with time windows. In *2011 International Symposium on Computer Science and Society*, pages 381–384.

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

[183] Muthuswamy, S. and Lam, S. S. (2011). Discrete particle swarm optimization for the team orienteering problem. *Memetic Computing*, 3(4):287–303.

[184] Nagata, Y. and Bräysy, O. (2009). Edge assembly-based memetic algorithm for the capacitated vehicle routing problem. *Networks*, 54(4):205–215.

[185] Nagata, Y., Bräysy, O., and Dullaert, W. (2010). A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 37(4):724 – 737.

[186] Oliveira, F. B. D., Enayatifar, R., Sadaei, H. J., Guimarães, F. G., and Potvin, J.-Y. (2016). A cooperative coevolutionary algorithm for the multi-depot vehicle routing problem. *Expert Systems with Applications*, 43:117 – 130.

[187] Ombuki-Berman, B. and Hanshar, F. T. (2009). *Using Genetic Algorithms for Multi-depot Vehicle Routing*, pages 77–99. Springer Berlin Heidelberg, Berlin, Heidelberg.

[188] Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41(4):421–451.

[189] 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.

[190] Pecin, D., Contardo, C., Desaulniers, G., and Uchoa, E. (2017). New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 29(3):489–502.

[191] Perboli, G., Pezzella, F., and Tadei, R. (2008). Eve-opt: a hybrid algorithm for the capacitated vehicle routing problem. *Mathematical Methods of Operations Research*, 68(2):361.

[192] Perboli, G. and Tadei, R. (2010). New families of valid inequalities for the two-echelon vehicle routing problem. *Electronic notes in discrete mathematics*, 36:639–646.

[193] 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.

[194] Pisinger, D. and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403 – 2435.

[195] Poggi, M. and Uchoa, E. (2014). *Chapter 3: New Exact Algorithms for the Capacitated Vehicle Routing Problem*, chapter 3, pages 59–86. Number 18 in MOS-SIAM Series on Optimization. SIAM, Philadelphia.

[196] Poggi, M., Viana, H., and Uchoa, E. (2010). The Team Orienteering Problem: Formulations and Branch-Cut and Price. In Erlebach, T. and Lübbecke, M., editors, *10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10)*, volume 14 of *OpenAccess Series in Informatics (OASIcs)*, pages 142–155, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[197] Potvin, J.-Y. and Rousseau, J.-M. (1995). An exchange heuristic for routing problems with time windows. *Journal of the Operational Research Society*, 46(12):1433–1446.

[198] Prescott-Gagnon, E., Desaulniers, G., and Rousseau, L.-M. (2009). A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54(4):190–204.

[199] Prins, C. (2004a). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002.

[200] Prins, C. (2004b). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002.

[201] Prins, C. (2009a). A GRASP x evolutionary local search hybrid for the vehicle routing problem. In *Bio-inspired algorithms for the vehicle routing problem*, pages 35–53. Springer.

[202] Prins, C. (2009b). *A GRASP Evolutionary Local Search Hybrid for the Vehicle Routing Problem*, pages 35–53. Springer Berlin Heidelberg, Berlin, Heidelberg.

[203] Prins, C., Labadie, N., and Reghioui, M. (2009). Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47(2):507–535.

[204] Ramesh, R. and Brown, K. M. (1991). An efficient four-phase heuristic for the generalized orienteering problem. *Computers & Operations Research*, 18(2):151 – 165.

[205] Ramesh, R., Yoon, Y.-S., and Karwan, M. H. (1992). An optimal algorithm for the orienteering tour problem. *ORSA Journal on Computing*, 4(2):155–165.

[206] Reimann, M., Doerner, K., and Hartl, R. F. (2004). D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research*, 31(4):563 – 591.

[207] Renaud, J., Boctor, F. F., and Laporte, G. (1996). An improved petal heuristic for the vehicle routeing problem. *Journal of the Operational Research Society*, pages 329–336.

[208] Repoussis, P. P., Tarantilis, C. D., and Ioannou, G. (2009). Arc-guided evolutionary algorithm for the vehicle routing problem with time windows. *IEEE Transactions on Evolutionary Computation*, 13(3):624–647.

[209] Righini, G. and Salani, M. (2009). Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. *Computers & Operations Research*, 36(4):1191–1203.

[210] Rochat, Y. and Taillard, É. D. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167.

[211] Rossi, R. A., Gleich, D. F., Gebremedhin, A. H., Patwary, M. M. A., and Ali, M. (2013). A fast parallel maximum clique algorithm for large sparse graphs and temporal strong components. *CoRR, abs/1302.6256*.

[212] Ryan, D. M., Hjorring, C., and Glover, F. (1993). Extensions of the petal method for vehicle routeing. *Journal of the Operational Research Society*, pages 289–296.

[213] Sadykov, R. and Vanderbeck, F. (2013). Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing*, 25(2):244–255.

[214] Santos, F. A., da Cunha, A. S., and Mateus, G. R. (2013). Branch-and-price algorithms for the two-echelon capacitated vehicle routing problem. *Optimization Letters*, 7(7):1537–1547.

[215] Santos, F. A., Mateus, G. R., and da Cunha, A. S. (2014). A branch-and-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. *Transportation Science*, 49(2):355–368.

[216] Schilde, M., Doerner, K. F., Hartl, R. F., and Kiechle, G. (2009). Metaheuristics for the bi-objective orienteering problem. *Swarm Intelligence*, 3(3):179–201.

[217] Schmid, V. and Ehmke, J. F. (2017). An effective large neighborhood search for the team orienteering problem with time windows. In *Computational Logistics*, pages 3–18, Cham. Springer International Publishing.

[218] Semet, F., Toth, P., and Vigo, D. (2014). *Chapter 2: Classical Exact Algorithms for the Capacitated Vehicle Routing Problem*, chapter 2, pages 37–57. Number 18 in MOS-SIAM Series on Optimization. SIAM, Philadelphia.

[219] Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming—CP98*, pages 417–431. Springer.

[220] Silva, M. M., Subramanian, A., and Ochi, L. S. (2015). An iterated local search heuristic for the split delivery vehicle routing problem. *Computers & Operations Research*, 53:234 – 249.

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

[222] Song, L., Gu, H., and Huang, H. (2017). A lower bound for the adaptive two-echelon capacitated vehicle routing problem. *Journal of Combinatorial Optimization*, 33:1145–1167.

[223] Souffriau, W., Vansteenwegen, P., Berghe, G. V., and Oudheusden, D. V. (2010). A path relinking approach for the team orienteering problem. *Computers & Operations Research*, 37(11):1853 – 1859. Metaheuristics for Logistics and Vehicle Routing.

[224] Souffriau, W., Vansteenwegen, P., Vanden Berghe, G., and Van Oudheusden, D. (2013). The multiconstraint team orienteering problem with multiple time windows. *Transportation Science*, 47(1):53–63.

[225] Souffriau, W., Vansteenwegen, P., Vertommen, J., Berghe, G. V., and Oudheusden, D. V. (2008). A personalized tourist trip design algorithm for mobile tourist guides. *Applied Artificial Intelligence*, 22(10):964–985.

[226] Soysal, M., Bloemhof-Ruwaard, J. M., and Bektaş, T. (2015). The time-dependent two-echelon capacitated vehicle routing problem with environmental considerations. *International Journal of Production Economics*, 164:366–378.

[227] Sörensen, K. and Sevaux, M. (2006). Ma—pm: memetic algorithms with population management. *Computers & Operations Research*, 33(5):1214 – 1225.

[228] Subramanian, A., Uchoa, E., and Ochi, L. S. (2013). A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519 – 2531.

[229] Tae, H. and Kim, B.-I. (2015). A branch-and-price approach for the team orienteering problem with time windows. *International Journal of Industrial Engineering*, 22(2):243 – 251.

[230] Taillard, É. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, 23(8):661–673.

[231] Takeo, Y., Seiji, K., and Kohtaro, W. (2002). Heuristic and exact algorithms for the disjunctively constrained knapsack problem. *Information Processing Society of Japan Journal*, 43(9):2864–2870.

[232] Tang, H. and Miller-Hooks, E. (2005a). A tabu search heuristic for the team orienteering problem. *Computers & Operations Research*, 32(6):1379 – 1407.

[233] Tang, H. and Miller-Hooks, E. (2005b). A tabu search heuristic for the team orienteering problem. *Computers & Operations Research*, 32:1379–1407.

[234] Taniguchi, E. and Thompson, R. G. (2002). Modeling city logistics. *Transportation Research Record*, 1790(1):45–51.

[235] Taniguchi, E. and Thompson, R. G., editors (2014). *City Logistics - Mapping The Future*. Crc press edition.

[236] Taniguchi, E., Thompson, R. G., and Yamada, T. (2016). New opportunities and challenges for city logistics. *Transportation Research Procedia*, 12:5 – 13. Tenth International Conference on City Logistics 17-19 June 2015, Tenerife, Spain.

[237] Taniguchi, E., Thompson, R. G., Yamada, T., and van Duin, J. (2001). *City logistics –Network modelling and intelligent transport systems*. Pergamon, Oxford, elsevier edition.

[238] Tarantilis, C. (2005). Solving the vehicle routing problem with adaptive memory programming methodology. *Computers & Operations Research*, 32(9):2309 – 2327.

[239] Tarantilis, C., Stavropoulou, F., and Repoussis, P. (2013a). The capacitated team orienteering problem: A bi-level filter-and-fan method. *European Journal of Operational Research*, 224(1):65 – 78.

[240] Tarantilis, C. D., Anagnostopoulou, A. K., and Repoussis, P. P. (2013b). Adaptive path relinking for vehicle routing and scheduling problems with product returns. *Transportation Science*, 47(3):356–379.

[241] Tarantilis, C. D., Zachariadis, E. E., and Kiranoudis, C. T. (2008). A hybrid guided local search for the vehicle-routing problem with intermediate replenishment facilities. *INFORMS Journal on Computing*, 20(1):154–168.

[242] Tarjan, R. E. (1975). Graph theory and gaussian elimination. Technical report, Stanford University.

[243] Thomadsen, T. and Stidsen, T. K. (2003). The quadratic selective travelling salesman problem. Technical report.

[244] Thompson, P. M. and Psaraftis, H. N. (1993). Cyclic Transfer Algorithms for Multivehicle Routing and Scheduling Problems. *Operations research*, 41(5):935–946.

[245] Thompson, R. G. and Taniguchi, E. (2014). City Logistics Mapping The Future. In Taniguchi, E. and Thompson, R. G., editors, *City Logistics - Mapping The Future*, chapter 13- Future Directions, pages 201–210. Crc press edition.

[246] Toth, P. and Vigo, D. (2002). The vehicle routing problem.

[247] Toth, P. and Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4):333–346.

[248] Toth, P. and Vigo, D., editors (2014). *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. Number 18 in MOS-SIAM Series on Optimization. SIAM.

[249] Trentini, A. and Mahléné, N. (2010). Toward a Shared Urban Transport System Ensuring Passengers & Goods Cohabitation. *TeMa Journal of Land Use, Mobility and Environment*, 3(2):37–44.

[250] Trentini, A., Masson, R., Lehuédé, F., Malhéné, N., Péton, O., and Tlahig, H. (2012). A shared " passengers & goods " city logistics system. page 10p.

[251] Tricoire, F., Romauch, M., Doerner, K. F., and Hartl, R. F. (2010). Heuristics for the multi-period orienteering problem with multiple time windows. *Computers & Operations Research*, 37:351–367.

[252] Tsiligirides, T. (1984). Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809.

[253] van Rooijen, T. and Quak, H. (2014). City logistics in the european civitas initiative. *Procedia - Social and Behavioral Sciences*, 125:312 – 325. Eighth International Conference on City Logistics 17-19 June 2013, Bali, Indonesia.

[254] Vansteenwegen, P., Souffriau, W., Berghe, G. V., and Oudheusden, D. V. (2009a). A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research*, 196(1):118 – 127.

[255] Vansteenwegen, P., Souffriau, W., Berghe, G. V., and Oudheusden, D. V. (2009b). Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research*, 36(12):3281–3290.

[256] Vansteenwegen, P., Souffriau, W., Berghe, G. V., and Oudheusden, D. V. (2009c). *Metaheuristics for Tourist Trip Planning*, pages 15–31. Springer Berlin Heidelberg, Berlin, Heidelberg.

[257] Vansteenwegen, P., Souffriau, W., and Oudheusden, D. V. (2011). The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10.

[258] Vansteenwegen, P., Souffriau, W., and Sörensen, K. (2012). The travelling salesperson problem with hotel selection. *Journal of the Operational Research Society*, 63(2):207–217.

[259] Vansteenwegen, P. and Van Oudheusden, D. (2007). The mobile tourist guide: An or opportunity. *OR Insight*, 20(3):21–27.

[260] Verbeeck, C., Vansteenwegen, P., and Aghezzaf, E.-H. (2014). An extension of the arc orienteering problem and its application to cycle trip planning. *Transportation Research Part E: Logistics and Transportation Review*, 68:64 – 78.

[261] Verdonck, L., Caris, A., Ramaekers, K., and Janssens, G. K. (2013). Collaborative logistics from the perspective of road transportation companies. *Transport Reviews*, 33(6):700–719.

[262] Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., and Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations research*, 60(3):611–624.

[263] Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013a). Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. *European Journal of Operational Research*, 231(1):1–21.

[264] Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013b). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231:1–21.

[265] Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013c). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475 – 489.

[266] Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673.

[267] 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*, 57(October):262–276.

[268] Wren, A. and Holliday, A. (1972). Computer scheduling of vehicles from one or more depots to a number of delivery points. *Journal of the Operational Research Society*, 23(3):333–344.

[269] Yamada, T. (2014). City Logistics Mapping The Future. In Taniguchi, E. and Thompson, R. G., editors, *City Logistics - Mapping The Future*, chapter 11- Cooperative Freight Transport Systems, pages 167–176. Crc press edition.

[270] Yellow, P. C. (1970). A computational modification to the savings method of vehicle scheduling. *Operational Research Quarterly*, pages 281–283.

[271] Yu, B., Yang, Z.-Z., and Yao, B. (2009). An improved ant colony optimization for vehicle routing problem. *European Journal of Operational Research*, 196(1):171–176.

[272] Zeng, Z. Y., Xu, W. S., Xu, Z. Y., and Shao, W. H. (2014). A Hybrid GRASP+VND heuristic for the two-echelon vehicle routing problem arising in city logistics. *Mathematical Problems in Engineering*, 2014.

[273] Zhou, L., Baldacci, R., Vigo, D., and Wang, X. (2018). A Multi-Depot Two-Echelon Vehicle Routing Problem with Delivery Options Arising in the Last Mile Distribution. *European Journal of Operational Research*, 265(2):765–778.

Detailed results of the MS-ILS on TOPTW benchmarks

In this appendix, we present the detailed results obtained by our Multi-Start Iterated Local Search on all the TOPTW instances from the standard and the "OPT" benchmarks.

Tables A.2 to A.11 and Tables A.13 to A.22 show the detailed results obtained by MS-ILS using two different parameter settings : one that offers good compromise between speed and solution quality, and one which favors solution quality. The performance of MS-ILS is measured in terms of relative percentage error (rpe) with respect to the best-known solution (BKS), and in terms of average relative percentage error ($arpe$) with respect to the BKS. These two metrics are computed as: $rpe = \frac{(BKS - Z_{max})}{BKS} * 100\%$ and $arpe = \frac{(BKS - Z_{avg})}{BKS} * 100\%$, where Z_{max} and Z_{avg} denote the best profit and the average profit obtained over five runs, respectively.

Each of the aforementioned tables is composed of two identically structured parts. Each part contains seven columns. The first column displays the name of the instance. Column (BKS) shows the current best-known solution to the instance, including the one found by our method. The columns remaining under the heading MS-ILS indicate the maximum score (Z_{max}) obtained by our algorithm, the relative error (RPE), the average score (Z_{avg}), the average error ($ARPE$), and the average computational time in seconds (cpu_{avg}), respectively. New best-known solutions are displayed in bold in column (Z_{max}).

1.1 Results of the MS-ILS using a fast setting

In this Section, we present the results obtained by the MS-ILS while using a combination of parameter values that offer a good compromise between computation times and solution quality (fast setting). The parameter values are detailed in Table A.1. Tables A.2 to A.9 report the results obtained by MS-ILS on instances from the standard benchmark, while

Tables A.10 to A.11 show those obtained on the "OPT" benchmark. Note that in the case of the "OPT" benchmark, the solution value reported in column (BKS) is known to be optimal.

Table A.1 – Fast setting parameter values.

Parameter	Description	Value
α	the control parameter of the BIA	$[1.8, 2.8]$
D_{max}	max. nb. of customers removed by the Random remove-and-repair operator	$0.25 * n_{routed}$
$\mathcal{M}_{size}$	size of the adaptive memory	80
$iter_{init}$	nb. of iterations of the initialization heuristic	1000
$iter_{max}$	nb. of iterations of the main algorithm	$2 * n/m$
$iter_{ils}$	nb. of iterations of the ILS	4

Table A.2 – Results for Solomon's instances with m = 1 using the fast setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
c101	320	320	0.00	320	0.00	0.4	c201	870	870	0.00	870	0.00	3.3
c102	360	360	0.00	360	0.00	1.9	c202	930	930	0.00	930	0.00	6.1
c103	400	400	0.00	400	0.00	3.7	c203	960	960	0.00	960	0.00	11.6
c104	420	420	0.00	420	0.00	5.7	c204	980	980	0.00	974	0.61	29.4
c105	340	340	0.00	340	0.00	1.0	c205	910	910	0.00	910	0.00	6.3
c106	340	340	0.00	340	0.00	0.6	c206	930	930	0.00	930	0.00	4.9
c107	370	370	0.00	370	0.00	0.9	c207	930	930	0.00	930	0.00	7.9
c108	370	370	0.00	370	0.00	1.1	c208	950	950	0.00	950	0.00	9.9
c109	380	380	0.00	380	0.00	2.6							
r101	198	198	0.00	198	0.00	0.4	r201	797	797	0.00	797	0.00	5.5
r102	286	286	0.00	286	0.00	2.0	r202	930	930	0.00	926.8	0.34	16.9
r103	293	293	0.00	293	0.00	2.7	r203	1028	1028	0.00	1026.8	0.12	23.5
r104	303	303	0.00	303	0.00	3.4	r204	1093	1093	0.00	1093	0.00	36.0
r105	247	247	0.00	247	0.00	0.9	r205	953	953	0.00	953	0.00	10.0
r106	293	293	0.00	293	0.00	2.4	r206	1032	1032	0.00	1031.6	0.04	20.4
r107	299	299	0.00	299	0.00	2.5	**r207**	**1078**	**1078**	0.00	1077.6	0.04	24.6
r108	308	308	0.00	308	0.00	3.1	**r208**	**1118**	**1118**	0.00	1116.6	0.13	46.6
r109	277	277	0.00	277	0.00	1.4	r209	962	959	0.31	958.6	0.35	24.4
r110	284	284	0.00	283.4	0.21	1.3	**r210**	**1002**	**1002**	0.00	1001.6	0.04	20.4
r111	297	297	0.00	297	0.00	2.4	r211	1051	1051	0.00	1051	0.00	23.8
r112	298	298	0.00	298	0.00	2.3							
rc101	219	219	0.00	219	0.00	0.6	rc201	795	795	0.00	795	0.00	3.9
rc102	266	266	0.00	266	0.00	1.2	rc202	938	938	0.00	938	0.00	10.0
rc103	266	266	0.00	266	0.00	1.1	rc203	1003	1003	0.00	1002.4	0.06	18.0
rc104	301	301	0.00	301	0.00	1.3	**rc204**	**1143**	**1143**	0.00	1139.4	0.31	28.6
rc105	244	244	0.00	244	0.00	0.8	rc205	859	859	0.00	859	0.00	7.4
rc106	252	252	0.00	251.6	0.16	0.8	rc206	899	899	0.00	896.6	0.27	8.8
rc107	277	277	0.00	277	0.00	1.5	rc207	983	983	0.00	983	0.00	12.1
rc108	298	298	0.00	298	0.00	1.3	rc208	1058	1057	0.00	1055	0.28	18.5
Avg.	303.7	303.7	0.00	303.6	0.01	1.77		969.7	969.6	0.01	968.7	0.10	16.2

Table A.3 – Results for Solomon's instances with m = 2 using the fast setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
c101	590	590	0.00	590	0.00	1.6	c201	1460	1460	0.00	1454	0.41	7.6
c102	660	660	0.00	658	0.30	3.2	c202	1470	1470	0.00	1470	0.00	12.4
c103	720	720	0.00	720	0.00	5.5	c203	1480	1480	0.00	1478	0.14	24.2
c104	760	760	0.00	760	0.00	9.3	c204	1490	1490	0.00	1490	0.00	45.9
c105	640	640	0.00	640	0.00	1.4	c205	1470	1470	0.00	1470	0.00	13.5
c106	620	620	0.00	620	0.00	1.9	c206	1480	1480	0.00	1480	0.00	8.7
c107	670	670	0.00	670	0.00	1.6	c207	1490	1490	0.00	1484	0.40	10.7
c108	680	680	0.00	680	0.00	2.6	c208	1490	1490	0.00	1488	0.13	11.8
c109	720	720	0.00	720	0.00	5.4							
r101	349	349	0.00	349	0.00	1.2	r201	1260	1255	0.40	1252.4	0.60	13.7
r102	508	508	0.00	508	0.00	1.9	r202	1353	1352	0.07	1349.8	0.24	42.9
r103	522	522	0.00	521.6	0.08	2.6	**r203**	**1431**	**1431**	0.00	1424.6	0.45	61.3
r104	552	552	0.00	550.4	0.29	4.5	r204	1458	1458	0.00	1458	0.00	3.6
r105	453	453	0.00	453	0.00	1.6	**r205**	**1402**	**1402**	0.00	1395.4	0.47	32.0
r106	529	529	0.00	529	0.00	2.3	r206	1452	1450	0.14	1449.2	0.19	60.4
r107	538	538	0.00	538	0.00	3.2	r207	1458	1458	0.00	1458	0.00	4.9
r108	560	560	0.00	560	0.00	3.7	r208	1458	1458	0.00	1458	0.00	0.2
r109	506	506	0.00	506	0.00	2.5	**r209**	**1423**	**1423**	0.00	1418.2	0.34	51.8
r110	525	525	0.00	524.6	0.08	2.4	r210	1438	1434	0.28	1430.4	0.53	54.5
r111	544	544	0.00	544	0.00	2.7	r211	1458	1458	0.00	1458	0.00	14.3
r112	544	544	0.00	544	0.00	3.5							
rc101	427	427	0.00	427	0.00	1.8	rc201	1386	1383	0.22	1378.8	0.52	7.8
rc102	505	505	0.00	504.2	0.16	2.5	**rc202**	**1523**	**1523**	0.00	1520.8	0.14	16.4
rc103	524	524	0.00	524	0.00	2.5	rc203	1639	1637	0.12	1634.8	0.26	31.1
rc104	575	575	0.00	574.6	0.07	3.3	rc204	1718	1716	0.12	1712.2	0.34	58.0
rc105	480	480	0.00	480	0.00	3.1	rc205	1462	1462	0.00	1460	0.14	13.1
rc106	483	483	0.00	482.2	0.17	1.8	rc206	1552	1550	0.13	1543.2	0.57	13.7
rc107	534	534	0.00	534	0.00	4.3	**rc207**	**1609**	**1609**	0.00	1602	0.44	27.3
rc108	556	556	0.00	556	0.00	4.0	rc208	1705	1695	0.59	1693.6	0.67	39.8
Avg.	561.2	561.2	0.00	561.0	0.04	3.03		1482.0	1480.9	0.08	1478.2	0.26	25.2

Table A.4 – Results for Solomon's instances with m = 3 using the fast setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
c101	810	810	0.00	810	0.00	1.6	c201	1810	1810	0.00	1810	0.00	1.0
c102	920	920	0.00	920	0.00	5.0	c202	1810	1810	0.00	1810	0.00	1.4
c103	990	990	0.00	984	0.61	8.2	c203	1810	1810	0.00	1810	0.00	1.4
c104	1030	1030	0.00	1030	0.00	10.4	c204	1810	1810	0.00	1810	0.00	1.8
c105	870	870	0.00	868	0.23	2.2	c205	1810	1810	0.00	1810	0.00	0.2
c106	870	870	0.00	870	0.00	3.0	c206	1810	1810	0.00	1810	0.00	0.1
c107	910	910	0.00	910	0.00	3.6	c207	1810	1810	0.00	1810	0.00	0.2
c108	920	920	0.00	920	0.00	4.7	c208	1810	1810	0.00	1810	0.00	0.1
c109	970	970	0.00	970	0.00	7.4							
r101	484	484	0.00	484	0.00	1.5	r201	1450	1445	0.34	1440.4	0.66	33.8
r102	694	694	0.00	692.2	0.26	3.1	r202	1458	1458	0.00	1458	0.00	0.7
r103	747	747	0.00	747	0.00	4.2	r203	1458	1458	0.00	1458	0.00	0.1
r104	778	778	0.00	778	0.00	5.5	r204	1458	1458	0.00	1458	0.00	0.1
r105	620	620	0.00	619.4	0.10	2.2	r205	1458	1458	0.00	1458	0.00	0.1
r106	729	729	0.00	727.2	0.25	2.9	r206	1458	1458	0.00	1458	0.00	0.1
r107	760	760	0.00	760	0.00	4.4	r207	1458	1458	0.00	1458	0.00	0.1
r108	797	797	0.00	797	0.00	3.7	r208	1458	1458	0.00	1458	0.00	0.1
r109	710	710	0.00	710	0.00	2.8	r209	1458	1458	0.00	1458	0.00	0.1
r110	737	737	0.00	736.4	0.08	2.7	r210	1458	1458	0.00	1458	0.00	0.1
r111	774	774	0.00	773.8	0.03	3.7	r211	1458	1458	0.00	1458	0.00	0.1
r112	776	776	0.00	776	0.00	5.6							
rc101	621	621	0.00	621	0.00	1.4	rc201	1698	1698	0.00	1693.2	0.28	19.3
rc102	714	714	0.00	711.6	0.34	2.5	rc202	1724	1724	0.00	1724	0.00	16.9
rc103	764	764	0.00	753.4	1.39	2.9	rc203	1724	1724	0.00	1724	0.00	0.1
rc104	835	835	0.00	834.8	0.02	4.3	rc204	1724	1724	0.00	1724	0.00	0.1
rc105	682	682	0.00	682	0.00	2.3	rc205	1719	1709	0.58	1709	0.58	27.2
rc106	706	706	0.00	705.2	0.11	2.3	rc206	1724	1724	0.00	1724	0.00	0.3
rc107	773	773	0.00	772	0.13	3.0	rc207	1724	1724	0.00	1724	0.00	0.2
rc108	795	795	0.00	795	0.00	4.3	rc208	1724	1724	0.00	1724	0.00	0.1
Avg.	785.7	785.7	0.00	784.8	0.12	3.84		1639.7	1639.1	0.03	1638.8	0.06	3.9

Table A.5 – Results for Solomon's instances with m = 4 using the fast setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
c101	1020	1020	0.00	1020	0.00	3.2	c201	1810	1810	0.00	1810	0.00	0.0
c102	1150	1150	0.00	1150	0.00	8.4	c202	1810	1810	0.00	1810	0.00	0.1
c103	1210	1210	0.00	1206	0.33	11.7	c203	1810	1810	0.00	1810	0.00	0.1
c104	1260	1260	0.00	1254	0.48	16.4	c204	1810	1810	0.00	1810	0.00	0.1
c105	1070	1060	0.93	1060	0.93	3.2	c205	1810	1810	0.00	1810	0.00	0.1
c106	1080	1080	0.00	1074	0.56	5.0	c206	1810	1810	0.00	1810	0.00	0.1
c107	1120	1120	0.00	1120	0.00	4.7	c207	1810	1810	0.00	1810	0.00	0.1
c108	**1140**	**1140**	0.00	1134	0.53	7.2	c208	1810	1810	0.00	1810	0.00	0.1
c109	1190	1190	0.00	1190	0.00	9.0							
r101	611	611	0.00	610.8	0.03	1.9	r201	1458	1458	0.00	1458	0.00	0.1
r102	843	839	0.47	837.4	0.66	3.2	r202	1458	1458	0.00	1458	0.00	0.1
r103	928	928	0.00	926.4	0.17	4.5	r203	1458	1458	0.00	1458	0.00	0.1
r104	975	974	0.10	974	0.10	7.4	r204	1458	1458	0.00	1458	0.00	0.1
r105	778	778	0.00	774.8	0.41	2.9	r205	1458	1458	0.00	1458	0.00	0.1
r106	906	906	0.00	906	0.00	4.3	r206	1458	1458	0.00	1458	0.00	0.1
r107	950	950	0.00	950	0.00	4.9	r207	1458	1458	0.00	1458	0.00	0.1
r108	995	995	0.00	992.6	0.24	5.8	r208	1458	1458	0.00	1458	0.00	0.1
r109	885	885	0.00	885	0.00	3.3	r209	1458	1458	0.00	1458	0.00	0.1
r110	915	915	0.00	915	0.00	4.1	r210	1458	1458	0.00	1458	0.00	0.1
r111	952	951	0.11	949.4	0.27	5.0	r211	1458	1458	0.00	1458	0.00	0.1
r112	**974**	**974**	0.00	972.2	0.18	5.9							
rc101	811	811	0.00	811	0.00	1.8	rc201	1724	1724	0.00	1724	0.00	0.1
rc102	909	902	0.77	901.8	0.79	3.0	rc202	1724	1724	0.00	1724	0.00	0.1
rc103	975	974	0.10	968.4	0.68	4.2	rc203	1724	1724	0.00	1724	0.00	0.1
rc104	1065	1065	0.00	1064.6	0.04	4.4	rc204	1724	1724	0.00	1724	0.00	0.1
rc105	875	875	0.00	872.2	0.32	2.8	rc205	1724	1724	0.00	1724	0.00	0.1
rc106	909	909	0.00	904.2	0.53	3.5	rc206	1724	1724	0.00	1724	0.00	0.1
rc107	987	987	0.00	984	0.30	3.7	rc207	1724	1724	0.00	1724	0.00	0.1
rc108	1025	1025	0.00	1025	0.00	4.4	rc208	1724	1724	0.00	1724	0.00	0.1
Avg.	983.0	982.2	0.09	980.4	0.26	5.16		1641.1	1641.1	0.00	1641.1	0.00	0.1

Table A.6 – Results for Cordeau's instances with m = 1 using the fast setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
pr01	308	308	0.00	308	0.00	0.9	pr11	353	353	0.00	351.8	0.34	1.4
pr02	404	404	0.00	404	0.00	3.6	pr12	442	441	0.23	440.8	0.27	6.1
pr03	394	394	0.00	394	0.00	5.9	pr13	467	467	0.00	464.4	0.56	8.3
pr04	489	489	0.00	489	0.00	12.7	pr14	567	555	2.12	553	2.47	22.7
pr05	595	595	0.00	594.4	0.10	24.3	pr15	708	708	0.00	708	0.00	39.5
pr06	591	590	0.17	587.6	0.58	40.6	pr16	674	650	3.56	648.2	3.83	53.0
pr07	298	298	0.00	298	0.00	1.3	pr17	362	362	0.00	362	0.00	1.8
pr08	463	463	0.00	463	0.00	5.3	pr18	539	539	0.00	539	0.00	8.7
pr09	493	493	0.00	493	0.00	10.8	pr19	562	560	0.36	553.4	1.53	28.8
pr10	594	594	0.00	587.2	1.14	24.6	pr20	667	652	2.25	650.4	2.49	44.3
Avg.	462.9	462.8	0.02	461.82	0.18	13.0		534.1	528.7	0.85	527.1	1.15	21.5

Table A.7 – Results for Cordeau's instances with m = 2 using the fast setting.

		MS-ILS						MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
pr01	502	502	0.00	502	0.00	1.2	pr11	566	564	0.35	564	0.35	1.9
pr02	715	714	0.14	711.8	0.45	5.0	pr12	774	765	1.16	765	1.16	8.6
pr03	742	742	0.00	738.4	0.49	7.6	pr13	845	840	0.59	836.4	1.02	13.8
pr04	**928**	**928**	0.00	926.8	0.13	22.5	pr14	1017	993	2.36	991	2.56	29.0
pr05	1103	1094	0.82	1093.2	0.89	26.0	**pr15**	**1238**	**1238**	0.00	1228.8	0.74	72.2
pr06	1076	1076	0.00	1060.8	1.41	39.9	pr16	1231	1198	2.68	1182	3.98	68.4
pr07	566	566	0.00	566	0.00	2.7	pr17	652	646	0.92	645.4	1.01	3.0
pr08	834	834	0.00	833	0.12	10.3	**pr18**	**955**	**955**	0.00	947.4	0.80	18.9
pr09	909	909	0.00	907.2	0.20	22.0	pr19	1041	1038	0.29	1030.8	0.98	35.6
pr10	**1145**	**1145**	0.00	1137.8	0.63	43.4	**pr20**	**1251**	**1251**	0.00	1246.8	0.34	69.8
Avg.	852	851	0.10	847.7	0.43	18.1		957	948.8	0.84	943.76	1.29	32.1

Table A.8 – Results for Cordeau's instances with m = 3 using the fast setting.

		MS-ILS						MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
pr01	622	622	0.00	619.6	0.39	1.7	pr11	654	654	0.00	654	0.00	3.0
pr02	945	943	0.21	941.6	0.36	8.3	pr12	1002	999	0.30	998	0.40	13.2
pr03	1014	1013	0.10	1009.4	0.45	14.1	pr13	1159	1157	0.17	1147	1.04	19.6
pr04	1296	1290	0.46	1279.8	1.25	21.7	pr14	1375	1367	0.58	1361.8	0.96	45.7
pr05	1500	1487	0.87	1485.6	0.96	38.5	pr15	1694	1684	0.59	1682.2	0.70	58.4
pr06	**1515**	**1515**	0.00	1504.6	0.69	51.8	pr16	1668	1637	1.86	1628.8	2.35	92.2
pr07	744	744	0.00	742.8	0.16	3.9	pr17	841	841	0.00	838.2	0.33	5.1
pr08	1141	1137	0.35	1135	0.53	13.0	pr18	1289	1289	0.00	1276.2	0.99	16.5
pr09	**1277**	**1277**	0.00	1265.6	0.89	26.8	pr19	1428	1423	0.35	1417.6	0.73	48.3
pr10	1582	1577	0.32	1571.6	0.66	76.9	pr20	1722	1706	0.93	1688.8	1.93	131.1
Avg.	1163.6	1160.5	0.23	1155.56	0.63	25.7		1283.2	1275.7	0.48	1269.26	0.94	43.3

Table A.9 – Results for Cordeau's instances with m = 4 using the fast setting.

		MS-ILS						MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
pr01	657	657	0.00	657	0.00	0.1	pr11	657	657	0.00	657	0.00	0.0
pr02	1083	1079	0.37	1075.6	0.68	12.8	pr12	1133	1131	0.18	1127.8	0.46	22.1
pr03	**1247**	**1247**	0.00	1237.4	0.77	14.3	pr13	1392	1380	0.86	1377.4	1.05	28.3
pr04	1595	1593	0.13	1587.4	0.48	38.7	pr14	1688	1684	0.24	1676	0.71	59.5
pr05	1858	1841	0.91	1835.4	1.22	61.5	pr15	2085	2071	0.67	2056.2	1.38	119.5
pr06	1894	1885	0.48	1875.8	0.96	67.9	pr16	2065	2052	0.63	2042.4	1.09	124.8
pr07	876	876	0.00	872	0.46	4.7	pr17	936	932	0.43	923.6	1.32	7.1
pr08	1390	1380	0.72	1373.6	1.18	17.9	pr18	1554	1539	0.97	1530.4	1.52	35.6
pr09	1622	1613	0.55	1607.4	0.90	45.6	pr19	1780	1772	0.45	1756	1.35	63.8
pr10	1965	1958	0.36	1943.4	1.10	80.2	**pr20**	**2115**	**2115**	0.00	2084.4	1.45	129.9
Avg.	1418.7	1412.9	0.35	1406.5	0.77	34.4		1540.5	1533.3	0.44	1523.12	1.03	59.1

Table A.10 – Results for Solomon's instances of "OPT" data set using the fast setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
c101	1810	1810	0.00	1810	0.00	2.6	c201	1810	1810	0.00	1810	0.00	0.0
c102	1810	1810	0.00	1810	0.00	0.1	c202	1810	1810	0.00	1810	0.00	0.1
c103	1810	1810	0.00	1810	0.00	0.1	c203	1810	1810	0.00	1810	0.00	0.1
c104	1810	1810	0.00	1810	0.00	0.1	c204	1810	1810	0.00	1810	0.00	0.1
c105	1810	1810	0.00	1810	0.00	0.2	c205	1810	1810	0.00	1810	0.00	0.1
c106	1810	1810	0.00	1810	0.00	1.1	c206	1810	1810	0.00	1810	0.00	0.1
c107	1810	1810	0.00	1810	0.00	0.0	c207	1810	1810	0.00	1810	0.00	0.1
c108	1810	1810	0.00	1810	0.00	0.1	c208	1810	1810	0.00	1810	0.00	0.1
c109	1810	1810	0.00	1810	0.00	0.1							
r101	1458	1458	0.00	1457.4	0.04	4.1	r201	1458	1458	0.00	1458	0.00	0.1
r102	1458	1458	0.00	1458	0.00	10.7	r202	1458	1458	0.00	1458	0.00	0.5
r103	1458	1458	0.00	1455.6	0.16	58.8	r203	1458	1458	0.00	1458	0.00	0.1
r104	1458	1442	1.10	1438.8	1.32	12.1	r204	1458	1458	0.00	1458	0.00	3.7
r105	1458	1458	0.00	1453.4	0.32	6.8	r205	1458	1458	0.00	1458	0.00	0.1
r106	1458	1458	0.00	1457.6	0.03	13.6	r206	1458	1458	0.00	1458	0.00	0.1
r107	1458	1453	0.34	1450.4	0.52	14.5	r207	1458	1458	0.00	1458	0.00	2.5
r108	1458	1455	0.21	1454.6	0.23	15.7	r208	1458	1458	0.00	1458	0.00	0.3
r109	1458	1450	0.55	1447.2	0.74	8.6	r209	1458	1458	0.00	1458	0.00	0.1
r110	1458	1447	0.75	1444.2	0.95	11.3	r210	1458	1458	0.00	1458	0.00	0.1
r111	1458	1450	0.55	1448.2	0.67	10.4	r211	1458	1458	0.00	1458	0.00	24.0
r112	1458	1449	0.62	1448.8	0.63	9.7							
rc101	1724	1719	0.29	1715	0.52	10.3	rc201	1724	1724	0.00	1724	0.00	0.1
rc102	1724	1721	0.17	1721	0.17	19.7	rc202	1724	1724	0.00	1724	0.00	9.9
rc103	1724	1724	0.00	1721.8	0.13	15.9	rc203	1724	1724	0.00	1724	0.00	0.1
rc104	1724	1724	0.00	1724	0.00	0.5	rc204	1724	1724	0.00	1724	0.00	0.1
rc105	1724	1711	0.75	1706.6	1.01	7.0	rc205	1724	1724	0.00	1724	0.00	0.1
rc106	1724	1719	0.29	1705.8	1.06	8.6	rc206	1724	1724	0.00	1724	0.00	0.2
rc107	1724	1724	0.00	1723.6	0.02	4.4	rc207	1724	1724	0.00	1724	0.00	0.2
rc108	1724	1724	0.00	1722.8	0.07	9.1	rc208	1724	1724	0.00	1724	0.00	0.1
Avg.	1640.6	1637.7	0.19	1636.0	0.30	8.82		1641.1	1641.1	0.00	1641.1	0.00	1.6

Table A.11 – Results for Cordeau's instances of "OPT" data set using the fast setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
pr01	657	619	5.78	619	5.78	1.5	pr06	3671	3671	0.00	3671	0.00	113.9
pr02	1220	1207	1.07	1202.4	1.44	16.9	pr07	948	942	0.63	939.2	0.93	4.5
pr03	1788	1770	1.01	1767.8	1.13	24.9	pr08	2006	2006	0.00	2005.8	0.01	11.7
pr04	2477	2474	0.12	2471.2	0.23	69.3	pr09	2736	2736	0.00	2736	0.00	0.2
pr05	3351	3351	0.00	3351	0.00	20.7	pr10	3850	3850	0.00	3850	0.00	0.3
Avg.	1898.6	1884.2	1.60	1882.3	1.72	26.7		2642.2	2641.0	0.13	2640.4	0.19	26.1

1.2 Results of the MS-ILS using a slow setting

In this Section, we present the results obtained by the MS-ILS while using a combination of parameter values to favor solution quality (slow setting) instead of computation time. The parameter values are detailed in Table A.12. Tables A.13 to A.20 report the results obtained by MS-ILS on instances from the standard benchmark, while Tables A.21 to A.22 show those obtained on the "OPT" benchmark. Note that in the case of the "OPT" benchmark, the solution value reported in column (BKS) is known to be optimal.

Table A.12 – Slow setting parameter values.

Parameter	Description	Value
α	the control parameter of the BIA	$[1.8, 2.8]$
D_{max}	max. nb. of customers removed by the Random remove-and-repair operator	$0.25 * n_{routed}$
$\mathcal{M}_{size}$	size of the adaptive memory	100
$iter_{init}$	nb. of iterations of the initialization heuristic	1000
$iter_{max}$	nb. of iterations of the main algorithm	$30 * n/m$
$iter_{ils}$	nb. of iterations of the ILS	10

Table A.13 – Results for Solomon's instances with m = 1 using the slow setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
c101	320	320	0.00	320	0.00	9.5	c201	870	870	0.00	870	0.00	39.1
c102	360	360	0.00	360	0.00	29.9	c202	930	930	0.00	930	0.00	67.7
c103	400	400	0.00	400	0.00	47.5	c203	960	960	0.00	960	0.00	83.0
c104	420	420	0.00	420	0.00	89.4	c204	980	980	0.00	980	0.00	169.0
c105	340	340	0.00	340	0.00	13.2	c205	910	910	0.00	910	0.00	99.0
c106	340	340	0.00	340	0.00	18.8	c206	930	930	0.00	930	0.00	65.8
c107	370	370	0.00	370	0.00	19.7	c207	930	930	0.00	930	0.00	69.0
c108	370	370	0.00	370	0.00	25.7	c208	950	950	0.00	950	0.00	50.4
c109	380	380	0.00	380	0.00	38.3							
r101	198	198	0.00	198	0.00	4.7	r201	797	797	0.00	797	0.00	61.6
r102	286	286	0.00	286	0.00	17.4	r202	930	930	0.00	927.4	0.28	99.3
r103	293	293	0.00	293	0.00	29.1	r203	1028	1028	0.00	1028	0.00	156.3
r104	303	303	0.00	303	0.00	42.2	r204	1093	1093	0.00	1093	0.00	266.1
r105	247	247	0.00	247	0.00	16.8	r205	953	953	0.00	953	0.00	107.1
r106	293	293	0.00	293	0.00	23.6	r206	1032	1032	0.00	1032	0.00	134.5
r107	299	299	0.00	299	0.00	27.9	**r207**	**1078**	**1078**	0.00	1078	0.00	136.2
r108	308	308	0.00	308	0.00	37.2	**r208**	**1118**	**1118**	0.00	1118	0.00	352.1
r109	277	277	0.00	277	0.00	26.1	**r209**	**962**	**962**	0.00	962	0.00	164.5
r110	284	284	0.00	284	0.00	23.9	**r210**	**1002**	**1002**	0.00	1001.4	0.06	158.3
r111	297	297	0.00	297	0.00	29.5	r211	1051	1051	0.00	1051	0.00	202.0
r112	298	298	0.00	298	0.00	31.7							
rc101	219	219	0.00	219	0.00	10.3	rc201	795	795	0.00	795	0.00	43.9
rc102	266	266	0.00	266	0.00	23.0	rc202	938	938	0.00	938	0.00	86.8
rc103	266	266	0.00	266	0.00	19.9	rc203	1003	1003	0.00	1002.4	0.06	248.6
rc104	301	301	0.00	301	0.00	26.6	**rc204**	**1143**	**1143**	0.00	1139.4	0.31	256.1
rc105	244	244	0.00	244	0.00	13.0	rc205	859	859	0.00	859	0.00	67.1
rc106	252	252	0.00	252	0.00	17.4	rc206	899	899	0.00	896.6	0.27	73.4
rc107	277	277	0.00	277	0.00	21.7	rc207	983	983	0.00	983	0.00	119.1
rc108	298	298	0.00	298	0.00	28.3	**rc208**	**1058**	**1058**	0.00	1055	0.28	159.4
Avg.	303.7	303.7	0.00	303.7	0.00	26.29		969.7	969.7	0.00	969.2	0.05	130.9

Table A.14 – Results for Solomon's instances with m = 2 using the slow setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
c101	590	590	0.00	590	0.00	17.1	c201	1460	1460	0.00	1458	0.14	54.3
c102	660	660	0.00	660	0.00	22.8	c202	1470	1470	0.00	1470	0.00	84.6
c103	720	720	0.00	720	0.00	41.6	c203	1480	1480	0.00	1478	0.14	129.2
c104	760	760	0.00	760	0.00	71.1	c204	1490	1490	0.00	1490	0.00	225.5
c105	640	640	0.00	640	0.00	25.7	c205	1470	1470	0.00	1470	0.00	36.4
c106	620	620	0.00	620	0.00	14.3	c206	1480	1480	0.00	1480	0.00	27.1
c107	670	670	0.00	670	0.00	17.7	c207	1490	1490	0.00	1486	0.27	65.5
c108	680	680	0.00	680	0.00	19.2	c208	1490	1490	0.00	1490	0.00	57.6
c109	720	720	0.00	720	0.00	41.9							
r101	349	349	0.00	349	0.00	12.5	**r201**	1260	**1260**	0.00	1254.6	0.43	95.0
r102	508	508	0.00	508	0.00	18.0	**r202**	1353	**1353**	0.00	1351.6	0.10	228.9
r103	522	522	0.00	521.6	0.08	24.8	r203	1431	1430	0.07	1425.6	0.38	371.6
r104	552	552	0.00	552	0.00	34.1	r204	1458	1458	0.00	1458	0.00	1.7
r105	453	453	0.00	453	0.00	17.0	**r205**	1402	**1402**	0.00	1396.6	0.39	275.8
r106	529	529	0.00	529	0.00	19.9	**r206**	1452	**1452**	0.00	1450.8	0.08	297.6
r107	538	538	0.00	538	0.00	29.6	r207	1458	1458	0.00	1458	0.00	2.7
r108	560	560	0.00	560	0.00	32.1	r208	1458	1458	0.00	1458	0.00	0.2
r109	506	506	0.00	506	0.00	20.5	**r209**	**1423**	**1423**	0.00	1419.4	0.25	344.9
r110	525	525	0.00	525	0.00	21.0	**r210**	**1438**	**1438**	0.00	1432	0.42	431.5
r111	544	544	0.00	544	0.00	24.3	r211	1458	1458	0.00	1458	0.00	17.5
r112	544	544	0.00	544	0.00	31.8							
rc101	427	427	0.00	427	0.00	13.9	**rc201**	**1386**	**1386**	0.00	1384.6	0.10	64.2
rc102	505	505	0.00	504.8	0.04	22.3	**rc202**	**1523**	**1523**	0.00	1519.8	0.21	136.9
rc103	524	524	0.00	524	0.00	18.7	**rc203**	**1639**	**1639**	0.00	1636.6	0.15	197.0
rc104	575	575	0.00	574.8	0.03	30.2	**rc204**	**1718**	**1718**	0.00	1717	0.06	249.4
rc105	480	480	0.00	480	0.00	16.4	rc205	1462	1462	0.00	1461.8	0.01	126.3
rc106	483	483	0.00	482.8	0.04	18.6	rc206	1552	1552	0.00	1546.2	0.37	132.9
rc107	534	534	0.00	534	0.00	19.3	rc207	1609	1607	0.00	1604.4	0.16	176.0
rc108	556	556	0.00	556	0.00	27.1	**rc208**	**1705**	**1705**	0.00	1696	0.53	292.5
Avg.	561.2	561.2	0.00	561.1	0.01	24.94		1481.9	1481.9	0.01	1479.7	0.16	152.7

Table A.15 – Results for Solomon's instances with m = 3 using the slow setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
c101	810	810	0.00	810	0.00	30.9	c201	1810	1810	0.00	1810	0.00	0.7
c102	920	920	0.00	920	0.00	38.5	c202	1810	1810	0.00	1810	0.00	1.1
c103	990	990	0.00	986	0.40	77.3	c203	1810	1810	0.00	1810	0.00	1.6
c104	1030	1030	0.00	1030	0.00	52.1	c204	1810	1810	0.00	1810	0.00	1.8
c105	870	870	0.00	870	0.00	18.7	c205	1810	1810	0.00	1810	0.00	0.3
c106	870	870	0.00	870	0.00	30.2	c206	1810	1810	0.00	1810	0.00	0.2
c107	910	910	0.00	910	0.00	21.4	c207	1810	1810	0.00	1810	0.00	0.2
c108	920	920	0.00	920	0.00	37.4	c208	1810	1810	0.00	1810	0.00	0.1
c109	970	970	0.00	970	0.00	51.5							
r101	484	484	0.00	484	0.00	15.3	**r201**	**1450**	**1450**	0.00	1442	0.55	204.4
r102	694	694	0.00	694	0.00	21.1	r202	1458	1458	0.00	1458	0.00	0.6
r103	747	747	0.00	747	0.00	29.6	r203	1458	1458	0.00	1458	0.00	0.1
r104	778	778	0.00	778	0.00	33.3	r204	1458	1458	0.00	1458	0.00	0.1
r105	620	620	0.00	620	0.00	12.0	r205	1458	1458	0.00	1458	0.00	0.1
r106	729	729	0.00	729	0.00	30.6	r206	1458	1458	0.00	1458	0.00	0.1
r107	760	760	0.00	760	0.00	43.6	r207	1458	1458	0.00	1458	0.00	0.1
r108	797	797	0.00	797	0.00	27.2	r208	1458	1458	0.00	1458	0.00	0.1
r109	710	710	0.00	710	0.00	32.7	r209	1458	1458	0.00	1458	0.00	0.1
r110	737	737	0.00	736.8	0.03	22.6	r210	1458	1458	0.00	1458	0.00	0.1
r111	774	774	0.00	774	0.00	27.8	r211	1458	1458	0.00	1458	0.00	0.1
r112	776	776	0.00	776	0.00	33.8							
rc101	621	621	0.00	621	0.00	16.5	rc201	1698	1698	0.00	1694	0.24	120.5
rc102	714	714	0.00	712.8	0.17	15.0	rc202	1724	1724	0.00	1724	0.00	13.9
rc103	764	764	0.00	760.6	0.45	33.8	rc203	1724	1724	0.00	1724	0.00	0.1
rc104	835	835	0.00	835	0.00	39.5	rc204	1724	1724	0.00	1724	0.00	0.2
rc105	682	682	0.00	682	0.00	18.2	rc205	1719	1719	0.00	1710	0.52	156.9
rc106	706	706	0.00	705.4	0.08	25.4	rc206	1724	1724	0.00	1724	0.00	0.2
rc107	773	773	0.00	773	0.00	27.5	rc207	1724	1724	0.00	1724	0.00	0.2
rc108	795	795	0.00	795	0.00	32.8	rc208	1724	1724	0.00	1724	0.00	0.1
Avg.	785.7	785.7	0.00	785.4	0.04	30.90		1639.7	1639.7	0.00	1638.9	0.05	18.7

Table A.16 – Results for Solomon's instances with m = 4 using the slow setting.

		MS-ILS						MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
c101	1020	1020	0.00	1020	0.00	20.8	c201	1810	1810	0.00	1810	0.00	0.0
c102	1150	1150	0.00	1150	0.00	30.0	c202	1810	1810	0.00	1810	0.00	0.1
c103	1210	1210	0.00	1208	0.17	60.6	c203	1810	1810	0.00	1810	0.00	0.1
c104	1260	1260	0.00	1260	0.00	70.4	c204	1810	1810	0.00	1810	0.00	0.1
c105	1070	1060	0.93	1060	0.93	26.9	c205	1810	1810	0.00	1810	0.00	0.1
c106	1080	1080	0.00	1078	0.19	40.2	c206	1810	1810	0.00	1810	0.00	0.1
c107	1120	1120	0.00	1120	0.00	34.8	c207	1810	1810	0.00	1810	0.00	0.1
c108	**1140**	**1140**	0.00	1132	0.70	57.1	c208	1810	1810	0.00	1810	0.00	0.1
c109	1190	1190	0.00	1190	0.00	61.9							
r101	611	611	0.00	611	0.00	16.0	r201	1458	1458	0.00	1458	0.00	0.1
r102	843	843	0.00	840	0.36	29.4	r202	1458	1458	0.00	1458	0.00	0.1
r103	928	928	0.00	928	0.00	41.4	r203	1458	1458	0.00	1458	0.00	0.1
r104	**975**	**975**	0.00	975	0.00	50.5	r204	1458	1458	0.00	1458	0.00	0.1
r105	778	778	0.00	776	0.26	22.4	r205	1458	1458	0.00	1458	0.00	0.1
r106	906	906	0.00	905	0.11	23.5	r206	1458	1458	0.00	1458	0.00	0.1
r107	950	950	0.00	950	0.00	39.2	r207	1458	1458	0.00	1458	0.00	0.1
r108	995	995	0.00	994.6	0.04	52.3	r208	1458	1458	0.00	1458	0.00	0.1
r109	885	885	0.00	885	0.00	37.2	r209	1458	1458	0.00	1458	0.00	0.1
r110	915	915	0.00	915	0.00	26.9	r210	1458	1458	0.00	1458	0.00	0.1
r111	952	952	0.00	950.8	0.13	38.9	r211	1458	1458	0.00	1458	0.00	0.1
r112	**974**	**974**	0.00	973.6	0.04	41.3							
rc101	811	811	0.00	811	0.00	25.9	rc201	1724	1724	0.00	1724	0.00	0.1
rc102	909	908	0.11	904.4	0.51	26.5	rc202	1724	1724	0.00	1724	0.00	0.1
rc103	975	975	0.00	975	0.00	39.0	rc203	1724	1724	0.00	1724	0.00	0.1
rc104	1065	1065	0.00	1065	0.00	28.8	rc204	1724	1724	0.00	1724	0.00	0.1
rc105	875	875	0.00	875	0.00	24.6	rc205	1724	1724	0.00	1724	0.00	0.1
rc106	909	909	0.00	909	0.00	33.7	rc206	1724	1724	0.00	1724	0.00	0.1
rc107	987	987	0.00	986	0.10	46.9	rc207	1724	1724	0.00	1724	0.00	0.1
rc108	1025	1025	0.00	1025	0.00	28.9	rc208	1724	1724	0.00	1724	0.00	0.1
Avg.	983.0	982.7	0.04	981.8	0.12	37.10		1641.1	1641.1	0.00	1641.1	0.00	0.1

Table A.17 – Results for Cordeau's instances with m = 1 using the slow setting.

		MS-ILS						MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
pr01	308	308	0.00	308	0.00	12.0	pr11	353	353	0.00	353	0.00	10.6
pr02	404	404	0.00	404	0.00	37.2	pr12	442	441	0.23	441	0.23	37.4
pr03	394	394	0.00	394	0.00	48.2	pr13	467	467	0.00	467	0.00	96.1
pr04	489	489	0.00	489	0.00	158.2	pr14	567	555	2.12	555	2.12	209.2
pr05	595	595	0.00	595	0.00	172.6	pr15	708	708	0.00	708	0.00	430.6
pr06	591	591	0.00	590.4	0.10	331.7	pr16	674	650	3.56	650	3.56	444.2
pr07	298	298	0.00	298	0.00	12.0	pr17	362	362	0.00	362	0.00	15.0
pr08	463	463	0.00	463	0.00	61.8	pr18	539	539	0.00	539	0.00	81.6
pr09	493	493	0.00	493	0.00	106.7	pr19	562	560	0.36	554.2	1.39	279.7
pr10	594	594	0.00	594	0.00	371.6	pr20	667	652	2.25	651.2	2.37	445.4
Avg.	462.9	462.9	0.00	462.84	0.01	131.2		534.1	528.7	0.85	528.04	0.97	205.0

Table A.18 – Results for Cordeau's instances with m = 2 using the slow setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
pr01	502	502	0.00	502	0.00	9.3	pr11	566	566	0.00	565.6	0.07	24.4
pr02	715	713	0.28	713	0.28	48.0	pr12	774	768	0.78	766.2	1.01	50.5
pr03	742	742	0.00	742	0.00	50.0	**pr13**	**845**	**845**	0.00	840.2	0.57	113.4
pr04	**928**	**928**	0.00	928	0.00	95.5	pr14	1017	996	2.06	994.8	2.18	251.4
pr05	**1103**	**1103**	0.00	1093.4	0.87	181.8	**pr15**	**1237**	**1237**	0.00	1230.2	0.55	383.6
pr06	1076	1076	0.00	1074.2	0.17	382.0	pr16	1231	1199	2.60	1190.6	3.28	666.2
pr07	566	566	0.00	566	0.00	24.4	pr17	652	646	0.92	646	0.92	22.2
pr08	834	834	0.00	834	0.00	56.2	**pr18**	**955**	**955**	0.00	955	0.00	181.5
pr09	909	909	0.00	909	0.00	166.2	**pr19**	**1041**	**1041**	0.00	1040.2	0.08	334.1
pr10	**1145**	**1145**	0.00	1137	0.70	230.8	**pr20**	**1251**	**1251**	0.00	1246.8	0.34	514.0
Avg.	852	851.8	0.03	849.86	0.20	124.4		956.9	949.8	0.64	947.16	0.91	254.1

Table A.19 – Results for Cordeau's instances with m = 3 using the slow setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
pr01	622	622	0.00	620.8	0.19	16.6	pr11	654	654	0.00	654	0.00	28.5
pr02	**945**	**945**	0.00	943.8	0.13	61.4	pr12	1002	1000	0.20	997.6	0.44	71.0
pr03	**1014**	**1014**	0.00	1012.6	0.14	110.3	**pr13**	**1159**	**1159**	0.00	1155	0.35	143.3
pr04	**1296**	**1296**	0.00	1286.8	0.71	122.7	**pr14**	**1375**	**1375**	0.00	1365.4	0.70	335.5
pr05	**1500**	**1500**	0.00	1488.8	0.75	291.5	**pr15**	**1694**	**1694**	0.00	1683.4	0.63	601.5
pr06	**1515**	**1515**	0.00	1514.6	0.03	351.2	pr16	1668	1651	1.02	1638.8	1.75	736.1
pr07	744	744	0.00	744	0.00	19.5	pr17	841	841	0.00	838.8	0.26	39.7
pr08	**1141**	**1141**	0.00	1137	0.35	120.5	**pr18**	**1289**	**1289**	0.00	1279	0.78	123.4
pr09	**1276**	**1276**	0.00	1271.4	0.36	168.7	**pr19**	**1428**	**1428**	0.00	1420.4	0.53	365.7
pr10	**1582**	**1582**	0.00	1577.8	0.27	424.4	**pr20**	**1722**	**1722**	0.00	1705.6	0.95	1049.1
Avg.	1163.5	1163.5	0.00	1159.76	0.29	168.7		1283.2	1281.3	0.12	1273.8	0.64	349.4

Table A.20 – Results for Cordeau's instances with m = 4 using the slow setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
pr01	657	657	0.00	657	0.00	0.0	pr11	657	657	0.00	657	0.00	0.0
pr02	**1083**	**1083**	0.00	1078.2	0.44	85.0	**pr12**	**1133**	**1133**	0.00	1129.4	0.32	120.1
pr03	**1247**	**1247**	0.00	1239.4	0.61	75.9	**pr13**	**1392**	**1392**	0.00	1385.8	0.45	170.5
pr04	**1595**	**1595**	0.00	1587	0.50	242.5	**pr14**	**1688**	**1688**	0.00	1682.6	0.32	473.1
pr05	**1858**	**1858**	0.00	1841.2	0.90	482.1	**pr15**	**2085**	**2085**	0.00	2063.4	1.04	669.6
pr06	**1894**	**1894**	0.00	1883.8	0.54	487.7	pr16	2065	2056	0.44	2040	1.21	908.9
pr07	876	876	0.00	872.4	0.41	36.7	**pr17**	**936**	**936**	0.00	932	0.43	95.3
pr08	**1390**	**1390**	0.00	1378.8	0.81	142.2	**pr18**	**1554**	**1554**	0.00	1541.2	0.82	171.6
pr09	**1622**	**1622**	0.00	1615.2	0.42	246.8	**pr19**	**1780**	**1780**	0.00	1763.6	0.92	540.6
pr10	**1965**	**1965**	0.00	1956.4	0.44	613.4	pr20	2115	2098	0.80	2085.8	1.38	1007.3
Avg.	1418.7	1418.7	0.00	1410.94	0.51	241.2		1538.8	1537.9	0.12	1528.08	0.69	415.7

Table A.21 – Results for Solomon's instances of "OPT" data set using the slow setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
c101	1810	1810	0.00	1810	0.00	3.7	c201	1810	1810	0.00	1810	0.00	0.0
c102	1810	1810	0.00	1810	0.00	0.1	c202	1810	1810	0.00	1810	0.00	0.1
c103	1810	1810	0.00	1810	0.00	0.1	c203	1810	1810	0.00	1810	0.00	0.1
c104	1810	1810	0.00	1810	0.00	0.1	c204	1810	1810	0.00	1810	0.00	0.1
c105	1810	1810	0.00	1810	0.00	0.1	c205	1810	1810	0.00	1810	0.00	0.1
c106	1810	1810	0.00	1810	0.00	1.0	c206	1810	1810	0.00	1810	0.00	0.1
c107	1810	1810	0.00	1810	0.00	0.0	c207	1810	1810	0.00	1810	0.00	0.1
c108	1810	1810	0.00	1810	0.00	0.1	c208	1810	1810	0.00	1810	0.00	0.1
c109	1810	1810	0.00	1810	0.00	0.1							
r101	1458	1458	0.00	1458	0.00	3.0	r201	1458	1458	0.00	1458	0.00	0.1
r102	1458	1458	0.00	1458	0.00	6.8	r202	1458	1458	0.00	1458	0.00	1.6
r103	1458	1458	0.00	1458	0.00	198.5	r203	1458	1458	0.00	1458	0.00	0.1
r104	1458	1458	0.00	1452.6	0.37	160.0	r204	1458	1458	0.00	1458	0.00	2.6
r105	1458	1458	0.00	1458	0.00	31.4	r205	1458	1458	0.00	1458	0.00	0.1
r106	1458	1458	0.00	1458	0.00	6.9	r206	1458	1458	0.00	1458	0.00	0.1
r107	1458	1458	0.00	1457.4	0.04	157.3	r207	1458	1458	0.00	1458	0.00	1.9
r108	1458	1458	0.00	1457.2	0.05	121.9	r208	1458	1458	0.00	1458	0.00	0.3
r109	1458	1458	0.00	1456.8	0.08	52.6	r209	1458	1458	0.00	1458	0.00	0.1
r110	1458	1458	0.00	1456	0.14	135.2	r210	1458	1458	0.00	1458	0.00	0.1
r111	1458	1457	0.07	1455	0.21	129.3	r211	1458	1458	0.00	1458	0.00	9.0
r112	1458	1456	0.14	1454.6	0.23	109.4							
rc101	1724	1724	0.00	1723.2	0.05	68.0	rc201	1724	1724	0.00	1724	0.00	0.1
rc102	1724	1713	0.64	1713	0.64	149.6	rc202	1724	1724	0.00	1724	0.00	13.4
rc103	1724	1724	0.00	1724	0.00	19.6	rc203	1724	1724	0.00	1724	0.00	0.1
rc104	1724	1724	0.00	1724	0.00	0.7	rc204	1724	1724	0.00	1724	0.00	0.2
rc105	1724	1724	0.00	1715	0.52	127.2	rc205	1724	1724	0.00	1724	0.00	0.1
rc106	1724	1724	0.00	1718	0.35	92.7	rc206	1724	1724	0.00	1724	0.00	0.3
rc107	1724	1724	0.00	1724	0.00	4.3	rc207	1724	1724	0.00	1724	0.00	0.1
rc108	1724	1724	0.00	1724	0.00	5.8	rc208	1724	1724	0.00	1724	0.00	0.1
Avg.	1640.6	1640.1	0.03	1639.1	0.09	54.66		1641.1	1641.1	0.00	1641.1	0.00	1.1

Table A.22 – Results for Cordeau's instances of "OPT" data set using the slow setting.

		MS-ILS							MS-ILS				
instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}	instance	BKS	Z_{max}	$rpe\%$	Z_{avg}	$arpe\%$	cpu_{avg}
pr01	657	622	5.33	620.2	5.60	18.4	pr06	3671	3671	0.00	3671	0.00	11.8
pr02	1220	1216	0.33	1213.2	0.56	226.7	pr07	948	943	0.53	941.8	0.65	44.9
pr03	1788	1785	0.17	1784	0.22	621.7	pr08	2006	2006	0.00	2006	0.00	5.2
pr04	2477	2477	0.00	2476.6	0.02	506.4	pr09	2736	2736	0.00	2736	0.00	0.2
pr05	3351	3351	0.00	3351	0.00	12.1	pr10	3850	3850	0.00	3850	0.00	0.3
Avg.	1898.6	1890.2	1.16	1889.0	1.28	277.1		2642.2	2641.2	0.11	2641.0	0.13	12.5

Detailed results of the NS-SC on 2E-VRP benchmarks

In this appendix, we present the detailed results obtained by our Neighborhood Search and Set Cover Hybrid Heuristic (NS-SC) on all the 2E-VRP benchmark instances.

B.1–**??** show the detailed results obtained by our NS-SC. For each instance, we report the best obtained solution, the average solution value of five runs, and the average computation time. The average and the best objective value of the five runs are given in columns "Avg. 5" and "Best 5", respectively. Column "CPU" shows the average runtime of the algorithm in seconds. The column "BKS" refers to the best-known solution of that set of instances.

Table B.3 – Results on Set 4a instances.

	NS-SC			BKS
Instance	Avg. 5	Best.5	CPU	
1	1569,42	1569,42	20	1569,42
2	1438,93	1438,32	25	1438,33
3	1570,43	1570,43	27	1570,43
4	1424,04	1424,04	22	1424,04
5	2193,52	2193,52	27	2193,52
6	1279,89	1279,89	36	1279,87
7	1458,60	1458,60	22	1458,63
8	1363,92	1363,76	25	1363,74
9	1450,25	1450,25	31	1450,27
10	1407,65	1407,65	28	1407,64
11	2052,99	2047,62	21	2047,46
12	1209,46	1209,46	25	1209,42
13	1481,80	1481,80	22	1481,83
14	1393,64	1393,64	24	1393,61
15	1489,92	1489,92	20	1489,94

Continued on next page

Table B.3 – *Continued from previous page*

16	1389,20	1389,20	29	1389,17
17	2088,88	2088,48	23	2088,49
18	1227,68	1227,68	26	1227,61
19	1564,66	1564,66	19	1564,66
20	1272,98	1272,98	24	1272,97
21	1577,82	1577,82	19	1577,82
22	1281,83	1281,83	22	1281,83
23	1807,35	1807,35	23	1807,35
24	1282,69	1282,69	24	1282,68
25	1527,46	1522,40	23	1522,42
26	1167,47	1167,47	24	1167,46
27	1481,91	1481,91	27	1481,57
28	1210,46	1210,46	25	1210,44
29	1723,22	1722,30	19	1722,04
30	1211,63	1211,63	23	1211,59
31	1490,32	1490,32	26	1490,34
32	1199,05	1199,05	18	1199,00
33	1509,94	1508,32	27	1508,30
34	1233,96	1233,96	22	1233,92
35	1718,42	1718,42	26	1718,41
36	1228,95	1228,95	22	1228,89
37	1528,73	1528,73	33	1528,73
38	1170,84	1169,20	28	1169,20
39	1520,92	1520,92	27	1520,92
40	1199,42	1199,42	35	1199,42
41	1667,96	1667,96	29	1667,96
42	1194,54	1194,54	34	1194,54
43	1439,67	1439,67	33	1439,67
44	1045,14	1045,14	37	1045,13
45	1456,19	1450,95	33	1450,96
46	1088,79	1088,79	41	1088,77
47	1593,38	1587,29	30	1587,29
48	1082,21	1082,21	31	1082,20
49	1434,88	1434,88	28	1434,88
50	1085,07	1083,16	20	1083,12
51	1398,04	1398,04	33	1398,05
52	1125,69	1125,69	29	1125,67
53	1567,79	1567,79	34	1567,77
54	1127,66	1127,66	29	1127,61
Avg	1420,50	1419,97	22	1419,94

Table B.4 – Results on Set 4b instances.

	NS-SC			BKS
Instance	Avg. 5	Best.5	CPU	
1	1569,42	1569,42	28	1569,42
2	1442,23	1438,32	33	1438,33
3	1570,43	1570,43	28	1570,43
4	1424,04	1424,04	31	1424,04
5	2193,64	2193,52	36	2193,52
6	1279,89	1279,89	30	1279,87
7	1408,58	1408,58	32	1408,57
8	1360,32	1360,32	38	1360,32
9	1403,53	1403,53	29	1403,53
10	1360,54	1360,54	34	1360,56
11	2052,99	2047,43	60	2047,46
12	1209,46	1209,46	36	1209,42
13	1450,94	1450,94	29	1450,93
14	1393,64	1393,64	27	1393,61
15	1466,84	1466,84	28	1466,83
16	1387,85	1387,85	24	1387,83
17	2089,19	2088,48	37	2088,49
18	1227,68	1227,68	34	1227,61
19	1548,16	1546,28	27	1546,28
20	1272,98	1272,98	26	1272,97
21	1577,82	1577,82	26	1577,82
22	1281,83	1281,83	26	1281,83
23	1652,98	1652,98	29	1652,98
24	1282,69	1282,69	28	1282,68
25	1408,58	1408,58	29	1408,57
26	1167,47	1167,47	28	1167,46
27	1444,49	1444,49	28	1444,50
28	1210,46	1210,46	34	1210,44
29	1552,66	1552,66	27	1552,66
30	1211,63	1211,63	36	1211,59
31	1441,48	1440,85	29	1440,86
32	1199,05	1199,05	29	1199,00
33	1478,87	1478,87	27	1478,86
34	1233,96	1233,96	32	1233,92
35	1570,73	1570,73	31	1570,72
36	1228,95	1228,95	32	1228,89

Continued on next page

Table B.4 – *Continued from previous page*

37	1528,73	1528,73	35	1528,73
38	1163,07	1163,07	26	1163,07
39	1520,92	1520,92	34	1520,92
40	1163,04	1163,04	39	1163,04
41	1652,98	1652,98	31	1652,98
42	1190,68	1190,17	36	1190,17
43	1407,09	1406,10	44	1406,11
44	1035,64	1035,05	42	1035,03
45	1405,37	1402,41	47	1401,87
46	1058,54	1058,10	48	1058,11
47	1552,66	1552,66	40	1552,66
48	1074,51	1074,51	34	1074,50
49	1436,07	1434,88	36	1434,88
50	1065,30	1065,30	34	1065,25
51	1387,51	1387,51	38	1387,51
52	1103,50	1103,47	34	1103,42
53	1545,76	1545,76	31	1545,73
54	1113,66	1113,66	29	1113,62
Avg	1397,43	1397,07	33	1397,04

Table B.1 – Results for Set2 instances.

	NS-SC			BKS
Instance	Avg	Best5	CPU	
Set 2a				
E-n22-k4-s10-14	371.50	371.50	1	371.50
E-n22-k4-s11-12	427.22	427.22	1	427.22
E-n22-k4-s12-16	392.78	392.78	1	392.78
E-n22-k4-s6-17	417.07	417.07	1	417.07
E-n22-k4-s8-14	384.96	384.96	1	384.96
E-n22-k4-s9-19	470.60	470.60	1	470.60
E-n33-k4-s14-22	779.05	779.05	5	779.05
E-n33-k4-s1-9	730.16	730.16	7	730.16
E-n33-k4-s2-13	714.63	714.63	5	714.63
E-n33-k4-s3-17	707.48	707.48	5	707.48
E-n33-k4-s4-5	778.74	778.74	7	778.74
E-n33-k4-s7-25	756.85	756.85	7	756.85
Avg	577.59	577.59	3	577.59
Set 2b				
E-n51-k5-s11-19	581.64	581.64	23	581.64
E-n51-k5-s11-19-27-47	527.63	527.63	18	527.63
E-n51-k5-s2-17	597.49	597.49	24	597.49
E-n51-k5-s2-4-17-46	530.76	530.76	21	530.76
E-n51-k5-s27-47	538.22	538.22	22	538.22
E-n51-k5-s32-37	552.28	552.28	21	552.28
E-n51-k5-s4-46	530.76	530.76	20	530.76
E-n51-k5-s6-12	554.81	554.81	24	554.81
E-n51-k5-s6-12-32-37	531.92	531.92	19	531.92
Avg	549.50	549.50	21	549.50
Set 2c				
E-n51-k5-s11-19	617.42	617.42	24	617.42
E-n51-k5-s11-19-27-47	530.76	530.76	18	530.76
E-n51-k5-s2-17	601.39	601.39	25	601.39
E-n51-k5-s2-4-17-46	601.39	601.39	25	601.39
E-n51-k5-s27-47	530.76	530.76	21	530.76
E-n51-k5-s32-37	752.59	752.59	30	752.59
E-n51-k5-s4-46	702.33	702.33	24	702.33
E-n51-k5-s6-12	567.42	567.42	22	567.42
E-n51-k5-s6-12-32-37	567.42	567.42	23	567.42
Avg	607.94	607.94	23	607.94

Table B.2 – Results for Set3 instances.

	NS-SC			BKS
Instance	Avg	Best 5	CPU	
Set 3a				
E-n22-k4-s13-14	526.14	526.14	1	526.15
E-n22-k4-s13-16	521.10	521.10	1	521.09
E-n22-k4-s13-17	496.39	496.39	1	496.38
E-n22-k4-s14-19	498.81	498.81	1	498.80
E-n22-k4-s17-19	512.80	512.80	1	512.80
E-n22-k4-s19-21	520.41	520.41	1	520.42
E-n33-k4-s16-22	672.19	672.19	6	672.17
E-n33-k4-s16-24	666.03	666.03	6	666.02
E-n33-k4-s19-26	680.38	680.38	4	680.37
E-n33-k4-s22-26	680.38	680.38	4	680.37
E-n33-k4-s24-28	670.41	670.41	4	670.43
E-n33-k4-s25-28	650.55	650.55	4	650.58
Avg	591.30	591.30	3	591.30
Set 3b				
E-n51-k5-s12-18	690.57	690.57	28	690.59
E-n51-k5-s12-41	683.01	683.01	28	683.05
E-n51-k5-s12-43	710.39	710.39	25	710.41
E-n51-k5-s39-41	728.50	728.50	28	728.54
E-n51-k5-s40-41	723.71	723.71	29	723.75
E-n51-k5-s40-43	752.15	752.15	29	752.15
Avg	714.72	714.72	28	714.75
Set 3c				
E-n51-k5-s13-19	560.71	560.71	25	560.73
E-n51-k5-s13-42	564.43	564.43	23	564.45
E-n51-k5-s13-44	564.43	564.43	24	564.45
E-n51-k5-s40-42	746.27	746.27	31	746.31
E-n51-k5-s41-42	771.54	771.54	28	771.56
E-n51-k5-s41-44	802.91	802.91	31	802.91
Avg	668.38	668.38	27	668.40

Table B.5 – Results on Set 5 instances.

	NS-SC				BKS
Instance	Avg	Best 5	Best	CPU	
100-5-1	1569.47	1564.46	1564.46	598	1564.46
100-5-1b	1112.89	1103.55	1103.55	694	1103.55
100-5-2	1017.44	1016.33	1016.32	708	1016.32
100-5-2b	782.29	782.29	782.25	617	782.25
100-5-3	1045.29	1045.29	1045.29	392	1045.29
100-5-3b	828.55	828.55	828.54	481	828.54
Avg	1059.32	1056.75	1056.74	582	1056.74
100-10-1	1129.47	1128.64	1124.93	560	1124.93
100-10-1b	912.62	911.95	911.95	563	911.95
100-10-2	1005.60	997.29	993.34	524	985.40
100-10-2b	772.88	766.28	766.28	647	766.28
100-10-3	1046.26	1042.66	1042.63	455	1042.63
100-10-3b	864.41	861.87	849.73	595	849.73
Avg	955.21	951.45	948.14	557	946.82
200-10-1	1548.47	1539.76	1539.76	900	1538.35
200-10-1b	1186.24	1175.81	1175.81	900	1175.81
200-10-2	1368.75	1361.58	1352.87	900	1352.87
200-10-2b	1003.22	995.20	986.48	900	986.48
200-10-3	1817.55	1814.23	1779.68	900	1779.68
200-10-3b	1208.28	1205.12	1196.93	900	1196.93
Avg	1355.42	1348.62	1338.59	900	1338.35

Table B.6 – Results on Set 6a instances.

	NS-SC			BKS
Instances	Avg5	Best 5	CPU	
A-n51-4	652.00	652.00	32	652.00
A-n51-5	663.41	663.41	40	663.41
A-n51-6	662.51	662.51	61	662.51
A-n76-4	985.98	985.95	193	985.95
A-n76-5	979.15	979.15	322	979.15
A-n76-6	970.20	970.20	326	970.20
A-n101-4	1194.17	1194.17	479	1194.17
A-n101-5	1212.14	1211.35	646	1211.38
A-n101-6	1156.56	1155.94	655	1155.89
B-n51-4	563.98	563.98	27	563.98
B-n51-5	549.23	549.23	27	549.23
B-n51-6	556.32	556.32	36	556.32
B-n76-4	792.73	792.73	128	792.73
B-n76-5	783.93	783.93	135	783.93
B-n76-6	774.17	774.17	238	774.17
B-n101-4	939.54	939.21	419	939.21
B-n101-5	967.82	967.82	483	967.82
B-n101-6	960.29	960.29	582	960.29
C-n51-4	689.18	689.18	25	689.18
C-n51-5	723.12	723.12	33	723.12
C-n51-6	699.24	697.00	36	697.00
C-n76-4	1054.89	1054.89	233	1054.89
C-n76-5	1115.32	1115.32	228	1115.32
C-n76-6	1065.37	1065.37	312	1060.52
C-n101-4	1300.58	1297.42	621	1297.42
C-n101-5	1306.23	1306.23	533	1304.86
C-n101-6	1288.43	1284.48	617	1284.48
Avg	911.35	910.94	277	910.71

Table B.7 – Results on Set 6b instances.

	NS-SC			BKS
Instances	Avg5	Best 5	CPU	
A-n51-4	744.24	744.24	24	744.24
A-n51-5	811.52	811.52	30	811.52
A-n51-6	930.11	930.11	35	930.11
A-n76-4	1385.51	1385.51	167	1385.51
A-n76-5	1519.86	1519.86	228	1519.86
A-n76-6	1666.06	1666.06	263	1666.06
A-n101-4	1881.44	1881.44	718	1881.44
A-n101-5	1709.06	1709.06	629	1709.06
A-n101-6	1788.98	1781.96	675	1777.69
B-n51-4	653.09	653.09	33	653.09
B-n51-5	672.10	672.10	33	672.10
B-n51-6	767.13	767.13	35	767.13
B-n76-4	1094.52	1094.52	248	1094.52
B-n76-5	1218.67	1218.11	900	1218.13
B-n76-6	1328.24	1326.73	220	1326.76
B-n101-4	1500.55	1500.55	623	1500.55
B-n101-5	1396.83	1395.32	714	1395.32
B-n101-6	1448.94	1445.97	900	1445.97
C-n51-4	867.56	867.56	32	866.58
C-n51-5	943.12	943.12	39	943.12
C-n51-6	1050.42	1050.42	38	1050.42
C-n76-4	1438.96	1438.96	152	1438.96
C-n76-5	1745.47	1745.39	193	1745.39
C-n76-6	1756.45	1756.45	900	1756.54
C-n101-4	2070.72	2064.87	598	2064.86
C-n101-5	1969.04	1964.63	797	1964.63
C-n101-6	1863.23	1861.50	752	1860.73
Avg	1341.55	1340.60	369	1340.38