Dottorato di Ricerca in Informatica
Università di Bologna, Padova
Ciclo XXI
Settore scientifico disciplinare: INF/01

Algorithms for Network Design and Routing Problems

Enrico Bartolini

Marzo 2009

Coordinatore:	Tutore:
Prof. Simone Martini	Prof. Vittorio Maniezzo

To my family

Abstract

Many practical problems arising in real world applications, mainly in the fields of logistics and telecommunications, can be modeled as network design or vehicle routing problems. In this thesis we study three combinatorial optimization problems belonging to the classes of Network Design and Vehicle Routing problems that are strongly linked in the context of the design and management of transportation networks. The problems considered in this thesis are the Non-Bifurcated Capacitated Network Design Problem (NBP), the Period Vehicle Routing Problem (PVRP) and the Pickup and Delivery Problem with Time Windows (PDPTW). These problems are interesting from both a theoretical and a practical viewpoint. They are inherently hard to solve since they all belong to the class of $\mathcal{NP}$-hard problems and contain as special cases some well known difficult problems such as the Traveling Salesman Problem and the Steiner Tree Problem. Moreover, they model the core structure of many practical problems arising in logistics and telecommunications.

The NBP is the problem of designing the optimum network to satisfy a given set of traffic demands. Given a set of nodes, a set of potential links and a set of point-to-point demands called commodities, the objective is to design the network topology, i.e., to select the links to install and dimension their capacities so that all the demands can be routed between their respective endpoints, and the sum of link fixed costs and commodity routing costs is minimized. The problem is called non-bifurcated because the solution network must allow each demand to

follow a single path, i.e., the flow of each demand cannot be splitted. Although this is the case in many real applications such as production-distribution with single sourcing and express package delivery, the NBP has received significantly less attention in the literature than other capacitated network design problems that allow bifurcation.

We describe an exact algorithm for the NBP that is based on solving by an integer programming solver a formulation of the problem strengthened by simple valid inequalities and four new heuristic algorithms. One of these heuristics is an adaptive memory metaheuristic, based on partial enumeration, that could be applied to a wider class of structured combinatorial optimization problems.

In the PVRP a fleet of vehicles of identical capacity must be used to service a set of customers over a planning period of several days. Each customer specifies a service frequency, a set of allowable day-combinations, and a quantity of product that the customer must receive every time he is visited. For example, a customer may require to be visited twice during a 5-day period imposing that these visits take place on Monday-Thursday or Monday-Friday or Tuesday-Friday. The problem consists in simultaneously assigning a day-combination to each customer and in designing the vehicle routes for each day so that: each customer is visited the required number of times, the number of routes on each day does not exceed the number of vehicles available, and the total cost of the routes over the period is minimized.

With respect to other vehicle routing problems the PVRP can be classified as strategic because, in practice, in most applications the problem is solved over a limited planning period but the routes in the solution are operated unchanged for several months. We also consider a tactical variant of this problem, called Tactical Planning Vehicle Routing Problem, where customers require to be visited on a specific day of the period but a penalty cost, called service cost, can be paid to postpone the visit to a later day than that required. At our knowledge all the

algorithms proposed in the literature for the PVRP are heuristics.

In this thesis we present for the first time an exact algorithm for the PVRP that is based on different relaxations of a set partitioning-like formulation. The effectiveness of the proposed algorithm is tested on a set of instances from the literature and on a new set of instances.

Finally, we study the PDPTW which is well suited to model a wide range of distribution problems at the operational level. The problem is to service a set of transportation requests using a fleet of identical vehicles of limited capacity located at a central depot. Each request specifies a pickup location and a delivery location, and requires that a given quantity of load is transported from the pickup location to the delivery location. Moreover, each location can be visited only within an associated time window. Each vehicle can perform at most one route and the problem is to satisfy all the requests using the available vehicles so that: each request is serviced by a single vehicle, the load on each vehicle does not exceed the capacity, and all locations are visited according to their time window.

We formulate the PDPTW as a set partitioning-like problem with additional cuts and we propose an exact algorithm based on different relaxations of the mathematical formulation and a branch-and-cut-and-price algorithm. The new algorithm is tested on two classes of problems from the literature and compared with a recent branch-and-cut-and-price algorithm from the literature.

Acknowledgements

First of all I would like to express my sincere gratitude to my tutor, Professor Vittorio Maniezzo, for his guidance and support through these years, and for having made possible to take this journey.

I would like to express my thanks to Professor Aristide Mingozzi, Dr. Roberto Baldacci and Dr. Marco A. Boschetti for their suggestions and many useful discussions.

I am also grateful to Professor Jean-Francoise Cordeau and Professor Roberto Wolfler Calvo for their kind willingness to review this thesis and for providing useful comments.

Last but not least, I am deeply indebted to my parents and my sister who granted me unconditioned support and encouragements through these years. Without their moral and practical support I would never have completed this task.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1. Transportation planning

Freight transportation is a fundamental brick in modern economic systems. Business competitiveness is strictly bound to logistic efficiency since transportation accounts for a significant percentage of companies yearly expenses.

In 2001, road transport alone accounted for 45% of all freight transported between the 15 member states of the European Union (EU), measured in ton-kilometers (each corresponding to 1000 kg. of freight transported for one kilometer). Another significant transport type was sea shipping, accounting for 40.4%, whereas only 7.8% was transported by rail, 4% by inland waterways and 2.8% through pipelines. Excluding sea shipping, road transportation is undoubtedly the dominant transport mode, covering 75.5% of total ton-kilometers (Lafontaine and Valeri [80]). This fraction was only 60% in 1980 (European Commission [98]) and has kept growing significantly in the last decades making the economical impact of these transport services more and more relevant. In 1998, road carriers accounted for 5% of GDP in the EU (European Union [97]), and similar statistics were recorded in the U.S., where trucking accounted for about 6% of GDP in 2001 (Corsi [41]).

Motor carrier transportation has been regulated for many years. Deregulation took place in the U.S. with the Motor Carrier Act in 1980 and started more gradually in the EU with most of it completed by the early 1990s [80]. Deregulation significantly changed the approach of carrier companies to transportation planning. In a deregulated environment competitive rates together with a high quality of service are crucial in attracting and retaining customers. On the other hand, increasing cost of fuels, globalization of production and the corresponding supply chains, and liberalization of transportation services are posing new challenges together with new options for service carriers. They operate in a highly competitive environment where customers require reliable and efficient services, prices must be kept competitive and costs keep growing.

In this context, the optimization of logistic operations plays a fundamental role, and it is not surprising that starting from early 80s transportation planning has been the subject of a considerable amount of attention from the research community. Many prominent studies discuss planning and optimizations systems that have been successfully employed by major carriers. From an algorithmic viewpoint, real world logistic problems are extremely interesting as their solution poses formidable challenges. However, it is often vain to hope studying these problems as a whole. In practice, even modeling such problems with reasonable approximation can prove to be a formidable task due to the number of extremely complex and poorly defined constraints and conflicting objectives. Focusing on well structured and mathematically well behaved subproblems is common practice, and represents the key in designing optimization and decision support tools for solving more complex problems.

The aim of this thesis is to study three combinatorial optimization problems belonging to the classes of Network Design and Vehicle Routing problems that are strongly related in the context of freight distribution problems such as express package delivery, production-distribution planning, and less-than-truckload (LTL) transportation. Moreover, these problems appear as core subproblems in many practical problems in logistics.

LTL transportation planning is an example where network design and vehicle routing problems are strongly related. Carriers and companies facing this kind of problems routinely handle a large number of transportation orders, each involving small quantities of goods, that must be moved between widely spread locations. To exploit economies of scale in transportation costs and maximize truck loads, intermediate facilities (or break-bulks terminals) are used to "consolidate" (i.e. aggregate) several shipments into truckloads. Truckloads are dispatched to terminal facilities close to the customer location where they are broken down into individual shipments and finally delivered to the end customers.

In this context LTL operations can be divided into long distance distribution and regional distribution corresponding, from a planning level viewpoint, to tactical and operational decision levels [see 44, 47]. In long distance distribution shipments are routed through a network where nodes represent intermediate and terminal facilities, whereas line-haul trucks can be scheduled to routinely travel between facilities to connect them and represent arcs. Terminal facilities collect or deliver shipments from/to their local service region using their own local fleet of pickup/delivery trucks. From terminal facilities shipments are loaded on line-haul trucks that bring them to intermediate facilities for consolidation with other shipments sharing the same destination. Shipments incoming at intermediate facilities are then unloaded and rearranged to fill other line-haul trucks headed to the required destination.

Typically, in designing such network companies extrapolate from historical data a periodic forecast of freight volume (flow) that is to be moved between endpoint locations, based on their expectations. Using this point-to-point flow estimation, they schedule trucks or other transportation services on arcs of the network and define their frequency thus providing "transportation capacity" on the network. The capacity installed must suffice to simultaneously route the expected flows to the destination endpoints while minimizing the expenses for providing transportation service on arcs.

Depending on the application context, this kind of problems can be compli-

cated by a number of additional constrains. Multiple transportation options may be available (e.g., different trailer types, rail transports, aircrafts etc.) each having different capacity and operating cost. Transportation services may be required to be scheduled on paths (or tours) rather than on single arcs. In some applications it may be necessary to integrate the time component into the model to represent quality of service restrictions. Moreover, the objective function may be much more complicated, keeping into account various operational and service restrictions or quality and reliability of service. This class of problems is known in the literature as Service Network Design (see Crainic [44, 47], Wieberneit [120], Armacost et al. [4]) or Load Planning Problem (see e.g. Powell and Sheffi [103, 102]) in the context of LTL transportation for motor carriers. These problems belong to the wide class of Network Design problems. At their hearth is the basic network design model that captures for the most part the fundamental structure (and hence the hardness) of such problems.

As a result of this network design one obtains a plan for moving freights between end terminals. However, another fundamental problem arises on the regional scale at the operational level. When shipments reach their destination terminals they have to be dispatched to end customers or, equivalently, upcoming shipments within the terminal operating area must be collected in order to be shipped through the network. These operations define a regional distribution problem where a number of customers scattered across the terminal facility operating region must be serviced by a local fleet of vehicles. Very generally, these problems are referred to as Vehicle Routing Problems and have been extensively studied in the literature in many variants. Several additional constraints must be accounted for at this level to obtain a realistic model. As an example, an interesting overview of the local operations of a small package shipping firm can be found in Wong [121]. In this thesis we focus on vehicle routing problems involving pickup and delivery operations and time constraints. We also consider periodic problems where multiple customer visits must be scheduled over a short planning period. Interestingly, this latter variant lends itself to model problems

where customer requests are not completely known in advance but may be revealed in a dynamic fashion during the course of a given planning period. Such problems may be quite relevant in some applications such as package pickup and delivery.

In the following two sections we describe in more detail the classes of Network Design and Vehicle Routing problems. We describe the variants of such problems that are relevant for this thesis focusing on their models and applications.

1.2. Network Design Problems

Network design problems arise in many different application contexts such as transportation (e.g., airline, rails, freight traffic), communications (telephone and computer networks), electric power systems, oil, gas and water pipelines. In general, the problem can be stated as that of determining a minimum cost configuration of a network, i.e., to place links between nodes and install sufficient capacity to simultaneously route a set of point-to-point flows through the network. As an example, in a traffic network nodes correspond both to origin and destination areas for vehicular traffic and road intersections, arcs define the road network, and flows represent the amount of traffic between urban areas. The problem of selecting a set of road improvements (i.e., resize some roads) so as to minimize the total travel cost for all travelers subject to a budget constraint (i.e., a maximum incurred cost) can be modeled as a network design problem. Another example is the set up of private telecommunication networks. Here, links correspond to private lines whose capacity (i.e., transmission rate) is provided by the transmission facilities that are leased from a telephone company at a fixed rate. Given a traffic estimation between various locations of a company, a fundamental problem is to select a configuration of transmission facilities that is able to carry this traffic at minimum cost. Solving this problem also means to define a route for each point-to-point traffic requirement such that all traffic requirements can be routed using

the installed capacity.

The Capacitated Network Design Problem (CNDP) is defined on a graph representing the network where the arc set corresponds to the set of links available in the network. Between a given set of origin-destination node pairs there is a flow requirement, called commodity, that must be routed through the network from the corresponding origin node to the corresponding destination node. On each arc is available an initial capacity that can be expanded by installing an integer number of copies of some base capacity having fixed size. In general two types of costs are considered: routing costs and fixed costs. Routing costs are associated with each arc-commodity pair and represent the cost of shipping the flow of each commodity through each arc. Depending on the application context, routing costs may correspond to travel times, risk factors, penalties, or other costs reflecting the amount of traffic on the arcs. On the other hand, fixed costs represent the installation cost for each base capacity. There are many variants of this general model. For example models without capacity restrictions (i.e., base capacities have unlimited size [see 9]) or models with reliability and survivability constraints and special topological restrictions on the network (e.g., node degree constraints). This model is quite general and contains as special cases several difficult problems. Indeed, even computing the optimal paths for the commodities once fixed the link capacities corresponds to solving a capacitated multicommodity flow problem.

When there is no initial capacity on arcs, routing costs are zero, and the base capacity size is unlimited, the problem reduces to the Steiner Tree Problem (Hakimi [69],Dreyfus and Wagner [53]) that is known to be $\mathcal{NP}$-hard (see Karp [77]). To obtain the Steiner Tree Problem it suffices to define one commodity for each required node in the Steiner tree except one, say node 1, representing the root. For each commodity, let node 1 be the source node and define the destination node as a required node of the Steiner Tree Problem.

Another important special case is the Facility Location Problem (van Roy [119], Holmberg et al. [74]). To model the Facility Location Problem as a CNDP it suf-

fices to add a dummy node to the network, representing the source of all flows required by the customer nodes, and special arcs connecting the dummy node with each facility node. The cost of each special arc (i, j) is set equal to the fixed cost for opening facility j, the routing cost is set equal to 0, and the base capacity that can be installed on it is set equal to the capacity of facility j. Conversely, each arc connecting a facility node with a customer node has no fixed cost and an unlimited initial capacity, but each commodity traversing it pays a routing cost equal to the associated facility-customer transportation cost.

Due to the large number of different applications, the network design problem has been studied in many variants. When routing costs increase linearly with the amount of each commodity on arcs, and at most one base capacity can be installed at fixed cost on each arc, the problem is called Capacitated Fixed Charge Network Design (see e.g. Gendron and Crainic [61], Holmberg and Yuan [73], Crainic et al. [45]). Other variants include multiple facility extensions where several facilities having different capacities and costs are available (Agarwal [1], Dahl and Stoer [48], Bienstock and Günlük [23]), and survivability requirements such as multiple node-disjoint paths for each commodity.

In the context of the design of communication and transportation systems, the main objective is often to provide connectivity between a set of points (i.e., communication or transportation links), and other than fixed costs there are no additional costs for routing messages or goods within the system capacity. In this case routing costs are zero and the problem is purely strategic. In this context, Magnanti et al. [91] and Magnanti et al. [89] define a subproblem of the CNDP, called Network Loading Problem (NLP), that models the design of capacitated networks where routing costs are zero, the network graph is undirected, and facilities of fixed capacity can be installed in integer multiples on edges. In general, different facility types $t_1, t_2, \ldots, t_r$ are available, each having a different base capacity $u^{t_1} < u^{t_2} < \cdots < u^{t_r}$. Base capacities are modular, that is, u^{t_i} is an integer multiple of $u^{t_{i-1}}$, $i = 2, \ldots, r$, and the smallest base capacity u^{t_1} is equal to 1. Magnanti et al. [89] study a two facility NLP motivated by a private communica-

tion network leasing problem. In this case, facilities correspond to Digital Signal Level (DS) channels that can be leased by a company to provide different transmission rates between central offices.

van Hoesel et al. [117] and van Hoesel et al. [118] distinguish between undirected and bidirected NLP models with respect to capacity usage. In the first case edge capacity has to be shared between commodities routed on the two corresponding arcs, i.e., the total capacity installed on each edge $\{i, j\}$ gives an upper bound on the sum of the flows on both directions (i, j) and (j, i). Conversely, in bidirected models each capacity unit can be used simultaneously in both directions, that is, installing a capacity unit on edge $\{i, j\}$ permits a simultaneous flow on both arcs (i, j) and (j, i). Bienstock and Günlük [23] study a bidirected NLP motivated by a real problem arising as a part of a more complex problem concerning the design of ATM optical networks. These problems are bidirected because telecommunication traffic has traditionally been bidirected and telecommunication networks have been designed in such a way that each physical link permits the same traffic flow in both directions. Moreover, even when networks are specifically designed to handle undirected traffic, they are still built to permit bidirected flows in the event that such traffic will have to be carried in the future. In telecommunications problems an initial capacity can be available on arcs to represent a preexistent network that is to be expanded in order to meet an increased demand. In this case the problem is generally known as Capacity Expansion Problem [see 23, 68].

An important distinction is between bifurcated and non-bifurcated models (also called splittable and unsplittable models). The non-bifurcated NLP models a wide variety of practical situations such as the design of voice networks, telecommunication networks running ATM protocols, production-distribution with single sourcing, and express package delivery. In non-bifurcated problems commodities are restricted to be routed through a single path from the origin node to the destination node. It is not hard to impose non-bifurcation in the NLP model as it suffices to restrict the flow variables associated with each arc-

commodity pair to be binary (0-1) variables. However, the resulting problem becomes much harder to solve in practice because the corresponding LP-relaxation provides a very weak lower bound. Despite their practical significance, non-bifurcated problems have received much less attention in the literature than bifurcated problems.

A non-bifurcated problem, similar to the NLP, is studied in Gavish and Altinkemer [59] for the design of a computer backbone network where the objective is to simultaneously assign capacity to the arcs and route each commodity through a single path. A limited number of capacity options is available on each arc and there are fixed costs for installing capacity on arcs, variable costs that are function of the flow on arcs, and delay costs taking into account the queueing of flows due to capacity congestion on arcs. Brockmüller et al. [27] and Brockmüller et al. [28] study a generalization of the non-bifurcated NLP motivated by a problem arising in telecommunication industry. The objective is to re-design a company's private line network in order to minimize the total lease costs for the lines, while permitting that a set of commodities are routed through a single path. Different capacity sizes are available and the cost structure for installing capacity on edges is non-linear. Moreover, only a subset of nodes, called hub nodes, can be used to route commodities. Non-bifurcated problems are studied also in Barahona [17], Berger et al. [22], Atamtürk and Rajan [6], van Hoesel et al. [117], van Hoesel et al. [118].

Non-bifurcated NLPs also find important applications in transportation planning. Here facilities correspond to trucks of fixed size that must be assigned to routes to define a load plan, whereas commodities correspond to freights loaded on trucks. In this context Bossert and Magnanti [25] use a non-bifurcated NLP to model a logistic problem, called Pup Matching, that consists in matching semi-trailers called pups to cabs that can handle up to two pups simultaneously. Pup Matching is defined on a transportation network corresponding to a directed graph, where each arc is associated with a fixed cost that represents the expense for sending a cab trough it. The problem is to send each pup from an origin node

to a destination node minimizing cab fixed costs. Since each cab can handle two pups, assigning a cab to an arc provides the option of routing through it up to two pups without additional costs. Then, each pup corresponds to an unsplittable commodity, generating a single unit of flow, whereas each cab corresponds to a facility that can be installed on arcs to provide two capacity units. The problem of minimizing cab costs while assigning an appropriate route to all pups can be modeled as a non-bifurcated NLP. Indeed, the non-bifurcated NLP represents the core structure of many problems in freight transportation. These problems can often be modeled with reasonable approximation as network loading problems under some simplifying assumptions.

1.3. Routing Problems

Routing problems can be viewed as the operational level in the management of distribution and transportation networks. Decisions related to the number and location of facilities (hubs, plants, warehouses or depots), size of the fleet, customer-depot allocation, and allocation of transportation services between locations concern the design of the network and may be viewed as strategic and tactical. On the other hand, the daily problem of routing vehicles to deliver goods from local depots to customers can be classified as operational. This distinction does not only depend on the nature of the decisions involved, but is clearly connected with the time-span and frequency of the decisions. In fact, structural changes in the network are considered by the management once every few years and once implemented cannot be changed without incurring in major capital investments.

The basic and probably most well studied vehicle routing problem is the Capacitated Vehicle Routing Problem (CVRP). In the CVRP a set of customers, each with an associated requirement of some product, must be supplied from a single depot by a homogeneous fleet of vehicles of known capacity. The problem is to design a set of simple circuits for the vehicles, called routes, starting and ending

at the depot, and such that each customer is supplied by exactly one route. The total load of each route (i.e., the sum of the loads required by the customers visited) must not exceed the vehicle capacity, and the objective is to minimize the sum of route costs. The CVRP is $\mathcal{NP}$-hard as it contains the well known Traveling Salesman Problem (TSP) as a special case. In fact, the TSP corresponds to a CVRP where a single vehicle of capacity greater or equal to the sum of all customers demands is available at the depot (the depot corresponding to the initial city). The CVRP is one of the most studied combinatorial optimization problems and since it was first proposed by Dantzing and Ramser [49] it has been studied in many variants.

In fact, routing problems arising in real applications usually involve a number of additional complications which may in practice make the problem much harder to solve. Typical complications include:

a) the presence of time windows associated with each customer, i.e., an earliest and latest time at which the customer can be visited;
b) the presence of pickup and delivery transportation requests, i.e., customers requiring that the associated load is picked up at a specified location and delivered to a corresponding delivery location;
c) the presence of multiple depots;
d) a planning horizon of several days in which each customer requires to be visited a given number of times according to specific day combinations.

Time constrained routing problems have grown as an important research area in the last two decades as a consequence of the increasing importance of the time dimension for manufacturing and transportation companies engaged in an effort to compete on service quality. In many applications such as bank and postal deliveries, industrial refuse collection or school bus routing, customer service must occur according to strict time constraints called time windows. Time windows correspond to given time intervals, associated with each customer, imposing that a customer can only be visited at a time within its time window. The Vehicle

Routing Problem with Time Windows (VRPTW) is a generalization of the CVRP that models this kind of problems. In the VRPTW a travel time is associated with each arc (representing the time consumed for traveling between the corresponding endpoints) and with each customer are associated both a time window and a service time. Usually, vehicles are allowed to arrive at a customer location before the earliest time imposed by the corresponding time window, but in this case the vehicle must wait until the the customer earliest time before starting to service it. The objective of the VRPTW is to design a set of routes of minimum cost to service all customers within the imposed time windows without exceeding the vehicle capacity. In some applications involving flexible time schedules it is permitted to violate time window constraints at a cost, usually a linear function of the amount of time window violation. In this case time windows are called soft (see Ferland and Fortin [55], Balakrishnan [11]).

A more general model that has received less attention in the literature is the Pickup and Delivery Problem with Time Windows (PDPTW). In the PDPTW the set of vertices is partitioned into two subsets of pickup and delivery vertices. Each pickup has an associated delivery and requires that a quantity of some product is loaded at the pickup location and delivered to the corresponding delivery location by the same vehicle. Each pickup-delivery pair defines a transportation request and, as for the VRPTW, a time window is associated with each vertex. Capacity constraints impose that the total load of the vehicle after visiting each customer cannot exceed the maximum vehicle capacity. Notice that, since the amount loaded at each pickup must be unloaded at the corresponding delivery before returning to the depot, when time windows are wide enough all customers could in principle be visited using a single vehicle.

The PDPTW contains as special cases both the CVRP and the VRPTW. When the set of delivery vertices is empty and the delivery vertex associated with each pickup vertex corresponds to the depot, the PDPTW reduces to the VRPTW. In addition, when all time windows are arbitrarily large, the PDPTW becomes the classical CVRP.

A wide class of different pickup and delivery problems have been studied in the literature motivated by different real applications (see Cordeau et al. [40], Parragh et al. [101] and Berbeglia et al. [21] for recent surveys). With respect to how vertices are paired to define the transportation requests one can distinguish between one-to-one (1-1), one-to-many-to-one (1-M-1), and many-to-many (M-M) problems. The PDPTW model described above belongs to the 1-1 class because each pickup is paired with a single delivery; whereas in 1-M-1 problems all delivery vertices must be supplied from the depot and all pickup demands must be delivered to the depot. Among 1-M-1 problems a further distinction can also be made between single demands and combined demands (see Gribkovskaia and Laporte [67]). In the first case each vertex must be either a pickup or a delivery (but not both). In the latter, a vertex can be both a pickup and a delivery, meaning that it may have both a delivery demand, that must be supplied to it from the depot, and a pickup demand, that must be brought to the depot. A well known problem in this class is the VRP with backhauls, where pickup and delivery vertices, here called linehauls and backhauls, have the depot as corresponding delivery and pickup vertex, respectively. Moreover, each route must visit all delivery vertices before starting visiting pickups, and both the total load delivered and the total load picked up by the route must not exceed the vehicle capacity.

In M-M problems pickups and deliveries are unpaired, that is, each unit of product loaded at a pickup vertex can be unloaded at any delivery customer in the same route to satisfy one unit of the associated demand. This variant is a relaxation of the PDPTW described above and models situations where there is a single product that must be shifted between vertices. When a single vehicle is available, this problem is called One-Commodity Pickup and Delivery Traveling Salesman Problem (1-PDTSP) or Traveling Salesman Problem with Pickup and Delivery (Anily and Bramel [3], Hernandez-Perez and Salazar-Gonzalez [71]).

An important extension of the PDPTW that has received considerable attention in the literature is the Dial-A-Ride-Problem (DARP) (see e.g. Cordeau [36],

Cordeau and Laporte [37]). The DARP is concerned with the transportation of people (elderly and disabled in most applications) that require to be picked up at a specified origin location within a specified time window and moved to a corresponding destination by a specified latest time. The DARP is modeled as the PDPTW except that an additional constraint is imposed to ensure service quality, i.e., the time between a person pickup and the corresponding delivery cannot exceed a maximum ride time.

An attempt to generalize the various PDP variants in unified notation can be found in Savelsbergh and Sol [113]. They propose a unified model, called General Pickup and Delivery Problem (GPDP), to model the different pickup and delivery variants including the DARP. The model can handle various complicating constraints found in many practical applications such as multiple pickups associated with a single delivery or vehicles with different start and end locations.

When the time horizon extends to several days, VRP problems assume a more strategic meaning. Companies offering periodic services to the customers, such as grocery distribution and refuse collection, often design a delivery plan over a given planning horizon that is then operated unchanged for several months. The Period Vehicle Routing Problem (PVRP) is the problem of designing, for an homogeneous fleet of vehicles located at a central depot, a set of routes for each day of a given p-day period servicing customers with known demands. Each customer requires to be visited a fixed number of times, called its frequency, and requires the same quantity of product every time he is visited. The visits of each customer can only occur according to a given number of allowable day-combinations. For example, a customer may require to be visited twice during a 5-day period imposing that these visits should take place on Monday-Thursday or Monday-Friday or Tuesday-Friday. Each vehicle can perform at most one route per day, each route must start and finish at the depot, and the vehicle can service a total customer demand that is smaller than or equal to its capacity. The PVRP consists in simultaneously assigning a day-combination to each customer and in designing the vehicle routes for each day of the planning period so that each customer

is visited the required number of times, the number of routes on each day does not exceed the number of vehicles available on that day, and the total cost of the routes over the p-day period is minimized.

The PVRP model described above is quite general and contains as special cases both the CVRP and the Multi-Depot VRP (MDVRP) that is an extension of the CVRP where a customer can be served from p depots and inter-depot routes are not allowed. In fact, the PVRP becomes the single depot CVRP when the planning period is of one day only (i.e., $p = 1$) and every customer must be visited exactly once. The MDVRP can be obtained as a special case of the PVRP where each day corresponds to a depot (i.e. p represents the number of depots), each customer has frequency 1, and each customer can be visited on any of the p days.

Besides strategic problems, a class of tactical VRPs called Tactical Planning VRPs (TPVRP) that involve the assignment of delivery days to customers can be modeled as PVRPs. Examples of such problems arise in application sectors such as beverage and food industries and companies operating in the service sector. Here, customer orders are placed over a p-day horizon and with each order is associated a day-window defined by an initial and a final day of the planning horizon. Each customer would like to be serviced on the initial day of his day window but it is possible to service him on a later day within his day window by paying a service cost that is function of the delay from the initial day. A limited vehicle fleet does not allow the company to service all customers on the initial day of their day window. Therefore, a dispatcher must choose every day a subset of customers to be visited from a portfolio of orders placed over the p-day horizon. This is a typical scenario for business organizations operating in the service sector, such as utility companies in energy, telecommunications, or water distribution that routinely face the problem of scheduling their commercial and technical personnel over a limited planning period to visit customers. Customer requests can result from a planned maintenance schedule defined by the company and directly from the clients through a call center. Depending on the urgency of each request a maximum delay is allowed with respect to the required

visit day, and the effect of postponing a visit can be accounted for by means of a service cost that must be minimized in order to meet reasonable quality of service standards. Bostel et al. [26] refer to this kind of problems as "field force planning and routing" and describe an application for optimizing the service logistic activities of a company operating in the area of water treatment and distribution. Such problems can be viewed as PVRPs in that a possible solution method consists in solving a PVRP that involves both the assignment of a delivery day to each order of the portfolio and the design of the routes for the p days. The objective is to service all orders within day p minimizing the sum of routing costs and customer service costs.

1.4. Research motives and goals

This thesis is based on the development of new exact and heuristic algorithms for three $\mathcal{NP}$-hard combinatorial optimization problems, belonging to the classes of Network Design and Vehicle Routing problems, that arise in the context of transportation planning. Specifically, we consider a generalization of the NLP with single facility where routing costs are non-zero, called Non-Bifurcated Capacitated Network design problem (NBP), and two important generalizations of the Capacitated Vehicle Routing Problem: the Period Vehicle Routing Problem (PVRP) and the Pickup and Delivery Problem with Time Windows (PDPTW).

In contrast to bifurcated network design problems, the NBP has received much less attention in the literature for what concerns both heuristic and exact algorithms. Exact solution algorithms for the NBP are particularly scarce, and it is often hard to compare the results achieved by different methods as each method uses a different set of test instances. Our first aim is to get a more clear picture for what concerns the limits of the current exact algorithms and their usefulness in solving NBPs of practical interest. We find that an exact method that is based on solving a strengthened formulation of the NBP using the commercial integer programming solver CPLEX [see 43], is competitive with a state of the art branch-and-cut algorithm from the literature. However, practical problems are still far from the reach of the current state of the art exact methods.

As far as new heuristic methods are concerned, we will concentrate our attention on different heuristic techniques to compare their effectiveness in solving the NBP. Our main goal in this direction is to design heuristic algorithms that can be used to effectively solve problems involving complete networks of moderate size. To this end, we investigate the structure of the optimal solutions of the problem to derive a necessary condition for optimality that can be enforced within a constructive heuristic. We also experiment different metaheuristic techniques, namely partial enumeration, adaptive memory and tabu search, with the aim to perform a wider exploration of the solution space while trying to avoid

getting trapped in local minima.

Concerning the VRPs we focus our attention on exact methods. We use a Set Partitioning (SP) like formulation to model these problems aiming at generalizing to the PVRP and to the PDPTW an effective solution method recently proposed for the CVRP in Baldacci et al. [16]. In the SP model of the CVRP each column of the SP matrix represents a feasible route servicing a subset of customers and an additional constraint limits the total number of routes in any feasible solution. The same formulation can be used to model the PDPTW as it suffices to remove from the SP model all routes that are not feasible for the PDPTW. The PVRP requires instead a more involved formulation in order to impose that customers are visited on each day of one corresponding day combination. However, the proposed PVRP formulation can still be relaxed to a SP-like formulation.

In both cases the resulting SP problem cannot be solved directly since the number of variables is exponential, but it can be used to compute a lower bound on the problem without generating the entire SP matrix. The method that we aim to investigate combines in an additive manner a number of dual ascent procedures that explore different relaxations of the problem in order to compute a near optimal dual solution of the LPrelaxation of the SP model. The advantage of using different bounding procedures is that by executing them in sequence from the weakest, but computationally faster, to the strongest, but computationally heavier, it is possible to achieve a significant speed up of the overall algorithm. In fact, the dual solution computed by any bounding procedure can be used as a starting point for the next procedure reducing its required number of iterations.

Some of these procedures require column generation so that the identification of a suitable pricing subproblem for generating routes, and the development of an effective algorithm for solving it, will constitute an important area of investigation. In fact, the effectiveness of any column generation method is heavily biased toward the efficiency of the pricing algorithm. In generating routes for the VRPs considered in this thesis several complications arise with respect to the CVRP.

For the PDPTW the pricing problem must also account for time window, pairing, and precedence constraints. Since the pricing problem is solved by dynamic programming, the introduction of the time dimension can potentially result in a dramatic increase in the number of states that must be generated. Thus, effective techniques that reduce the state space by exploiting the structure of the problem are likely to be crucial to the efficiency of the pricing algorithm.

Regarding the PVRP, since its mathematical formulation is significantly different from that of the CVRP due to the constraints involving day-combinations, it is not straightforward to extend to the PVRP the bounding procedures proposed for the CVRP that are based on a pure SP model. Yet, in the case of the PVRP it is crucial to start the column generation phase with a dual solution of reasonable quality because, at each iteration, the pricing subproblem must be solved for each day of the planning period.

An important role in computing a tight lower bound is played by valid inequalities. In this thesis we aim at generalizing to the PVRP those inequalities that proved most effective for the CVRP, namely Capacity Constraints and Clique Inequalities. Moreover, inspired by the recent success reported by Jepsen et al. [76] in solving the VRPTW, we investigate the effectiveness of a limited subset of the Subset-Row (SR) inequalities, introduced in such paper, that happen also to be a subset of the clique inequalities. SR inequalities have the nice property of being defined on the rows of the SP matrix instead of the columns. Therefore, within a column generation schema they can grant several advantages from a computational viewpoint with respect to clique inequalities.

Once computed a lower bound, and knowing an upper bound on the problem, it is possible to derive a reduced integer problem where the SP matrix contains only a limited number of columns, but that is still guaranteed to contain an optimal solution to the original problem. If this reduced problem is small enough it can be solved very efficiently using a general purpose integer programming solver.

The effectiveness of this approach greatly depends on the quality of both the

lower and the upper bounds available because the number of columns in the reduced problem depends on the gap between these two values. Moreover, this method can fail in solving highly degenerate problems having a large number of equivalent solutions. To overcome this drawback, we propose to experiment a hybrid method that first tries to derive the reduced problem, and then resorts to a branch-and-cut-and-price algorithm if the reduced problem is too large.

Ultimately, it is our hope to present new exact methods that are competitive with, if not better than, other exact methods proposed in the literature for the same problems. Indeed, at our knowledge no exact algorithms nor lower bounds have been proposed so far for the PVRP. We aim at demonstrating that PVRP instances from the literature, for which only heuristics are available, are within the reach of the proposed exact algorithm within a reasonable computing time.

1.5. Thesis plan

The remainder of this thesis is organized in three main chapters, each dealing with a different problem, followed by a chapter containing concluding remarks.

In Chapter 2 we study the Non-Bifurcated Capacitated Network Design Problem and we experiment both exact and heuristic approaches. We describe an exact algorithm that solves an integer formulation of the problem strengthened by simple valid inequalities by means of an integer programming solver. Computational experiments on a set of instances from the literature show that the exact algorithm is competitive with a recent branch-and-cut algorithm from the literature. However, both algorithms are time consuming and fail to optimally solve several instances of limited size.

With the aim to attack problems of more realistic size we propose four new heuristic algorithms. We describe a two phase heuristic method, called TPH, based on a necessary condition for optimality and two partial enumerative heuristics called PEM and F&B. Algorithm PEM performs a partial enumeration of the

solution space and uses TPH to derive an upper bound from each partial solution. Algorithm F&B is a more general metaheuristic than PEM as it does not rely on TPH to compute upper bounds, and can be adapted to solve a wider class of combinatorial problems exhibiting a common substructure. Moreover, we describe an iterative procedure, called RTS, that uses at each iteration a tabu search method to improve a starting solution. The new algorithms are compared with the exact method on two classes of instances. The first class contains the same instances from the literature that are used to compare the new exact method and the branch-and-cut algorithm from the literature. The second class contains new instances corresponding to complete graphs with 30 nodes that are randomly generated to simulate practical problems.

Routing problems are the subject of chapters 3 and 4 where we describe new exact algorithms for solving the Period Routing Problem and the Pickup and Delivery Problem with Time Windows.

In chapter 3 we study the Period Vehicle Routing (PVRP). We describe a set partitioning-like formulation of the problem and five bounding procedures that are based on three different relaxations of the mathematical formulation. The first relaxation corresponds to an integer problem where the planning period is of one day only and the number of vehicles available is equal to the sum of vehicles over all days of the planning period. In this relaxation each route can be performed more than once and each customer must be visited a number of times exactly equal to its frequency. The second relaxation corresponds to the LP-relaxation of the integer formulation of the problem, whereas the third relaxation is obtained by adding to the LP-relaxation a generalization to the PVRP of the capacity constraints and the clique inequalities. Different bounding procedures using both elementary and non elementary routes are used to solve each relaxation and to find near optimal solutions to the dual of the LP-relaxation of the problem.

Then, we describe an exact algorithm that attempts to derive a reduced integer problem containing only the variables having a reduced cost smaller than the difference between a known upper bound and the final lower bound obtained by

the bounding procedures. The optimal PVRP solution is obtained by solving the reduced integer problem using a general purpose integer programming solver.

We test the proposed algorithm on a large set of PVRP instances from the literature comparing the upper and lower bounds achieved against the best known solutions from the literature. We also test the performance of the algorithm on a set of randomly generated Tactical Planning VRP (TPVRP) instances and we report a comparison among the best solutions found by the exact algorithm with and without service costs. This latter tests are aimed at studying how much taking into account service costs can impact on the quality of the solutions with respect to the customer service level. In particular, we are interested in evaluating the effectiveness of the TPVRP as a tool for optimizing the tradeoff between customer satisfaction and the associated increase in routing costs.

In chapter 4 we study the pickup and delivery problem with time windows (PDPTW). We distinguish between two variants of the PDPTW with respect to the objective function that is to be minimized, called PDPTW-o1 and PDPTW-o2. The PDPTW-o1 asks to minimize the sum of the route costs, whereas the PDPTW-o2 involves a fixed cost W associated with each vehicle and asks to minimize the sum of fixed costs and route costs. The PDPTW-o2 is often used to model problems where the primary objective is to minimize the number of vehicles used.

We formulate the PDPTW-o1 as a set partitioning-like problem and we propose an exact method based on two different relaxations of the mathematical formulation and a branch-and-cut-and-price algorithm. The first relaxation corresponds to an integer problem that is obtained by relaxing in a Lagrangian fashion the set partitioning constraints. Since this problem involves an exponential number of variables, it is further relaxed by allowing routes to be non feasible. The route set is therefore replaced with a set containing relaxations of feasible routes, called t-routes, that are only guaranteed to satisfy time window constraints and can be computed in pseudo-polynomial time. The second relaxation corresponds to the LP-relaxation of the original problem and is solved by adapting to the PDPTW a Lagrangian dual ascent method proposed by Baldacci et al. [16] for the

CVRP. Finally, a third relaxation is obtained by strengthening the LP-relaxation using a subset of the Subset-Row inequalities [see 76]. This latter relaxation is solved by a simplex-based column and cut generation method.

The last two relaxations are based on column generation and in solving the pricing problem require a dynamic programming (DP) procedure, called GENR, that permits to generate feasible PDPTW routes. In procedure GENR, DP states correspond to paths starting from the depot that are expanded to obtain feasible routes. Procedure GENR differs from other DP algorithms proposed in the literature for solving the pricing subproblem of the PDPTW. A main difference is that GENR computes for each path a lower bound on the reduced cost of any route containing it. This allows GENR to dramatically reduce the number of states that must be generated by fathoming all states having a lower bound greater than a known upper bound on the cost of any route. This lower bound is computed by means of two different state-space relaxations [see 32] that relax in different ways the feasibility of the routes.

We describe an exact algorithm for the PDPTW-o1 that uses the final lower bound achieved by the bounding procedures to generate a reduced problem containing only the routes whose reduced costs are smaller than the gap between a known upper bound and the lower bound achieved. If the resulting problem has moderate size it is solved by an integer programming solver, otherwise it uses a branch-and-cut-and-price algorithm to close the integrality gap.

A standard method for solving the PDPTW-o2 is to modify the travel cost matrix by adding the fixed cost W to each outgoing arc from the depot and then solve the resulting problem as a PDPTW-o1. We propose a different strategy that uses the bounding procedures developed for the PDPTW-o1 and is based on solving a number of PDPTW-o1 problems, each involving a different number of vehicles.

The new algorithms are tested on two classes of problems from the literature and compared with a recent branch-and-cut-and-price algorithm from the literature in order to assess their effectiveness.

Chapter 2

Non-Bifurcated Network Design

The Capacitated Network Design Problem (CNDP) represents a general model for a wide range of applications in planning the construction, development and improvement of transportation, logistics, telecommunication and production systems as well as in many other major areas [see 10, 88, 92]. Given a network, a set of origin-destination node pairs (*commodities*) and demand data for the commodities, the capacitated network design problem is to install integer multiples of some base capacity unit on the links of the network and route the flow of commodities so that the sum of flows on each link does not exceed the installed capacity and the sum of capacity fixed costs and flow routing costs is minimized. The CNDP becomes the Network Loading Problem with single facility (NLP) [see 91, 17, 117] when the same base capacity unit is available on each link and flow routing costs are zero. This problem has been studied in many variants with respect to network layout, capacity usage and commodity routing options.

When the flow of each commodity must run through a single path along the network the problem is called *non-bifurcated* (or unsplittable), whereas if the flow is allowed to be split among several paths the problem is called *bifurcated* (or splittable). Non-bifurcated network design problems arise in the design of telecommunication networks running ATM protocols, production-distribution with single sourcing and express package delivery (see Barnhart et al. [19] and Gavish and Altinkemer [59]). With respect to capacity usage the problem is called *di-*

rected (or bidirected) if the capacity installed on each link can be used twice, once in each direction, i.e., if a capacity unit is installed on a link then this capacity can be used in one or both directions. Conversely, the problem is called *undirected* if capacity has to be shared by the commodities routed in each direction. Other variants include the availability of multiple facilities (i.e., different base capacity units, each having different size and cost), integer non-binary flows (i.e., flows can be splitted in integer parts) and non-linear capacity installation costs. In this chapter we consider the directed non-bifurcated CNDP, called NBP in the following. The NBP is $\mathcal{NP}$-hard as MINIMUM COVER polynomially transforms to the recognition version of NBP (van Hoesel et al. [117]).

In this chapter we describe an exact method as well as four new heuristics for the NBP. All algorithms have been tested on both a set of instances from the literature and a new set of bigger NBP instances. The remainder of this chapter is organized as follows. In Section 2.1 we give an overview of the solution methods presented in the literature for the NBP and for some of its variants sharing the same basic structure. We summarize recent developments focusing on the distinction between bifurcated and non-bifurcated models. In Section 2.2 we describe the main notation used in this chapter and we give an integer programming formulation of the NBP. In Section 2.3 we describe the exact methods presented in the literature for the NBP and we introduce an exact method that solves a strengthened formulation of the NBP using a commercial integer programming solver. Sections 2.4 – 2.7 describe new heuristics for the NBP. In Section 2.4 we present a two phase heuristic method based on a necessary condition for optimality. In Sections 2.5 and 2.6 we propose two partial enumerative heuristics called PEM and F&B. We describe how heuristic F&B can be used to solve a broad class of combinatorial problems exhibiting a common substructure and we show an application of F&B for solving the NBP. In Section 2.7 we describe an iterative procedure that uses, at each iteration, a tabu search method to improve a starting NBP solution. In Section 2.8 we present computational experiments on two sets of problems to test the effectiveness of the proposed heuristics and the per-

formance of the exact method when compared with a branch-and-cut algorithm from the literature. Concluding remarks follow in Section 2.9.

2.1. State of the art

In this Section we summarize recent results presented in the literature for different variants of the NBP. Comprehensive surveys covering the earlier literature up to 1989 can be found in Magnanti and Wong [90] and Minoux [92]. This section is divided in two parts, the first dealing with bifurcated network design problems and the second with non-bifurcated problems. Bifurcated problems are much more studied than non-bifurcated problems, possibly because in practice they are significantly easier to solve. Unfortunately, most often it is not possible to extend the results obtained for bifurcated problems to the corresponding non-bifurcated ones.

2.1.1 Bifurcated network design

A well studied variant of the bifurcated CNDP is the Fixed Charge Network Design Problem (FCNDP) that arises in the special case where at most one base capacity can be installed on each arc.

For the FCNDP Gendron and Crainic [61] describe three formulations to derive different Lagrangian-based relaxations that are strengthened using knapsack inequalities. They report computational results comparing different bounding procedures based on a formulation, called strong formulation, that is obtained by adding a-priori valid inequalities stating that a commodity cannot use an arc unless the fraction of capacity needed is installed on it. They report that the best results are achieved when relaxing capacity constraints, however, the end gaps between the best upper and lower bounds are very large.

Crainic et al. [46] compare two different Lagrangian relaxations obtained by dualizing capacity constraints and flow conservation constraints, respectively.

They present a detailed analysis of the resulting relaxations and compare subgradient and bundle methods for solving the associated Lagrangian duals on a large set of test problems corresponding to graphs with up to 30 vertices, 700 arcs and 400 commodities. Computational experiments show that bundle methods converge faster and are more robust with respect to different relaxations, problem characteristics, and parameter settings. Although the best lower bounds (on average within 9% of optimality) are obtained by solving the strong formulation using a commercial integer programming solver, the proposed bounding procedures achieve for large scale problems very close lower bounds in a fraction of the computing time.

Crainic et al. [45] propose a tabu search meta-heuristic based on a path formulation that explores the space of the path-flow variables using pivot-like moves and column generation. Ghamlouche et al. [62] develop a new class of cycle-based neighborhood structures that improves the searching strategy of Crainic et al. and test such neighborhoods within a tabu-based local search. To define cycle-based neighborhoods the idea is to identify two points in the network and two paths between them that form a cycle. Then, the algorithm tries to deviate flow from one path to the other so that a different solution is obtained where previously open arcs have no flow and can be dropped. The method was further improved in Ghamlouche et al. [63] by combining the cycle-based tabu search described above with a path relinking framework [see 64]. The algorithm was tested on two sets of instances also used in Ghamlouche et al. [62] comprising FCNDP instances with up to 100 nodes 700 arcs and 400 commodities. The comparison with the cycle-based tabu search and the integer programming solver CPLEX [43] shows that the new method, on average, outperforms the cycle-based tabu search in terms of solution quality. The algorithm shows an average gap of 2.32% and 3.08% from the best solutions found by CPLEX on the two sets of instances but achieves better solutions than CPLEX on 3 instances.

Holmberg and Yuan [73] propose a branch-and-bound algorithm based on a Lagrangian heuristic. The Lagrangian relaxation is obtained by relaxing the flow

conservation constraints so that the resulting Lagrangian subproblem is decomposable into a problem for each arc of the network. The subproblems satisfy the integrality property and thus can be solved by the greedy principle. After solving the Lagrangian subproblems a feasible solutions is obtained by solving a multicommodity flow problem over a network defined by the subproblems solutions. The proposed Lagrangian schema is embedded into a heuristic branch-and-bound algorithm that uses heuristic variable fixing and different dominance rules. The algorithm is compared with the integer programming solver CPLEX on a large set of problems with up to 150 nodes 1,000 arcs and 282 commodities, and obtains better solutions or shorter computing times in 52 problems out of 65.

Agarwal [1] presents a heuristic algorithm for solving a multiple facility CNDP where different base capacities are available on each arc. Here, each unit of flow between two nodes is considered as a commodity, therefore the problem can be viewed as bifurcated with the restriction that flows can be splitted in integer parts. The basic approach can be thought as a neighborhood search technique based upon local improvement. Starting from an initial feasible solution the algorithm selects at each step a base link $\{a, b\}$ and defines an associated subnetwork. The subnetwork contains the base link itself and all pairs of links $\{a, i\}$ and $\{i, b\}$ for each node i in the network. A corresponding subproblem is then solved to reroute the flow within the subnetwork so as to minimize the total cost of the subnetwork. The subproblem is reduced to a Multiple Choice Knapsack Problem which is solved using a dynamic programming approach. The algorithm is tested on networks having different size and topologies. For networks with up to 20 vertices the algorithm is tested by comparing its heuristic solution with the lower bound achieved by a branch-and-cut algorithm. On such instances the gap between the upper and lower bound is in most cases below 5%. A set of bigger networks with up to 99 nodes 401 arcs and 4 facilities is also considered. For these instances the paper reports the worst and best solution achieved over 3 runs of the algorithm but no evaluation of the solution quality is given.

Different classes of valid inequalities have been proposed in the literature for

bifurcated CNDPs.

Magnanti et al. [91] introduce the NLP and study two of its subproblems to derive valid inequalities. The first subproblem is a knapsack-type problem, called *single arc design problem*, and is defined by the capacity constraint associated with a single edge. The second subproblem, called *three node network problem*, is obtained by considering a 3-node network with an edge between each node pair.

A single arc design problem for each edge arises when relaxing in a Lagrangian fashion the flow conservation constraints. Therefore, the optimal value of the Lagrangian dual of such relaxation provides an upper bound to the value of the LP-relaxation that can be achieved by adding all inequalities derived for single arc design problems. Studying this subproblem the authors introduce a new set of valid inequalities called *residual capacity inequalities*. They prove that adding residual capacity inequalities together with upper bound constraints (stating that the flow of a commodity on each edge cannot exceed the commodity demand) to the subproblem provides a complete description of the associated convex hull of feasible solutions. Studying the three node network problem they introduce the families of *cut set inequalities* and *three-partition inequalities*, expressing lower bounds on the total capacity that must be installed on two and three-sets partitions of the network. These last two classes of inequalities, together with non negativity constraints, completely describe the convex hull of feasible solutions of the three node network problem.

In Magnanti et al. [89] the above inequalities are generalized to the Two Facility NLP, where two facilities are available, the smallest one having unitary base capacity. Computational results over networks with up to 15 nodes show that the average integrality gap is around 8%. The authors report that cut-set inequalities alone are more effective than residual capacity inequalities alone in reducing the integrality gap. Moreover, they show that even adding a-priori a limited subset of cut-set inequalities significantly reduces the integrality gap at the root node.

Recently, Atamtürk and Rajan [6] described a linear time algorithm for separating residual capacity inequalities and present a cutting plane algorithm for

the bifurcated CNDP where these inequalities are separated exactly. Other valid inequalities, related to the Mixed-Integer Rounding inequalities, were introduced in Bienstock and Günlük [23] and Günlük [68] studying different variants of the bifurcated NLP.

Metric inequalities are a class of inequalities introduced by Onaga and Kakusho [99] and Iri [75] to characterize the feasibility of (bifurcated) multicommodity flows that generalize the min-cut max-flow duality to Capacitated Multicommodity Network Flow Problems (CMNFP). Metric inequalities can be viewed as Benders cuts associated with extreme rays of the dual polyhedron of the LP-relaxation of CMNFPs [see 42]. Therefore, these inequalities can be used to characterize the set of capacity vectors that can accommodate a feasible multicommodity flow through the network. Metric inequalities give rise to an alternative formulation for the bifurcated NLP, called *capacity formulation*, that uses capacity variables only but requires an exponential number of constraints. Metric inequalities can be strengthened in different ways, e.g. by rounding arguments giving rise to the class of *rounded metric inequalities* [see 24] that contains as a special case the cut-set inequalities.

Bienstock et al. [24] consider a variant of the bifurcated NLP where the graph is directed and base capacities of unitary size can be installed independently on each arc. The authors describe two branch-and-cut algorithms based on two different formulations enforced by valid inequalities. The first formulation is a standard multicommodity flow formulation based on cut-set inequalities, flow-cutset inequalities and three-partition inequalities. The second formulation is a capacity formulation that is based on rounded metric inequalities. This formulation uses two other classes of valid inequalities called *partition inequalities* and *total capacity inequalities*. Partition inequalities are derived as a special case of rounded metric inequalities and are used to bound the flow that must traverse a partition of the graph. Total capacity inequalities are obtained by applying the Chvátal-Gomory procedure to partition inequalities. The two formulations are compared on two sets of instances with up to 27 nodes, 102 arcs and 702 commodities. The

results obtained show that the multicommodity formulation with the additional cuts grants, on average, better results.

The capacity formulation of the bifurcated NLP is also studied in Dahl and Stoer [48] and Avella et al. [8]. Dahl and Stoer [48] study a generalization of the NLP with additional survivability requirements and model it using the capacity formulation. They propose a cutting-plane algorithm where violated metric inequalities are generated by solving a LP by column generation and introduce a new class of inequalities called *band inequalities* that exploit the Knapsack substructure of the problem. Avella et al. [8] introduce the new class of *tight metric inequalities* that completely characterize the convex hull of the integer feasible solutions of the problem. They present a branch-and-cut algorithm that looks for violated tight metric inequalities in two steps. First it looks for violated metric inequalities and then attempts to tighten them to derive tight metric inequalities. This algorithm is able to improve some of the results reported in Bienstock et al. [24].

2.1.2 Non-bifurcated network design

Gavish and Altinkemer [59] were among the first to study a non-bifurcated CNDP. They consider a multiple facility problem where in addition to fixed costs and flow routing costs each link of the network is associated with a delay cost that is proportional to the capacity utilization on it. The proposed algorithm is based on a path formulation of the problem where paths are dynamically generated. Capacity constraints are dualized in a Lagrangian fashion and cut-set inequalities are added to the model a-priori to strengthen the Lagrangian relaxation. A Lagrangian heuristic is run at each iteration to obtain feasible solutions. Computational results over a set of different network topologies with up to 32 nodes show that the gap between upper and lower bounds can be as high as 20%, although most often it is less than 10%. A computational analysis of the impact of fixed costs on solution quality shows that the percentage difference between lower and upper bound significantly increases when fixed costs are associated with base capacities.

Heuristics for the non-bifurcated undirected NLP are proposed in Barahona [17] and Berger et al. [22]. Barahona [17] proposes a relaxation which is based on cut-set inequalities and Spanning Tree inequalities. Cut-set inequalities are also used in Barahona [17] to formulate a relaxation of the NLP that involves only an integer variable for each edge but requires an exponential number of constraints. In this formulation, the existence of a multicommodity flow is partially enforced by adding violated cut-set inequalities while connectivity is enforced by adding Spanning Tree inequalities. The algorithm proposed by Barahona works as follows.

First subsets of vertices are aggregated into "supervertices" in order to reduce the problem size to obtain a so called backbone network. Then, the bifurcated relaxation of the problem is solved for the aggregated network using a branch-and-cut method based on cut-set inequalities and Spanning Tree inequalities. The separation problem for cut-set inequalities is formulated as a max-cut problem. Once a bifurcated solution is found for the backbone network, the algorithm ex-

pands this solution to obtain a bifurcated solution for the original network and successively converts it to a non-bifurcated solution using a heuristic procedure. The algorithm is tested on a set of instances corresponding to complete undirected graphs with up to 64 vertices, representing real world telecommunication networks. A 6% increase in the solution cost between the bifurcated and corresponding non-bifurcated solution is reported in the worst case. Although solutions to problems with up to 64 vertices have been obtained, the aggregated networks never contains more than 15 vertices.

Berger et al. [22] present a tabu search method for solving a multiple facility NLP. Solution neighborhoods are based on the k-shortest paths between the origin and the destination nodes of each commodity computed with respect to the flows of the remaining commodities. They report computational results over a set of randomly generated instances with up to 200 vertices and edge density up to 0.2, and show that the proposed algorithm outperforms 1-opt and 2-opt greedy neighborhood search heuristics (Raghavan [104]) which are based on a similar neighborhood.

Brockmüller et al. [27, 28] study a generalization of the NLP motivated by a real problem in telecommunications where only a subset of nodes can be used to route commodities and capacity installation costs are non linear. They define a hierarchy of relaxations to reduce the large number of 0-1 variables needed to linearize costs and introduce new classes of valid inequalities, called c$-$*strong inequalities*, that are derived by studying the projection of the feasible region onto the space of variables related to a single edge. The new inequalities, together with cut-set inequalities, are embedded in a cutting-plane procedure that uses a preprocessed formulation obtained by variable reduction and by adding a-priori special cases of the above inequalities. The algorithm uses a heuristic procedure that, given a LP solution, derives an integer solution by applying branch-and-bound on a restricted problem obtained by imposing that the value of each capacity variable cannot differ more than a constant from its LP value. When the algorithm is unable to find violated inequalities it stops returning a tightened

formulation that is solved by branch-and-bound.

van Hoesel et al. [117] study both the directed and undirected variants of the NLP with single facility. They introduce three classes of valid inequalities, both including the class of c−strong inequalities, and report the results obtained by a branch-and-cut algorithm based on the new inequalities on a set of real-life NLP instances provided by KPN Research in Liendschendam, The Netherlands. In addition to the new inequalities the algorithm also incorporates cut-set inequalities, three-partition inequalities[see 23] and the general *k*-cuts [see 18]. The instances are defined on complete graphs ranging between 4 and 8 nodes and fully dense non-symmetric demand matrices. This branch-and-cut algorithm is able to optimally solve instances with up to 7 nodes.

Atamtürk and Rajan [6] consider both the non-bifurcated and bifurcated versions of the CNDP. They formally prove that the separation problem of c−strong inequalities is $\mathcal{NP}$-hard and introduce two new classes of valid inequalities, both of which include the c−strong inequalities as a special case. The effectiveness of the new inequalities is then tested on a set of NBP instances using a branch-and-cut algorithm.

2.2. Mathematical formulation of the NBP

Let $G = (V, E)$ be an undirected and connected graph where V is the node set and E is the edge set. We denote by (i_e, j_e) the endpoints of edge $e \in E$. Let $G' = (V, A)$ be the directed graph associated with G, where A is the set of arcs obtained from E replacing every edge $e \in E$ with two arcs in opposite directions, i.e. $A = \{(i_e, j_e), (j_e, i_e) : e \in E\}$. The mapping $e(i, j)$ gives the edge of E corresponding to arc $(i, j) \in A$. A set R of p commodities is given. Each commodity $k \in R$ specifies a demand $d_k \in \mathbb{Z}^+$ that must be sent through a single path from a origin node $s_k \in V$ to a destination node $t_k \in V$, with $s_k \neq t_k$. Let w_{ij}^k be the cost for routing commodity k through arc $(i, j) \in A$. On each edge $e \in E$ can be installed integer multiples of a base capacity $u_e \in \mathbb{Z}^+$, each at fixed cost c_e. The total

capacity installed on edge e represents an upper bound for the flow on edge e in each direction, so that, the required capacity on each edge e is determined by the maximum flow on the corresponding arcs (i_e, j_e) and (j_e, i_e).

The NBP is to assign a single path to each commodity and to install on each edge sufficient capacity to meet the demands of the commodities so that the sum of capacity installation and flow routing costs is minimized. Let x_e be a non-negative integer variable representing the number of base capacity units u_e installed on edge $e \in E$, and let f_{ij}^k be a $(0-1)$ binary variable that is equal to 1 if and only if commodity k is sent through arc $(i,j) \in A$. The NBP can be formulated as the following integer program.

$$(\text{NBP}) \quad z_{\text{NBP}} = \min \sum_{e \in E} c_e x_e + \sum_{(i,j) \in A} \sum_{k \in R} w_{ij}^k f_{ij}^k \tag{2.1}$$

$$s.t. \sum_{j \in \Gamma_i} f_{ji}^k - \sum_{j \in \Gamma_i} f_{ij}^k = \begin{cases} 1 \text{ if } i = t_k \\ -1 \text{ if } i = s_k \\ 0 \text{ otherwise} \end{cases} \quad \begin{matrix} \forall i \in V \\ \forall k \in R \end{matrix} \tag{2.2}$$

$$\sum_{k \in R} d_k f_{i_e j_e}^k \leqslant u_e x_e, \qquad \forall e \in E \tag{2.3}$$

$$\sum_{k \in R} d_k f_{j_e i_e}^k \leqslant u_e x_e, \qquad \forall e \in E \tag{2.4}$$

$$x_e \in \mathbb{Z}^+, \qquad \forall e \in E, \tag{2.5}$$

$$f_{ij}^k \in \{0,1\}, \qquad \forall (i,j) \in A, \forall k \in R \tag{2.6}$$

where Γ_i denotes the set of nodes adjacent to node $i \in V$. Constraints (2.2) specify the usual flow conservation for each commodity at each node of the graph, while constraints (2.3) and (2.4) ensure that total capacity installed on edge e can meet the total flow on arcs (i_e, j_e) and (j_e, i_e), respectively. Notice that by relaxing the $0-1$ restrictions (2.6) on flow variables we obtain the formulation of the bifurcated version of the problem, while the undirected problem can be obtained by substituting constraints (2.3) and (2.4) with a single capacity constraint summing up all flows in direction (i_e, j_e) and (j_e, i_e).

2.3. Exact methods

A mayor difficulty in solving the integer program NBP comes from the fixed costs associated with each edge capacity and the large number of flow variables and flow balance constraints. Indeed, just solving the corresponding LP-relaxation takes considerable computing time and memory and the value of the optimal LP-solution provides a very weak lower bound. Therefore, in designing exact algorithms for CNDPs most of the attention has been devoted to strengthening the LP relaxation with valid inequalities.

In contrast to bifurcated problems very few exacts algorithms have been proposed for non-bifurcated CNDPs. At our knowledge the only exacts methods for the NBP are due to van Hoesel et al. [117, 118] and Atamtürk and Rajan [6]. These algorithms and the inequalities they are based on are briefly described below.

Several classes of valid inequalities for the NBP can be derived by partitioning the node set and imposing lower bounds on the total capacity that must be installed between the node subsets induced by this partition. The cut-set inequalities are a class of valid inequalities which are known to significantly improve the LP-relaxation of the CNDP see [see 91, 89, 23, 17] and express a necessary (but not sufficient) condition for the existence of a multicommodity flow in a graph. Roughly speaking, they state that for any cut in G the sum of capacities installed on edges traversing it must be at least equal to the total demand of the commodities that must traverse the cut, rounded up to the nearest integer. Let $\delta(S)$ be the cut induced in G by a subset $S \subseteq V$ and let $\gamma^+(S)$ and $\gamma^-(S)$ be the set of commodities having origin node in S and destination node in $V \setminus S$ and the set of commodities having origin node in $V \setminus S$ and destination node in S, respectively. In the special case where the base capacity is the same for all edges (i.e., $u_e = u$ $\forall e \in E$) the cut-set inequalities can be strengthened to:

$$\sum_{e \in \delta(S)} x_e \geqslant \pi(S), \quad \forall S \subseteq V, \tag{2.7}$$

where

$$\pi(S) = \max\left\{\left\lceil \sum_{k\in\gamma^+(S)} d_k/u \right\rceil, \left\lceil \sum_{k\in\gamma^-(S)} d_k/u \right\rceil\right\}.$$

Similar inequalities that generalize the cut-set inequalities are the three-partition inequalities that are obtained by partitioning the node set in three subsets [see 89, 23, 68].

Different classes of valid inequalities have been derived for the NBP studying a related subproblem, that is a generalization of the well known $0-1$ Knapsack problem, defined by the capacity constraint associated with a single edge [see 6, 117, 27]. Consider a relaxation of the NBP defined by constraints (2.3), (2.5), (2.6) for a fixed edge $e \in E$. For notational convenience let $x = x_e$ and $f^k = f^k_{i_e j_e}$, $\forall k \in K$, be the variables involved by this relaxation. Dividing equations (2.3) by u_e one obtains the following integer set.

$$\mathcal{F} = \left\{(\mathbf{f}, x) \in \{0,1\}^p \times \mathbb{Z} : \sum_{k\in R} a_k f^k \leqslant x\right\}, \tag{2.8}$$

where $a_k = \frac{d_k}{u_e}$ represents the fraction of the base capacity u_e required by commodity $k \in R$ to run through edge e. In the following we denote by $conv(\mathcal{F})$ the convex hull of the integer set $\mathcal{F}$. Notice that since $\mathcal{F}$ is a relaxation of NBP any valid inequality for $\mathcal{F}$ is also valid for NBP.

Brockmüller et al. [27] proved that all non-trivial facet defining inequalities for $conv(\mathcal{F})$ take the form $\beta x \geqslant \boldsymbol{\pi}\mathbf{f} - \pi_0$, $\beta, \pi_0 \in Z_0^+$, $\pi_k \in Z_0^+$, $\forall k \in R$. Moreover, it has been proved that when looking for strong valid inequalities for $\mathcal{F}$ it is sufficient to study the set $\mathcal{F}'$ that is obtained from $\mathcal{F}$ by substituting a_k with its fractional part $r_k = a_k - \lfloor a_k \rfloor$, $\forall k \in R$ [see 6, 117]. Strong inequalities for $\mathcal{F}$ can then be obtained by strong inequalities for $\mathcal{F}'$ [see 117]. Therefore, in the following we assume $0 \leqslant a_k \leqslant 1$, $\forall k \in R$.

Brockmüller et al. [27] introduced the c*−strong inequalities* for a generalization of $\mathcal{F}$ and characterize necessary and sufficient conditions under which the inequalities are facet defining. These inequalities can be defined as follows. Let $S \subseteq R$ be a subset of commodities and let $X(S) = \left\lceil \sum_{k\in S} a_k \right\rceil$. The set S is called

c−strong if $c = \sum_{k\in S} \lceil a_k \rceil - X(S) = |S| - X(S)$, since we assumed $0 < a_k < 1$. If $S \subseteq R$ is c−strong the following c−strong inequality is valid for $\mathcal{F}$:

$$x \geqslant \sum_{k\in S} f^k - c. \tag{2.9}$$

A c−strong set $S \subseteq R$ is called *maximal* c−*strong* if $S \setminus \{i\}$ is c−strong and $S \cup \{i\}$ is not c−strong $\forall i \in R \setminus S$. Inequality (2.9) is facet defining for $conv(\mathcal{F})$ if and only if the set S is maximal c−strong.

By taking $S = \{k\}$, $k \in R$ one obtains as a special case of the c−strong inequalities the following inequalities, imposing that no commodity can run through any arc (i, j) if no capacity is installed on the associated edge $e(i, j)$:

$$f^k_{ij} \leqslant x_{e(i,j)} \quad \forall (i, j) \in A, \forall k \in R. \tag{2.10}$$

Notice that when $\frac{d_k}{u_e} > 1$ inequalities (2.10) can be strengthened to

$$\left\lceil \frac{d_k}{u_e} \right\rceil f^k_{ij} \leqslant x_{e(i,j)} \quad \forall (i, j) \in A, \forall k \in R.$$

The separation of c−strong inequalities is $\mathcal{NP}$-hard because the separation problem reduces to a number of knapsack problems [see 6]. However, given a fractional solution $(\bar{\mathbf{f}}, \bar{x})$, when looking for violated c−strong inequalities it is possible to ignore all variables $\bar{f}^k$ having integral value because their coefficients in each violated inequality can be fixed a-priori.

Atamtürk and Rajan [6] describe a branch-and-cut algorithm based on the c−strong inequalities and two other classes of inequalities called κ−*split* c−*strong inequalities* and *lifted knapsack cover inequalities*. κ−split c−strong inequalities are a generalization of c−strong inequalities. For any positive integer κ and any $S \subseteq R$ let $c^\kappa_S = \sum_{k\in S} \lceil \kappa a_k \rceil - \left\lceil \sum_{k\in S} \kappa a_k \right\rceil$. The following κ−split c−strong inequality is valid for $\mathcal{F}$:

$$\sum_{k\in S} \lceil \kappa a_k \rceil f^k + \sum_{k\in R\setminus S} \lfloor \kappa a_k \rfloor f^k \leqslant c^\kappa_S + \kappa x. \tag{2.11}$$

Notice that each inequality (2.11) can be viewed as a c−strong inequality for a relaxation of $\mathcal{F}$ where the capacity variable x is allowed to take values that are integer multiples of $\frac{1}{\kappa}$ (c−strong inequalities are in fact obtained by taking $\kappa = 1$).

Lifted knapsack cover inequalities are derived by considering the knapsack set $\mathcal{F}(\nu, N_0, N_1)$ that is obtained from $\mathcal{F}$ by projecting x to $\nu \in \mathbb{N}$, all flow variables indexed by a set $N_0 \subseteq R$ to 0 and all flow variables indexed by a set $N_1 \subseteq (R \setminus N_0)$ to 1. Then, the set $C \equiv R \setminus (N_0 \cup N_1)$ is called a cover if $\sum_{k \in C \cup N_1} a_k > \nu$ and it is a *minimal cover* if $C \setminus \{i\}$ is not a cover $\forall i \in C$. Atamtürk and Rajan use sequential lifting to derive lifted knapsack cover inequalities. The procedure starts from a minimal knapsack cover C and a corresponding cover inequality for $\mathcal{F}(\nu, N_0, N_1)$ and then lifts it by sequentially introducing first the capacity variable x, and second all the projected variables in $N_0 \cup N_1$. Given a minimal cover C, a corresponding lifted minimal cover inequality takes the following form:

$$\sum_{k \in C} f^k + \sum_{k \in N_0} \alpha_k f^k + \sum_{k \in N_1} \alpha_k (1 - f^k) + \alpha(\nu - x) \leqslant |C| - 1, \tag{2.12}$$

where α and α_k, $k \in R \setminus C$, are lifting coefficients that are sequentially computed by solving an appropriate lifting problem. The authors also show that, given a minimal cover, a lifted inequality (2.12) can be obtained in time $O(p^3)$. Similar inequalities have also been derived in van Hoesel et al. [117].

The branch-and-cut of Atamtürk and Rajan uses an initial formulation that is strengthened by adding a-priori a subset of the cut-set inequalities defined for one and two-node subsets of the network. c−strong inequalities are separated by means of a greedy heuristic, whereas κ−split c−strong inequalities are separated similarly to c−strong inequalities using values of κ up to 4. Knapsack cover inequalities are separated using a greedy heuristic by setting $\nu = \lceil \bar{x} \rceil$, where $\bar{x}$ is the value of the capacity variable in the current LP-solution, and then lifted by sequential lifting. A comparison between the improvement of the integrality gap at the root node after adding the different classes of valid inequalities shows that on most instances κ−split c−strong inequalities provide the best improvements. However, the improvement with respect to c−strong inequalities is rather limited. The branch-and-cut algorithm is tested over a set of NBP instances with up to 29 nodes and 61 arcs using CPLEX as the integer programming solver. The results show that adding κ−split c−strong inequalities and lifted cover inequalities

in addition to c−strong inequalities has in general a positive effect. However, the impact of adding the new inequalities in addition to c−strong inequalities is not much significant.

van Hoesel et al. [117] describe a branch-and-cut algorithm for the NBP that is based on cut-set inequalities, three-partition inequalities, c−strong inequalities and the new classes of *lower convex envelope inequalities* and *two-side inequalities*. Lower convex envelope inequalities contain the c−strong inequalities as a special case and are obtained by studying a projection $\mathcal{F}(R_0, R_1)$ of the set $\mathcal{F}$ where all variables indexed by $R_0 \subseteq R$ are set equal to 0 and all variables indexed by $R_1 \subseteq R$ are set equal to 1. Letting $S = R \backslash (R_0 \cup R_1)$ be the set of free variables, different lover convex envelope inequalities can be derived for $\mathcal{F}(R_0, R_1)$ having the following general form:

$$\beta x \geqslant \eta \sum_{k \in S} f^k - \rho, \tag{2.13}$$

for an appropriate choice of coefficients $\beta, \eta, \rho \in \mathbb{Z}_0^+$. van Hoesel et al. [117] provide two choices of the coefficients β, η and ρ that, under some conditions, give rise to facet defining inequalities for $conv(\mathcal{F}(R_0, R_1))$. Notice that, since inequalities (2.13) are defined for a projection of $\mathcal{F}$, they have to be lifted to $\mathcal{F}$. van Hoesel et al. prove that once given an ordering of the commodities in $R_0 \cup R_1$ sequential lifting of inequalities (2.13) can be done in polynomial time.

Two side inequalities are derived for the NBP by studying a relaxation, similar to $\mathcal{F}$, that is defined by all solutions satisfying both constraints (2.3) and (2.4) for a fixed edge $e \in E$, together with integrality constraints (2.5), (2.6). Let $\hat{k} \in R$ and let $\alpha \in \mathbb{Z}_0^+$ such that $1 \leqslant \alpha \leqslant \lceil a_{\hat{k}} \rceil$. Two side inequalities are as follows:

$$x_e \geqslant \alpha f^{\hat{k}}_{i_e j_e} + \sum_{k \in R} (\lceil a_k \rceil - \alpha) f^k_{j_e i_e}. \tag{2.14}$$

The proposed branch-and-cut algorithm is tested on a set of instances defined by complete graphs with up to 8 nodes and fully dense non-symmetric demand matrices. Since the size of the graphs is small, the separation of cut-set and three-partition inequalities is performed by complete enumeration. c−strong inequalities are separated for values of $c = 1, 2, 3$ by means of a greedy heuristic, whereas

lower convex envelope inequalities are separated by enumeration after defining the sets R_0 and R_1. Violated two side inequalities for an edge are found by first fixing α and then finding the commodity $\hat{k}$ having the greatest value $f^{\hat{k}}_{i_e j_e}$. To attest the usefulness of the different inequalities the authors compare the bound at the root node after adding only cut-set and three-partition inequalities or only lower convex envelope and c−strong inequalities. The results show that cut-set and three-partition inequalities tend to provide tighter bounds when the commodity demands are smaller. Conversely, the percentage of gap closed by lower convex envelope and c−strong inequalities increases when commodity demands get bigger. Using all the inequalities the algorithm is able to solve instances with up to 7 nodes, but it fails in solving instances with as few as 8 nodes.

We attempted to solve the same NBP instances considered by Atamtürk and Rajan [6] using the integer programming solver CPLEX (see CPLEX [43]) for an improved formulation, called SNBP, that is obtained by adding a-priori to NBP the following two types of valid inequalities.

a) The cut-set inequalities (2.7) defined for single-node subsets, that is, defined for all sets S such that $|S| = 1$.

b) A subset of inequalities (2.10) defined as follows. We compute for each commodity $k \in R$ the shortest path $P(k)$ from origin s_k to destination t_k in the directed graph G' using arc costs $c_{e(ij)} + w^k_{ij}$, $\forall (i,j) \in A$. Then, we add the following subset of inequalities (2.10):

$$f^k_{ij} \leqslant x_{e(ij)} \quad \forall (i,j) \in P(k), \forall k \in R. \tag{2.15}$$

Our computational results (see Section 2.8.3) show that CPLEX using formulation SNBP (in the following called CPLEX(SF)) is competitive with the branch-and-cut of Atamtürk and Rajan [6]. However, the size of NBP instances that can be solved by both exact methods is much smaller than real life problems and, moreover, both methods are time consuming. Therefore, in the following sections we propose four heuristic methods for the NBP.

2.4. Two phase heuristic TPH

In this section we derive a condition that is satisfied by any optimal NBP solution, then we use this condition to design a heuristic procedure for improving a feasible NBP solution.

2.4.1 A necessary condition for optimality

Consider a NBP partial solution $(\bar{x}, \bar{f})$ of cost $\bar{z}$ for a given subset $\bar{R} \in R$ and let $\bar{q}_{ij}$ be the total flow of commodities in $\bar{R}$ on arc $(i,j) \in A$, that is:

$$\bar{q}_{ij} = \sum_{k \in \bar{R}} d_k \bar{f}^k_{ij}, \quad \forall (i,j) \in A. \tag{2.16}$$

We have $\bar{f}^k_{ij} = 0$, $\forall (i,j) \in A$, $\forall k \in R \setminus \bar{R}$, and $\bar{x}_e = \max[\lceil \frac{\bar{q}_{i_e j_e}}{u_e} \rceil, \lceil \frac{\bar{q}_{j_e i_e}}{u_e} \rceil]$, $\forall e \in E$.

Let $\bar{\delta}_{ij}(\Phi)$ be the additional capacity installation cost for augmenting, in solution $(\bar{x}, \bar{f})$, the flow on arc $(i,j) \in A$ from the current value $\bar{q}_{ij}$ to $\bar{q}_{ij} + \Phi$. It is quite obvious that $\bar{\delta}_{ij}(\Phi) = 0$ if the directed *residual* capacity $(u_{e(i,j)} \bar{x}_{e(i,j)} - \bar{q}_{ij})$ on arc (i,j) is greater than or equal to Φ, while $\bar{\delta}_{ij}(\Phi) > 0$ if the residual capacity is smaller than Φ. Specifically, the values $\bar{\delta}_{ij}(\Phi)$, $\forall (i,j) \in A$, are computed as follows:

$$\bar{\delta}_{ij}(\Phi) = \max \left[c_{e(i,j)} \left(\left\lceil \frac{\bar{q}_{ij} + \Phi}{u_{e(i,j)}} \right\rceil - \bar{x}_{e(i,j)} \right), 0 \right], \quad \forall (i,j) \in A. \tag{2.17}$$

For a given commodity $r \in R \setminus \bar{R}$, consider the partial solution $(\tilde{x}, \tilde{f})$ of cost $\tilde{z}$ that is obtained from $(\bar{x}, \bar{f})$ by routing commodity r through some simple path $P(r)$ from node s_r to node t_r. Solution $(\tilde{x}, \tilde{f})$ is obtained from $(\bar{x}, \bar{f})$ and $P(r)$ by setting:

$$\begin{aligned} &\tilde{f}^k_{ij} = \bar{f}^k_{ij}, \quad \forall (i,j) \in A, \quad \forall k \in \bar{R}, \\ &\tilde{f}^r_{ij} = 1, \quad \forall (i,j) \in P(r), \\ &\tilde{f}^r_{ij} = 0, \quad \forall (i,j) \notin P(r). \end{aligned} \tag{2.18}$$

The values $\tilde{x}_e, \forall e \in E$, are given by:

$$\tilde{x}_e = \begin{cases} \max\left[\left\lceil \frac{\bar{q}_{i_e j_e} + d_r}{u_e} \right\rceil, \bar{x}_e\right], & \text{if} \quad (i_e, j_e) \in P(r), \\ \max\left[\left\lceil \frac{\bar{q}_{j_e i_e} + d_r}{u_e} \right\rceil, \bar{x}_e\right], & \text{if} \quad (j_e, i_e) \in P(r), \\ \bar{x}_e, & \text{otherwise.} \end{cases} \tag{2.19}$$

Associate with each arc $(i,j) \in A$ the *incremental* cost $\bar{c}_{ij}^{\,r}$ for routing commodity $r \in R \setminus \bar{R}$ through arc (i,j) in solution $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$. Incremental cost $\bar{c}_{ij}^{\,r}$ is the sum of the additional installation cost $\bar{\delta}_{ij}(d_r)$ for augmenting the capacity of edge $e(i,j)$ to meet the flow $\bar{q}_{ij} + d_r$ and of the flow routing cost w_{ij}^r, that is:

$$\bar{c}_{ij}^{\,r} = \bar{\delta}_{ij}(d_r) + w_{ij}^r \quad \forall (i,j) \in A, \ \ \forall\, r \in R \setminus \bar{R}. \tag{2.20}$$

The following lemma shows the relation between the costs $\bar{\mathbf{z}}$ and $\tilde{\mathbf{z}}$ of the two partial solutions $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$ and $(\tilde{\mathbf{x}}, \tilde{\mathbf{f}})$.

Lemma 1 *The following relation holds:*

$$\tilde{\mathbf{z}} = \bar{\mathbf{z}} + \sum_{(i,j) \in P(r)} \bar{c}_{ij}^{\,r}. \tag{2.21}$$

Proof: Let $E(r)$ be the edge subset covered by path $P(r)$ (i.e. $E(r) = \{e(i,j) \in E : (i,j) \in P(r)\}$) and $\tilde{R} = \bar{R} \cup \{r\}$. Using the definitions of $\bar{c}_{ij}^{\,r}$ and $\bar{\delta}_{ij}(d_r)$, the right hand side of expression (2.21) can be written as follows:

$$\begin{aligned} &\sum_{e \in E \setminus E(r)} c_e \bar{x}_e + \sum_{e \in E(r)} c_e \bar{x}_e + \sum_{k \in \bar{R}} \sum_{(i,j) \in A} w_{ij}^k \bar{f}_{ij}^k \\ &\quad + \sum_{(i,j) \in P(r)} \max\left[c_{e(i,j)} \left(\left\lceil \frac{\bar{q}_{ij} + d_r}{u_{e(i,j)}} \right\rceil - \bar{x}_{e(i,j)} \right), 0\right] + \sum_{(i,j) \in P(r)} w_{ij}^r \end{aligned}$$

$$\begin{aligned} = &\sum_{e \in E \setminus E(r)} c_e \bar{x}_e + \sum_{k \in \tilde{R}} \sum_{(i,j) \in A} w_{ij}^k \tilde{f}_{ij}^k \\ &+ \sum_{(i,j) \in P(r)} \left(\max\left[c_{e(i,j)} \left(\left\lceil \frac{\bar{q}_{ij} + d_r}{u_{e(i,j)}} \right\rceil - \bar{x}_{e(i,j)} \right) + c_{e(i,j)} \bar{x}_{e(i,j)}, c_{e(i,j)} \bar{x}_{e(i,j)} \right] \right) \end{aligned}$$

$$= \sum_{e \in E \setminus E(r)} c_e \bar{x}_e + \sum_{k \in \bar{R}} \sum_{(i,j) \in A} w_{ij}^k \tilde{f}_{ij}^k + \sum_{(i,j) \in P(r)} c_{e(i,j)} \max \left[\left\lceil \frac{\bar{q}_{ij} + d_r}{u_{e(i,j)}} \right\rceil, \bar{x}_{e(i,j)} \right]. \quad (2.22)$$

Using the definition of $\tilde{x}_{ij}$ given by equation (2.19), the expression (2.22) becomes:

$$\sum_{e \in E \setminus E(r)} c_e \tilde{x}_e + \sum_{e \in E(r)} c_e \tilde{x}_e + \sum_{k \in \bar{R}} \sum_{(i,j) \in A} w_{ij}^k \tilde{f}_{ij}^k = \tilde{\mathbf{z}}. \quad (2.23)$$

□

We now describe a condition that is satisfied by any optimal NBP solution.

Theorem 1 *(Necessary condition for optimality) Let* $(\mathbf{x}, \mathbf{f})$ *be an optimal NBP solution of cost* $\mathbf{z}$ *and let* $P(r)$ *be the path assigned to commodity* $r \in R$ *in solution* $(\mathbf{x}, \mathbf{f})$*. Let* $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$ *be the partial solution derived from* $(\mathbf{x}, \mathbf{f})$ *removing commodity* r *(i.e. setting* $\bar{f}_{ij}^r = 0$*,* $\forall (i,j) \in A$ *and* $\bar{f}_{ij}^k = f_{ij}^k$*,* $\forall (i,j) \in A$*,* $\forall k \in R \setminus \{r\}$*). The path* $P(r)$ *corresponds to the shortest path in graph* G' *from node* s_r *to node* t_r *when with each arc* $(i,j) \in A$ *is associated the incremental cost given by expressions* (2.20) *with respect to the partial solution* $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$*.*

Proof: By contradiction. Assume there is a path $P^*(r)$ in G' from from s_r to t_r such that:

$$\sum_{(i,j) \in P^*(r)} \bar{c}_{ij}^{\,r} < \sum_{(i,j) \in P(r)} \bar{c}_{ij}^{\,r}. \quad (2.24)$$

From Lemma 1 we have:

$$\mathbf{z} = \bar{\mathbf{z}} + \sum_{(i,j) \in P(r)} \bar{c}_{ij}^{\,r}. \quad (2.25)$$

Consider the NBP feasible solution $(\mathbf{x}^*, \mathbf{f}^*)$ obtained from $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$ by running commodity r through path $P^*(r)$. From Lemma 1 we have:

$$\mathbf{z}^* = \bar{\mathbf{z}} + \sum_{(i,j) \in P^*(r)} \bar{c}_{ij}^{\,r}. \quad (2.26)$$

Thus, from (2.25) and (2.26) we have:

$$\mathbf{z} - \mathbf{z}^* = \sum_{(i,j) \in P(r)} \bar{c}_{ij}^{\,r} - \sum_{(i,j) \in P^*(r)} \bar{c}_{ij}^{\,r}. \quad (2.27)$$

From expression (2.27) and inequality (2.24) we obtain $\mathbf{z} > \mathbf{z}^*$ which contradicts the assumption that solution $(\mathbf{x}, \mathbf{f})$ is optimal. □

Theorem 1 can be used iteratively to improve a feasible NBP solution as described in the following.

2.4.2 Heuristic TPH

In this section we describe a Two Phases Heuristic, called TPH, for solving the NBP that is based on Theorem 1. The first phase generates an initial feasible solution that is iteratively improved in the second phase until the resulting solution satisfies the necessary condition of optimality stated by Theorem 1.

The first phase of TPH starts from an empty solution $\bar{\mathbf{x}}$=0, $\bar{\mathbf{f}}$=0 and $\bar{R} = \emptyset$ and iteratively finds the shortest path $P^*(k)$ for a commodity $k \in R \setminus \bar{R}$ using the incremental costs $\bar{c}^k_{ij}$ defined by expressions (2.20) with respect to the emerging solution ($\bar{\mathbf{x}}$,$\bar{\mathbf{f}}$). At each iteration, the solution is updated by setting $\bar{f}^k_{ij} = 1$, $\forall (i,j) \in P^*(k)$. The quality of the NBP solution achieved at the end of the first phase strongly depends on the order in which the commodities are considered. In our computational experience the best results were obtained by numbering the commodities so that $shp_1 \geqslant shp_2 \geqslant \cdots \geqslant shp_p$, where shp_k denotes the cost of the shortest path in G' from s_k to t_k using arc costs $c_{e(i,j)} + w^k_{ij}$, $(i,j) \in A$.

The second phase of TPH iteratively removes a commodity k from the solution, computes a new shortest path $P^*(k)$ using the incremental costs $\bar{c}^k_{ij}$ with respect to the partial solution $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$ resulting from $(\mathbf{x}, \mathbf{f})$ removing commodity k, and inserts commodity k in solution $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$ along the path $P^*(k)$. To reduce the effect on the solution quality of the order in which commodities are considered, a sequence of $(2p-1)$ iterations are performed. The first p iterations consider the sequence of commodities $(1, 2, \ldots, p)$ while the last $(p-1)$ iterations consider the commodity sequence $(p-1, p-2, \ldots, 1)$. The second phase is repeated if during the previous execution some improvement has been achieved.

In the following, a step-by-step description of algorithm TPH is given.

Description of heuristic TPH

Phase 1: (Building an initial solution)

1. (Initialization) It is assumed that the commodities are numbered as described above. Set $\bar{\mathbf{x}} = 0$, $\bar{\mathbf{f}} = 0$ and $\bar{q}_{ij} = 0, \forall (i,j) \in A$. Initialize $\bar{\mathbf{z}} = 0$ and $k = 1$.

2. (Routing commodity k) Let $P^*(k)$ be the shortest path in G' from s_k to t_k using the incremental costs $\{\bar{c}^k_{ij}\}$ given by expressions (2.20) with respect to $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$.

3. (Update the partial solution $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$) For each arc $(i,j) \in P^*(k)$, set:
   $\bar{f}^k_{ij} = 1$, $\bar{q}_{ij} = \bar{q}_{ij} + d_k$ and $\bar{x}_{e(i,j)} = \max\left[\left\lceil \frac{\bar{q}_{ij}}{u_{e(i,j)}} \right\rceil, \left\lceil \frac{\bar{q}_{ji}}{u_{e(i,j)}} \right\rceil\right]$.
   Update $\bar{\mathbf{z}} = \bar{\mathbf{z}} + \sum_{(i,j)\in P^*(k)} \bar{c}^k_{ij}$.

4. Set $k = k + 1$. If $k = p$ then return to step 2, otherwise set $\mathbf{x} = \bar{\mathbf{x}}$, $\mathbf{f} = \bar{\mathbf{f}}$ and $\mathbf{z} = \bar{\mathbf{z}}$.

Phase 2: (Improving the solution $(\mathbf{x}, \mathbf{f})$ found by phase 1)
Define $L = (1, 2, \ldots, p, p-1, p-2, \ldots, 1)$.

5. Set $\mathbf{z}_{old} = \mathbf{z}$, $q_{ij} = \sum_{k\in R} d_k f^k_{ij}, \forall (i,j) \in A$ and initialize $r = 1$.

5. (Removing a commodity from solution $(\mathbf{x}, \mathbf{f})$) Let l_r be the index of the r-th commodity in the list L and let $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$ be the partial solution derived from $(\mathbf{x}, \mathbf{f})$ removing commodity l_r as follows:

$$\bar{x}_{e(i,j)} = \begin{cases} \max\left[\left\lceil \frac{q_{ij} - d_{l_r}}{u_{e(i,j)}} \right\rceil, \left\lceil \frac{q_{ji}}{u_{e(i,j)}} \right\rceil\right], & \text{if } f^{l_r}_{ij} = 1, \quad \forall (i,j) \in A \\ x_{e(i,j)}, & \text{if } f^{l_r}_{ij} = 0, \quad \forall (i,j) \in A \end{cases}$$

$$\bar{f}^k_{ij} = \begin{cases} f^k_{ij}, & \text{if } k \neq l_r, \quad \forall (i,j) \in A, \forall k \in R \\ 0, & \text{if } k = l_r, \quad \forall (i,j) \in A, \forall k \in R \end{cases}$$

   Let $\bar{\mathbf{z}}$ be the cost of solution $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$.

5. (Routing commodity l_r) Define $\bar{q}_{ij} = \sum_{k\in R} d_k \bar{f}^k_{ij}, \forall (i,j) \in A$. Let $P^*(l_r)$ be the shortest path in G' from s_{l_r} to t_{l_r} using the incremental arc costs $\{\bar{c}^k_{ij}\}$ defined by expression (2.20). Let $\mathbf{z} = \bar{\mathbf{z}} + \sum_{(i,j)\in P^*(l_r)} \bar{c}^{l_r}_{ij}$.

5. If $\mathbf{z} < \mathbf{z}_{old}$, then compute the new solution $(\mathbf{x}, \mathbf{f})$ as follows:
   $f^k_{ij} = \bar{f}^k_{ij}, \forall (i,j) \in A, \forall k \in R \setminus \{l_r\}$,
   $x_{e(i,j)} = \bar{x}_{e(i,j)}, \forall (i,j) \in A \setminus P^*(l_r)$,
   $q_{ij} = \bar{q}_{ij}, \forall (i,j) \in A \setminus P^*(l_r)$,
   $f^{l_r}_{ij} = 0, \forall (i,j) \in A \setminus P^*(l_r)$,
   $f^{l_r}_{ij} = 1$, $q_{ij} = \bar{q}_{ij} + d_{l_r}$ and $x_{e(i,j)} = \max\left[\left\lceil \frac{\bar{q}_{ij}+d_{l_r}}{u_{e(i,j)}} \right\rceil, \left\lceil \frac{\bar{q}_{ji}}{u_{e(i,j)}} \right\rceil\right], \forall (i,j) \in P^*(l_r)$.
   If $r < (2p-1)$, set $r = r+1$ and return to step 6, otherwise return to step 5.

5. If $\mathbf{z} = \mathbf{z}_{old}$ and $r < (2p-1)$ then set $r = r+1$ and return to step 6, otherwise Stop.

2.5. Partial enumeration heuristic PEM

In this section we describe a partial enumeration method, called PEM, that generates a number of feasible solutions of the NBP assuming that each commodity $k \in R$ can be routed using one path of an a-priori defined path set $\mathcal{P}_k$.

Let $\mathcal{P}_k$ be the set containing, for each commodity $k \in R$, the largest subset of the least cost shortest paths in G' from s_k to t_k using arc costs $\{c_{e(i,j)} + w^k_{e(i,j)}\}$ and satisfying the following conditions:

1. $|\mathcal{P}_k| \leqslant \chi$, where χ is an a-priori defined parameter;

2. the cost of the path of maximum cost in $\mathcal{P}_k$ is smaller than or equal to ρ times the cost of the least cost path, where $\rho \geqslant 1$ is an a-priori defined parameter.

Consider an enumerative tree where the nodes at level h represent a set $\mathcal{S}^h$ of partial solutions generated at level h involving commodities $1, 2, \ldots, h$. Each partial solution $S \in \mathcal{S}^h$ is represented by an ordered list of h paths, i.e., $S = (P^1_{j_1}, \ldots, P^k_{j_k}, \ldots, P^h_{j_h})$, where $P^k_{j_k}$ is the j_k-th element of the path set $\mathcal{P}_k$, $k = 1, \ldots, h$.

With each partial solution $S = (P^1_{j_1}, \ldots, P^k_{j_k}, \ldots, P^h_{j_h})$, at level h, we define $\mathbf{x}(S)$

and $\mathbf{f}(S)$ as follows:

$$f_{ij}^k(S) = \begin{cases} 1, & \text{if arc } (i,j) \in P_{j_k}^k, \quad \forall (i,j) \in A, k = 1, \ldots, h \\ 0, & \text{if arc } (i,j) \notin P_{j_k}^k, \quad \forall (i,j) \in A, k = 1, \ldots, h \\ 0, & \forall (i,j) \in A, k = h+1, \ldots, h \end{cases} \tag{2.28}$$

and

$$x_e(S) = \max \left[\left\lceil \frac{g_{i_e j_e}(S)}{u_e} \right\rceil, \left\lceil \frac{g_{j_e i_e}(S)}{u_e} \right\rceil \right], \quad \forall e \in E, \tag{2.29}$$

where:

$$g_{ij}(S) = \sum_{k=1}^{h} d_k f_{ij}^k(S), \quad \forall (i,j) \in A. \tag{2.30}$$

Then, the cost $\mathbf{z}(S)$ of partial solution S is given by:

$$\mathbf{z}(S) = \sum_{e \in E} c_e x_e(S) + \sum_{k \in R} \sum_{(i,j) \in A} w_{ij}^k f_{ij}^k(S). \tag{2.31}$$

Moreover, with each $S \in \mathcal{S}^h$ we associate a label $UB(S)$ which denotes the upper bound on the NBP solution that is achieved by heuristic TPH (see Section 2.4) where step 1 of phase 1 is modified as follows.

$1'$. (Initialization) Define $\bar{x}_e = x_e(S)$, $\forall e \in E$, $\bar{q}_{ij} = g_{ij}(S)$, $\forall (i,j) \in A$, and $\bar{f}_{ij}^k = f_{ij}^k(S)$, $\forall (i,j) \in A$, $k = 1, \ldots, h$. Set $k = h$ and $\bar{\mathbf{z}} = \mathbf{z}(S)$.

At level h the largest subset $U \subset \mathcal{S}^h$ such that $|U| \leqslant \Delta$ (where Δ is an a-priori defined positive integer) is selected and every $S \in U$ is expanded by appending to S all paths in $\mathcal{P}_{h+1}$, thus creating $|\mathcal{P}_{h+1}|$ elements of the set $\mathcal{S}^{h+1}$.

If $|\mathcal{S}^h| \leqslant \Delta$ then set $U = \mathcal{S}^h$, otherwise U contains the first Δ partial solutions in $\mathcal{S}^h$ for the lexicographical order of the pairs $(\sigma(S), UB(S))$, $S \in \mathcal{S}^h$, where $\sigma(S) = \sum_{e \in E} x_e(S)$, that is, every $S \in U$ satisfies $\sigma(S) < \sigma(S')$ or $\sigma(S) = \sigma(S')$ and $UB(S) \leqslant UB(S')$, for every $S' \in \mathcal{S}^h \setminus U$. A backtracking occurs if either all partial solutions $\mathcal{S}^h$ have been previously expanded or level p has been reached. In this case the algorithm goes back to the first stage h containing some unexpanded partial solution.

In expanding a partial solution $S \in \mathcal{S}^h$ we apply a heuristic dominance rule that in practice strongly reduces the size of $\mathcal{S}^{h+1}$. In expanding a partial solution S the dominance imposes that in case $g_{s_{h+1}t_{h+1}}(S) \geqslant d_{h+1}$, then from S it is generated only the partial solution of $\mathcal{S}^{h+1}$ obtained by appending to S the path $P(h+1) = (s_{h+1}, t_{h+1})$ of commodity $h+1$. This dominance rule becomes an exact rule to prevent alternative optimal solutions for the special version of the NBP where all commodity demands are equal and the routing costs do not depend on the commodities. More precisely.

Lemma 2.1 *Let* $(\mathbf{x}, \mathbf{f})$ *be an optimal solution of cost* $\mathbf{z}$ *of a NBP problem where all the commodities specify the same demand (i.e.* $d_k = d, \forall k \in R$*) and the routing costs do not depend on the commodities (i.e.* $w^k_{ij} = w_{ij}, \forall k \in R, \forall e \in E$*). Assume that in the optimal solution* $(\mathbf{x}, \mathbf{f})$ *the arc* (s_k, t_k) *is used by commodity* r *but is not used by commodity* k*, that is,* $f^k_{s_k t_k} = 0$ *and* $f^r_{s_k t_k} = 1$*. Then there exist an alternative optimal solution* $(\tilde{\mathbf{x}}, \tilde{\mathbf{f}})$ *such that* $\tilde{f}^k_{s_k t_k} = 1$*.*

Proof: Consider a new solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{f}})$ derived from $(\mathbf{x}, \mathbf{f})$ by routing commodity k on the arc (s_k, t_k) and by replacing, in the path assigned to commodity r, the arc (s_k, t_k) with the path assigned to commodity k from s_k to t_k (i.e. by setting $\tilde{f}^k_{s_k t_k} = 1$, $\tilde{f}^k_{ij} = 0, \forall (i,j) \in A \setminus \{(s_k, t_k)\}$, $\tilde{f}^r_{s_k t_k} = 0$, $\tilde{f}^r_{ij} = f^r_{ij}$, $(i,j) \in A \setminus \{(s_k, t_k)\}$ and $\tilde{f}^r_{ij} = f^k_{ij}, \forall (i,j) \in A$). The new solution $(\tilde{\mathbf{x}}, \tilde{\mathbf{f}})$ is feasible since both commodities r and k are routed from the corresponding source node to the destination node. Moreover, since $d_k = d_r = d$, we have $\sum_{k \in \overline{R}} d_k \tilde{f}^k_{ij} = \sum_{k \in \overline{R}} d_k f^k_{ij}$, $\forall (i,j) \in A$, and thus $\tilde{x}_e = x_e$, $\forall e \in E$. Finally, as the routing costs do not depend on the commodity, we have $\tilde{\mathbf{z}} = \mathbf{z}$. □

Description of heuristic PEM

In the description of algorithm PEM, we use the following two binary labels:

- $\beta(S)$ that is set equal to 1 if S has been expanded and equal to 0 otherwise;
- $\theta(h)$ that is set equal to 1 if level h does not contain any unexpanded partial solution and equal to 0 otherwise.

Moreover, we use a parameter NIT, whose value is a-priori defined, to limit the maximum number of iterations.

1. (Initialization) Generate the path sets $\mathcal{P}_k$, $k \in R$, as described above. In the following we assume that the commodities are numbered so that $|\mathcal{P}_1| \geqslant |\mathcal{P}_2| \geqslant \cdots \geqslant |\mathcal{P}_p|$. Define $S = \emptyset$, $\beta(S) = 0$. Set $\mathcal{S}^0 = \{S\}$ and $\mathcal{S}^k = \emptyset$, $k \in R$, $\mathbf{z}_{UB} = +\infty$, $h = 0$ and $iter = 1$.

2. (Expansion of the partial solution set $\mathcal{S}^h$) Let $\bar{U} = \{S \in \mathcal{S}^h : \beta(S) = 0\}$. We have three cases:

   (a) $|\bar{U}| > \Delta$: let U be the first Δ partial solutions in $\bar{U}$ for the lexicographical order of the pairs $(\sigma(S), UB(S))$, $S \in \bar{U}$. Set $\theta(h) = 0$.

   (b) $|\bar{U}| = \Delta$: set $U = \bar{U}$ and $\theta(h) = 1$.

   (c) $|\bar{U}| = 0$: set $\theta(h) = 1$ and go to 4.

   In case (a) and (b) repeat 3 for every $S \in U$.

3. (Expansion of $S \in U$) Set $\beta(S) = 1$. For each $P \in \mathcal{P}_{h+1}$ consider the partial solution S' obtained by appending to S the path P and let $\mathbf{z}(S')$ be the cost of S' given by expression (2.31). Compute $UB(S')$ using heuristic TPH as described above. If $UB(S') < \mathbf{z}_{UB}$, then update $\mathbf{z}_{UB} = UB(S')$.
   If S' is not dominated, then update $\mathcal{S}^{h+1} = \mathcal{S}^{h+1} \cup \{S'\}$ and set $\beta(S') = 0$.

4. Set $h = h + 1$. If $h < p$ then go to step 2.

5. Set $iter = iter + 1$.
   If $iter = NIT$ then STOP, otherwise let $h = \min_{1 \leqslant k \leqslant p} [k : \theta(k) = 0]$ and go to step 2.

Our computational experiments indicate that heuristic PEM is time consuming as it requires to run algorithm TPH to complete each partial solution. The computing time can be significantly reduced using only phase 1 of TPH but this deteriorates the quality of the solution achieved.

2.6. F&B: an adaptive memory-based algorithm for structured optimization problems

In this section we propose an iterative memory-based algorithm, called F&B, for solving a class of structured combinatorial optimization problems, and we show how the algorithm can be tailored to solve the NBP. Like algorithm PEM also F&B is based on partial enumeration, but it is much faster than PEM as it does not require to run heuristic TPH to complete each partial solution. In the following we first describe the general schema of algorithm F&B and then we present its application to the NBP.

2.6.1 Forward-Backward trees

Many combinatorial optimization problems exhibit a regular substructure that makes them decomposable into n smaller (and possibly easier) subproblems which are linked together by a set of coupling constraints. These problems can often be modeled by defining, for each subproblem k, a set $\mathcal{P}_k$ containing all the feasible solutions for subproblem k, and by reformulating the coupling constraints so that the resulting problem consists in choosing from each set $\mathcal{P}_k$, $k = 1, ..., n$, a single item $s^k_{i_k} \in \mathcal{P}_k$ in such a way that the selected items $S = \{s^1_{i_1}, \ldots, s^n_{i_n}\}$ satisfy all the constraints. With each item $s^k_i \in \mathcal{P}_k$ is associated a cost c^k_i and a weight a^k_i. The cost $\mathbf{z}(S)$ of solution S is a function of the selected items and the objective is to find a solution S of minimum cost. As an example, consider the Multiple Choice Knapsack problem (MCKP). In the MCKP are given n item sets $\mathcal{P}_k$, $k = 1, ..., n$ and a bin of size W. The objective is to select exactly one item from each set so that the sum of the item weights does not exceed W, and the sum of the item costs is minimized.

Algorithm F&B tries to avoid being trapped in local minima by adopting a memory-based look ahead strategy that exploits the knowledge gained in its past search history. F&B iterates a partial exploration of the solution space by gener-

ating a sequence of enumerative trees of two types, called *forward* and *backward* trees. Each node at level h of the trees represents a partial solution S' containing h items. At each iteration t, the algorithm generates a forward tree, if t is odd, or a backward tree if t is even. In generating a tree, each partial solution S' is extended to a feasible solution using the partial solutions generated at the previous iteration, and the cost of the resulting solution is used to "guess" the quality of the best complete solution that can be obtain from S'.

A forward tree is an n-level tree where each level $h = 1, ..., n$ is associated with the set $\mathcal{P}_h$ and each node at level h corresponds to a partial solution containing one item of each set $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_h$. Conversely, in a backward tree each level h is associated with the set $\mathcal{P}_{n-h+1}$, and a node at level h represents a partial solution containing one item of each set $\mathcal{P}_n, \mathcal{P}_{n-1}, \ldots, \mathcal{P}_{n-h+1}$. Associated with each level h of a tree built at iteration t there is a list, called $\mathcal{L}ist^t(h)$, containing Δ nodes generated at level h, where Δ is an a-priori defined parameter. Once the tree at iteration t has been completely expanded, the nodes in the lists $\mathcal{L}ist^t(h)$, $h = 1, ..., n$, represent the algorithm memory of past iterations $1, ..., t$ that will be used to guide the tree exploration in the following iteration $t + 1$. In order to make the exposition simpler, in the following no distinction is made between a node and the corresponding partial solution.

2.6.2 Evaluation of partial solutions

The key idea is to evaluate the completion cost of each partial solution generated at level h of the tree at iteration t using the partial solutions stored in $\mathcal{L}ist^{t-1}(n - h)$ at iteration $t - 1$. Suppose we are building the forward tree associated with an odd iteration t. Let $\mathcal{S}^h$ be the set of all partial solutions generated at level h, and consider two partial solutions $S \in \mathcal{S}^h$ and $\overline{S} \in \mathcal{L}ist^{t-1}(n - h)$. Notice that, since t is odd, S contains one item of the sets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_h$, while $\overline{S}$ contains one item of each set $\mathcal{P}_n, \mathcal{P}_{n-1}, \ldots, \mathcal{P}_{h+1}$. These two solutions can be combined to obtain a (not necessarily feasible) complete solution $S \cup \overline{S}$ of cost $\mathbf{z}\left(S \cup \overline{S}\right)$. Clearly, if the resulting solution $S \cup \overline{S}$ satisfies all the coupling constraints, then the associated

cost represents a valid upper bound on the problem. At each iteration t algorithm F&B builds the associated tree and computes for each node $S \in \mathcal{S}^h$ a label $\psi(S)$ as follows:

$$\psi(S) = \min_{\overline{S} \in \mathcal{L}ist^{t-1}(n-h)} \left\{ \mathbf{z}\left(S \cup \overline{S}\right) + \alpha\left(S \cup \overline{S}\right) \right\}, \tag{2.32}$$

where $\alpha(S \cup \overline{S})$ is a (strongly problem specific) function whose value is related to the degree of infeasibility of $S \cup \overline{S}$ and that is equal to 0 if $S \cup \overline{S}$ is a feasible solution. In this latter case $\psi(S)$ represents a valid upper bound on the problem.

When building the first forward tree at iteration $t = 1$ we assume that the lists $\mathcal{L}ist^0(h) = \emptyset$, $h = 1, \ldots, n$. Therefore, at iteration 1, expression (2.32) gives $\psi(S) = \mathbf{z}(S)$, where $\mathbf{z}(S)$ is the cost of the partial solution S.

2.6.3 Level expansion

Let Δ be an a-priori defined parameter that controls the number of nodes expanded at each level of both forward and backward trees. To expand level h of a tree at iteration t the algorithm computes the value $\psi(S)$ for each node $S \in \mathcal{S}^h$, and builds the set $\mathcal{L}ist^t(h) \subseteq \mathcal{S}^h$ containing the Δ nodes in $\mathcal{S}^h$ having the smallest label value $\psi(S)$. For every $S \in \mathcal{L}ist^t(h)$ such that $\psi(S)$ represents the cost of a feasible solution, we update $\mathbf{z}_{UB} = \min\{\mathbf{z}_{UB}, \psi(S)\}$, where $\mathbf{z}_{UB}$ represents the cost of the best solution achieved by F&B and is initialized equal to ∞ at the beginning of the algorithm. Each node S included in $\mathcal{L}ist^t(h)$ is expanded to create a new node $S \cup \{s\}$ for each item s of the set $\mathcal{P}_{h+1}$ associated with level $h + 1$. Any node $S \cup \{s\}$ that violates the problem constraints is rejected.

Algorithm F&B terminates after NIT iterations (where NIT is an a-priori defined parameter) or after two consecutive iterations where the value of $\mathbf{z}_{UB}$ does not improve.

Figure 2.1 shows an example of algorithm F&B at iteration $t + 1$ (even) expanding $\Delta = 2$ nodes per level. The label value for each node $S \in \mathcal{S}^4$ at level 4 of the backward tree associated with iteration $t + 1$ is computed using the partial solutions stored in $\mathcal{L}ist^t(6)$ of the forward tree computed at the previous iteration

t. The Δ nodes having the smallest label value are then included in $\mathcal{L}ist^{t+1}(4)$ and further expanded.

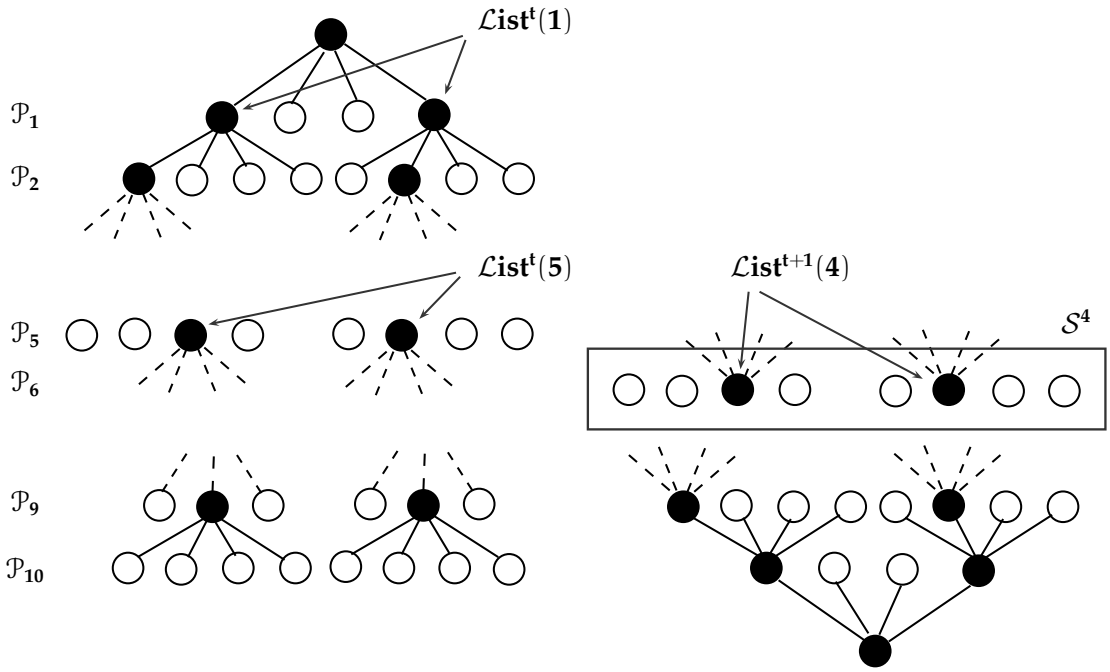

Figure 2.1: Example of algorithm F&B

In the following we give a step-by-step description of algorithm F&B. In describing the algorithm we use a counter $\mathtt{flag}$ that records the number of consecutive iterations in which the best upper bound $\mathbf{z}_{UB}$ does not improve. When $\mathtt{flag}$ takes value 2 the algorithm stops because no further improvements can be achieved in the following iterations.

Description of heuristic F&B

1. (Initialization): Set $t = 1$, $\mathbf{z}_{UB} = \infty$ and $\mathtt{flag} = 0$.

2. (build the tree associated with iteration t): set $\mathtt{flag} = \mathtt{flag}+1$ and let $\mathcal{L}ist^{t}(0) = \{\{\emptyset\}\}$. Repeat the following step 3 for each level $h = 1, \ldots, n$.

3. (generate the node set $\mathcal{S}^h$): set $\mathcal{S}^h = \{\emptyset\}$. If t is odd set $k = h$ otherwise set $k = n - h + 1$. Repeat for each node $S \in \mathcal{L}ist^t(h-1)$:

   - For each item $s \in \mathcal{P}_k$ let $S' = S \cup \{s\}$:
     If S' is feasible set $\mathcal{S}^h = \mathcal{S}^h \cup S'$.
     If S' is feasible and $\mathbf{z}(S') < \mathbf{z}_{UB}$ set $\mathbf{z}_{UB} = \mathbf{z}(S')$ and $flag = 0$.

4. (compute label $\psi(S)$ for each $S \in \mathcal{S}^h$): For each node $S \in \mathcal{S}^h$ compute $\psi(S)$ according to expression (2.32) and let $\overline{S}$ be the partial solution of $\mathcal{L}ist^{t-1}(n-h)$ producing the minimum in expression (2.32). If $S \cup \overline{S}$ is a feasible solution and $\mathbf{z}(S \cup \overline{S}) < \mathbf{z}_{UB}$ update $\mathbf{z}_{UB} = \mathbf{z}(S \cup \overline{S})$ and set $flag = 0$.

5. (Extract the subset $\mathcal{L}ist^t(h) \subseteq \mathcal{S}^h$):

   - If $|\mathcal{S}^h| \leqslant \Delta$ set $\mathcal{L}ist^t(h) = \mathcal{S}^h$.
   - If $|\mathcal{S}^h| > \Delta$ let $\mathcal{L}ist^t(h)$ be the set containing the Δ partial solutions of $\mathcal{S}^h$ having the smallest label $\psi(\cdot)$.

6. If $flag = 2$ the upper bound $\mathbf{z}_{UB}$ has not been improved in the last two consecutive iterations: Stop.

7. Set $t = t + 1$. If $t = NIT$, then Stop, otherwise return to step 2.

2.6.4 Some applications of algorithm F&B

In the next section we give a detailed description of F&B when applied to the NBP. However, algorithm F&B described in Sections 2.6.1, 2.6.2 and 2.6.3 can also be applied to other combinatorial optimization problems than NBP. In this section we give some examples of such problems. The aim of this section is to show how different combinatorial optimization problems, other than NBP, can be reformulated to fit the general schema required to use algorithm F&B.

a) **Variable Size Bin Packing**

It is given a set $J = \{1, 2, \ldots, n\}$ of items, where each item $k \in J$ is associated

with a weight w_k, and a set $B = \{1, 2, \ldots, n\}$ of bins, where each bin $i \in B$ is associated with a capacity b_i and a cost c_i. The Variable Size Bin Packing Problem (VSBPP) is to select a subset $J^* \subseteq J$ of bins of minimum cost such that each item can be assigned to exactly one bin in J^* and the total weight of the objects in each bin $i \in J^*$ does not exceed its capacity b_i.

To use algorithm F&B, let $\mathcal{P}_k \subseteq B$ be the set of bins that can contain item k, for each item $k \in J$, that is, $\mathcal{P}_k = \{i \in B : b_i \leqslant w_k\}$. A partial solution S at level h of a forward tree built by algorithm F&B represents an assignment of each of the first h items to a bin, whereas a partial solution $\overline{S}$ at level $n - h$ of a backward tree represents an assignment of each of the last items $h + 1, \ldots, n$ to a bin. Therefore partial solutions S and $\overline{S}$ provide a complete solution $S \cup \overline{S}$. Notice however that the resulting solution $S \cup \overline{S}$ may be infeasible as the total weight of the objects assigned to a bin could exceed its capacity.

b) **Set Covering Problem**

It is given a set $A = \{1, 2, \ldots, n\}$ and a family $\mathcal{F} \subseteq 2^A$ of subsets of A, where each set $S \subseteq \mathcal{F}$ is associated with a cost w_S. The Set Covering Problem (SCP) is to select a subset $\mathcal{F}^* \subseteq \mathcal{F}$ such that $\mathcal{F}^*$ covers A at minimum cost, that is, $\cup_{S \in \mathcal{F}^*} S = A$ and $\sum_{S \in \mathcal{F}^*} w_S$ is minimized.

To use algorithm F&B, let $\mathcal{P}_k \subseteq \mathcal{F}$ be the family of sets covering element $k \in A$, that is, $\mathcal{P}_k = \{S \in \mathcal{F} : k \in \mathcal{S}\}$. Any pair S, $\overline{S}$ of partial solutions at level h and $n - h$ of a forward and backward tree, respectively, provides a cover of A, that is, a feasible SCP solution $S \cup \overline{S}$.

c) **Traveling Salesman Problem**

It is given a set $N = \{1, 2, \ldots, n\}$ of cities and a cost c_{ij} associated with each pair of cities $i, j \in N$, representing the travel cost from i to j. The Traveling Salesman Problem (TSP) is to find the cheapest tour that visits all cities in N exactly once starting and ending at city 1.

To use algorithm F&B let $\mathcal{P}_k = \{2, 3, \ldots, n\}$ be the set of cities that can be visited in position k of the tour, $k = 2, \ldots, n$. Then, two partial solutions S and $\overline{S}$ at

level h and $n - h$ of a forward and backward tree, respectively, provide a, not necessarily feasible, tour by specifying the city visited in each position k, $k = 1, 2, \ldots, n$.

Notice that in describing the applicability of algorithm F&B for solving the VSPBB and the TSP a non trivial issue has been left unspecified. Since the algorithm can generate infeasible solutions, it is necessary to define an appropriate penalty function $\alpha()$ to correctly define the node labels according to expression (2.32).

Penalty functions are often used within genetic algorithms as a tool to penalize the fitness of infeasible solutions [see 66, 105, 114]. An alternative approach is described by Chu and Beasley [34] who propose to store separately two labels for each solution representing fitness and unfitness scores, respectively. Fitness and unfitness are then used to drive the parent selection and solution replacement mechanisms within a genetic algorithm.

The effectiveness of a penalty function and its most effective integration within the general framework provided by algorithm F&B strongly depends on the structure of the problem to be solved. Since the aim of this chapter is to study the NBP, this topic is not investigated here.

2.6.5 Solving the NBP using Algorithm F&B

As described in Section 2.5 the NBP can be modeled defining for each commodity $k = 1, \ldots, p$ a set $\mathcal{P}^k$ containing all the simple paths in G' from the origin node $s_k \in V$ to the destination node $t_k \in V$. Then, a NBP solution is represented by an ordered list of p paths, one path from each set $\mathcal{P}_k$. Here, objects are paths and each forward and backward tree has one level for each commodity. As for algorithm PEM each partial solution $S = (P^1_{j_1}, \ldots, P^k_{j_k}, \ldots, P^h_{j_h})$ corresponds to an ordered list of h paths where $P^k_{j_k}$ is the j_k-th element of the path set $\mathcal{P}_k$, $k = 1, \ldots, h$. The weight a^k_i of each object $P^k_{j_k} \in \mathcal{P}_k$ corresponds to the demand d_k of commodity

$k \in R$, and the NBP is to select a single path from each path set $\mathcal{P}_k$, $k = 1, ..., p$, to obtain a complete solution $S = (P^1_{j_1}, ..., P^k_{j_k}, ..., P^p_{j_p})$ of minimum cost $\mathbf{z}(X)$.

Notice that, since there is no restriction on the maximum number of base capacities u_e that can be installed on each edge $e \in E$, at each iteration t of algorithm F&B any two partial solutions $S \in \mathcal{S}(h)$ and $\overline{S} \in \mathcal{L}ist^{t-1}(n-h)$ always provide a feasible solution $S \cup \overline{S}$ of cost $\mathbf{z}(S \cup \overline{S})$. This means that the label $\beta(S)$ of any node S computed by means of expression (2.32) always represents a valid upper bound on the NBP.

Forward trees are built following the enumerative scheme of the PEM algorithm by considering the commodities in the order $(1, 2, \ldots, p)$, while backward trees are built as in PEM but considering the commodities in the reverse order $(p, p-1, \ldots, 1)$. The path subsets $\mathcal{P}_k$, $k \in R$, are generated, and the commodities are numbered, as described in algorithm PEM. At each iteration $t > 1$ algorithm F&B builds the associated tree and computes the label $\beta(S)$ for each $S \in \mathcal{S}(h)$ generated at level h using expression (2.32). At iteration $t = 1$ we define $\beta(S) = \infty$, for each partial solution S generated at level $h < p$ and $\beta(S) = \mathbf{z}(S)$ for every S generated at level p. Notice that at each iteration t only the lists $\mathcal{L}ist^{t-1}(p-h)$, $h = 1, \ldots, p$, are used to compute the labels when expanding level h. Therefore one only needs to store a single list $\mathcal{L}ist(h)$ for each level $h = 1, \ldots p$ that is updated after expanding each level. For the sake of clarity we also report a step-by-step description of algorithm F&B for the NBP.

Description of heuristic F&B for the NBP

1. (Initialization) Generate the path sets $\mathcal{P}_k$, $k \in R$, (as described in Section 2.5) and number the commodities so that $|\mathcal{P}_1| \geqslant |\mathcal{P}_2| \geqslant \cdots \geqslant |\mathcal{P}_p|$. Initialize $\mathcal{L}ist(k) = \emptyset$, $k \in R$.
   Set $t = 1$ and $\mathbf{z}_{UB} = \infty$.

2. (Initialization of the enumerative tree associated with iteration t) Define $S = \emptyset$ and set $\mathcal{S}^0 = S$, $\mathcal{S}^k = \emptyset$, $k = 1, \ldots, p$, and $h = 0$.

3. (Expansion of the partial solutions $\mathcal{S}^h$) If t is *odd*, then set $k = h$, otherwise set $k = p - h$.

   - If $|\mathcal{S}^k| > \Delta$: let $\mathcal{L}ist(k)$ be the first Δ partial solutions in $\mathcal{S}^k$ for the lexicographical order of the pairs $(\sigma(S), \beta(S))$, $S \in \mathcal{S}^k$.
   - If $|\mathcal{S}^k| = \Delta$: set $\mathcal{L}ist(k) = \mathcal{S}^k$.

   Repeat the following step 4 for any $S \in U$.

4. (Expansion of the partial solution $S \in U$) For each $P \in \mathcal{P}_k$ consider the partial solution S' of cost $\mathbf{z}(S')$, given by expression (2.31), obtained by appending path P to S and set $\mathcal{S}^{k+1} = \mathcal{S}^{k+1} \cup \{S'\}$. We have two cases:

   (a) $h + 1 = p$: set $\beta(S') = \mathbf{z}(S')$.

   (b) $h + 1 < p$:

   - If $t = 1$, then set $\psi(S') = +\infty$.
   - If $t > 1$, then define $\psi(S') = \min_{\bar{S} \in \mathcal{L}ist(p-k)} \left[\mathbf{z}(S' \cup \bar{S})\right]$.

   If $\beta(S') < \mathbf{z}_{UB}$, then update $\mathbf{z}_{UB} = \beta(S')$.

5. Set $h = h + 1$. If $h < p$ then go to step 3.

6. Set $t = t + 1$. If $t = NIT$, then Stop, otherwise return to step 2.

Algorithm F&B is significantly faster than algorithm PEM, but in computing the upper bound $\psi(S)$ (see step 4) strongly depends on the path set $\mathcal{P}_k$ of commodity k associated with level h, and on the partial solutions $\bar{\mathcal{S}}^{p-h}$ generated at the previous iteration. Algorithm PEM partially avoids this drawback as it uses algorithm TPH to complete partial solutions. In fact, phase 2 of algorithm TPH can generate a solution containing one or more paths not included in the path sets $\mathcal{P}_k$, $k \in R$, generated at the beginning of algorithm PEM.

2.7. Heuristic RTS based on tabu search

In this section we describe an iterative procedure, called RTS, that performs at each iteration a Tabu Search algorithm (see [65]), called TS, using a different initial solution. TS assumes that each commodity $k \in R$ can be routed using one of the paths in the set $\mathcal{P}_k$ that is defined with respect to an initial NBP solution $(\mathbf{x}, \mathbf{f})$ as follows. For each commodity $k \in R$ let $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$ be the partial solution resulting from $(\mathbf{x}, \mathbf{f})$ after removing commodity k. Then, each set $\mathcal{P}_k$ contains the best χ shortest paths in G', from node s_k to node t_k, computed with respect to the incremental costs $\{\bar{c}^k_{ij}\}$ given by expressions (2.20) with respect to the partial solution $(\bar{\mathbf{x}}, \bar{\mathbf{f}})$.

The neighborhood structure is defined by all solutions $N(\mathbf{x}, \mathbf{f})$ that can be reached from the current solution $(\mathbf{x}, \mathbf{f})$ by replacing, for each commodity $k \in R$, the current path $P(k)$ with all other paths in $\mathcal{P}_k \setminus \{P(k)\}$. To avoid cycling, solutions that were recently examined are forbidden, or tabu, for a number of iterations. The next move is made to the non tabu neighbor of minimum cost of the current solution $(\mathbf{x}, \mathbf{f})$. If multiple non tabu solutions of minimum cost are found in $N(\mathbf{x}, \mathbf{f})$ a *diversity score* is computed for each of them with respect to the solutions in the tabu list TL, and the solution with the highest score is chosen. The diversity score $\varphi(\mathbf{x}', \mathbf{f}')$, of solution $(\mathbf{x}', \mathbf{f}') \in N(\mathbf{x}, \mathbf{f})$ is defined as follows:

$$\varphi(\mathbf{x}', \mathbf{f}') = \max_{(\tilde{\mathbf{x}}, \tilde{\mathbf{f}}) \in TL} \left[\sqrt{\sum_{e \in E} (x'_e - \tilde{x}_e)^2} \right] \tag{2.33}$$

Algorithm TS performs a fixed number of iterations NIT. At the first iteration RTS provides to TS the initial solution obtained by phase 1 of algorithm TPH. In the following iterations RTS gives to TS the best solution $(\mathbf{x}^*, \mathbf{f}^*)$ achieved at the previous iteration as the new starting solution.

Algorithm RTS terminates if the solution achieved at the end of an iteration is not better than the one achieved at the previous iteration.

To alleviate the time requirement for generating the neighbourhood set $N(\mathbf{x}, \mathbf{f})$, we apply the following criteria. If the best solutions obtained by two consecutive

iterations of RTS use the same path for a given commodity k, then such path is permanently assigned to commodity k in the following iterations of RTS.

In the description of algorithm RTS we use a binary label $\nu(k)$ for each commodity $k \in R$ that is initialized equal to 0 at the beginning of each iteration of RTS. During the execution of TS we set $\nu(k) = 1$ every time a better solution is found where the path associated with commodity k is different from the one used in the initial NBP solution given by RTS to TS.

Description of heuristic RTS

1. (Initialization) Let $(\mathbf{x}, \mathbf{f})$ be the solution of cost $\mathbf{z}_{UB}$ produced by phase 1 of algorithm TPH. Set $\mathbf{z}^* = \mathbf{z}_{UB}$, $(\mathbf{x}^*, \mathbf{f}^*) = (\mathbf{x}, \mathbf{f})$, $\mathbf{z}_{old} = +\infty$ and $TL = \{(\mathbf{x}, \mathbf{f})\}$. Define $\nu(k) = 0$, $\forall k \in R$.

The following steps 2 - 5 perform TS on the initial solution $(\mathbf{x}, \mathbf{f})$

2. (Initialization of TS) For each commodity $k \in R$ such that $\nu(k) = 0$, compute the path set $\mathcal{P}_k$ containing the χ shortest paths in G' with respect to the initial solution $(\mathbf{x}, \mathbf{f})$, as described above. Set $iter = 1$.

2. (Build the neighborhood set $N(\mathbf{x}, \mathbf{f})$) Set $N(\mathbf{x}, \mathbf{f}) = \emptyset$. For each commodity $k \in R$ such that $\nu(k) = 0$, add to $N(\mathbf{x}, \mathbf{f})$ every solution $(\mathbf{x}', \mathbf{f}')$ that is obtained replacing in $(\mathbf{x}, \mathbf{f})$ the current path $P(k)$, assigned to commodity k, with any other path in $\mathcal{P}_k \setminus \{P(k)\}$.

2. (Extract a neighbor solution) Let $U \subseteq N(\mathbf{x}, \mathbf{f}) \setminus TL$ be the subset of all non tabu solutions of minimum cost.
   Let $(\mathbf{x}', \mathbf{f}')$ be the solution in U of cost $\mathbf{z}'_{UB}$ such that $\varphi(\mathbf{x}', \mathbf{f}') = \max_{(\mathbf{x},\mathbf{f}) \in U} [\varphi(\mathbf{x}, \mathbf{f})]$.
   If $\mathbf{z}'_{UB} < \mathbf{z}^*$, then set $\nu(k) = 1$ for every commodity k that in solution $(\mathbf{x}', \mathbf{f}')$ uses a path different from the one used in solution $(\mathbf{x}^*, \mathbf{f}^*)$. Moreover, update $\mathbf{z}^* = \mathbf{z}'_{UB}$ and $(\mathbf{x}^*, \mathbf{f}^*) = (\mathbf{x}', \mathbf{f}')$.

2. (Update the tabu list TL) If TL is full remove the oldest solution from the list.
   Set $TL = TL \cup \{(\mathbf{x}', \mathbf{f}')\}$.
   Set $iter = iter + 1$. If $iter \leqslant NIT$ set $(\mathbf{x}, \mathbf{f}) = (\mathbf{x}', \mathbf{f}')$ and return to step 3.

2. (Termination criteria of RTS) If $\mathbf{z}_{old} > \mathbf{z}^*$, then set $R = R \setminus \{k \in R : v(k) = 0\}$, $(\mathbf{x}, \mathbf{f}) = (\mathbf{x}^*, \mathbf{f}^*)$, $\mathbf{z}_{old} = \mathbf{z}^*$ and return to step 2; otherwise, Stop.

2.8. Computational Experiments

The algorithms described in this chapter have been coded in ANSI C and experimentally evaluated on the two classes of NBP instances, called A and B, using a Pentium 4 running at 3.2 GHz with 3 Gb RAM. In the following we briefly describe the characteristics of the two classes of test instances and we compare the results achieved by the branch-and-cut algorithm of Atamtürk and Rajan [6] with those obtained by the new exact algorithm described in Section 2.3. We discuss the impact of the parameter tuning on the performance of the heuristic algorithms, and we report detailed computational results comparing the results achieved by the new heuristics with those obtained by the exact algorithm on both the classes of instances.

2.8.1 Test instances

Class A instances

The instances of class A were introduced by Atamtürk and Rajan [6] and are publicly available at `http://www.ieor.berkeley.edu/~atamturk/data/`.

These instances correspond to undirected graphs with up to 29 nodes and 61 edges whose characteristics are summarized in Table 2.1. The base capacities and the installation costs are fixed (i.e., all edges have the same capacity and installation cost), and commodity demands range between 5 and 60. There are four possible base capacities 4, 25, 60 and 120, having unit installation cost 50, 250, 450 and 720, respectively. For each problem listed in Table 2.1 five instances are given, one for each problem and capacity combination. In all instances the routing costs are defined as: $w^k_{ij} = d_k$, $\forall (i,j) \in A$, $\forall k \in R$.

Table 2.1: Instances of class A

Problem	Commodities	Nodes	Edges	Flow variables	Design variables	Constraints
1	70	29	61	8,540	61	2,181
2	58	18	29	3,364	29	1,120
3	93	27	37	7,178	37	2,612
4	87	24	42	7,308	42	2,196
5	81	27	36	5,832	36	2,284

Class B instances

The instances of class B correspond to complete undirected graphs with 30 nodes and one commodity for each edge. These problems have been randomly generated to imitate the characteristics of practical network loading problems arising in the design of telecommunication networks (see Barahona [17]). Table 2.2 summarizes the characteristics of class B instances. These instances are further partitioned into two subclasses B.1 and B.2 with respect to the method used for computing the installation costs c_e of base capacities. For each subclass we randomly generated 5 instances as follows:

- the node set V is randomly generated in the square $[3,000 \times 3,000]$;
- the base capacity u_e is set equal to 56 for each edge $e \in E$, imitating the link capacity of a DS0 channel (see Kousik et al. [79], and Magnanti et al. [89]);
- associated with each edge $e \in E$ there is a commodity k_e having as origin and destination nodes the endpoints of edge e (i.e. $s_{k_e} = i_e$, $t_{k_e} = j_e$);
- the commodity demands are integers chosen from the set $\{8, 16, 24\}$ with probability 70%, 20% and 10%, respectively, and all routing costs $\{w_{ij}^k\}$ are set equal to 0.

For the instances of subclass B.1, according to Table 4 presented in Kousik et al. [79], we used the costs for the annual leasing of DS0 channels for computing the

Table 2.2: Instances of class B

Commodities	Nodes	Edges	Flow variables	Design variables	Constraints
435	30	435	26,100	435	13,920

values $\{c_e\}$. Let $eucd(ij)$ be the Euclidean distance between nodes $i, j \in V$.

The edge costs $\{c_e\}$ for instances of subclass B.1 are computed as follows:

$$c_e = \begin{cases} 232 + 7.74\ eucd(i_e j_e), & \text{if } eucd(i_e j_e) \leqslant 50, \\ 435 + 3.68\ eucd(i_e j_e), & \text{if } 50 < eucd(i_e j_e) \leqslant 100, \\ 571 + 2.32\ eucd(i_e j_e), & \text{if } 100 < eucd(i_e j_e) \leqslant 500, \\ 1,081.4 + 1.30\ eucd(i_e j_e), & \text{if } eucd(i_e j_e) > 500. \end{cases}$$

For the instances of subclass B.2 the edge costs $\{c_e\}$ are computed as:

$$c_e = 572 + \ eucd(i_e j_e).$$

2.8.2 Parameter settings

The quality of the solutions achieved by the heuristics PEM, F&B and RTS (see sections 2.5, 2.6 and 2.7) strongly depends on the value of the input parameters. We made several preliminary experiments to identify good parameter settings for our heuristics. In our experiments we found that both heuristics PEM and F&B are particularly sensitive to the value of the parameters Δ and χ, and it was not possible to choose a parameter set which gives the best results on all the instances.

We found interesting the behavior of algorithm F&B with respect to the value of parameter Δ that controls how many nodes are expanded at each level of a tree. One could expect that, given enough time, the more nodes are expanded at each level, the better results are obtained. Indeed, we found that in most cases the best results were obtained by F&B when using small values of Δ, i.e., $\Delta \leqslant 20$. We

noticed that by using larger values of Δ the algorithm was able to obtain better results in the first iterations but it was then unable to improve them significantly later on. As an example, figure 2.2 plots the best upper bound achieved by F&B on an instance of class B.2 (problem 2) within one hour of CPU time, when Δ ranges between 1 and 100. Figure 2.2 shows that the best solutions are achieved when expanding a small number of nodes at each level.

This behavior was common to all the NBP instances, showing that the computing time is better invested in performing more iterations (thus refining the algorithm's knowledge of the solution space) rather than trying a wider exploration. This could be specially true at earlier iterations when the algorithm memory is largely ineffective in guiding the exploration toward good solutions. Figure 2.3 shows the percentage distance of the obtained solution costs, for increasing values of Δ, with respect to the best known upper bounds averaged over all B.1 instances. The general structure is the same as that of figure 2.3. It is interesting to notice that in general there seems to be a "hot-spot" between values of Δ ranging from 20 to 10 where the algorithm provides the best results.

Table 2.4 gives a numerical comparison between the results obtained by algorithm F&B on class B instances for different values of the parameter Δ. For each value of the parameter Table 2.4 reports:

- $\mathbf{z}_{\mathrm{UB}}$: cost of the best solution achieved by F&B within the time limit;
- Gap: percentage distance between the costs of the best known solution ($\mathbf{z}_{\mathrm{best}}$) and the one found by the algorithm (i.e. Gap = $100 \times (\mathbf{z}_{\mathrm{UB}} - \mathbf{z}_{\mathrm{best}})/\mathbf{z}_{\mathrm{best}}$);
- Time: total computing time in seconds.

Algorithm PEM behaved differently. It relies heavily on the heuristic TPH to obtain feasible solutions and guide the search of the solution space, in contrast to the adaptive memory mechanism of F&B. On average, algorithm PEM tends to give better results when expanding more nodes per level. However, the computing time required to perform each iteration quickly increases as Δ gets bigger because heuristic TPH must be run each time a new partial solution is created.

Table 2.3: Parameter configurations for PEM and F&B

Algorithm	Class A instances	Class B instances
PEM	$\Delta = 50\ \chi = 30$	$\Delta = 20\ \chi = 100$
F&B	$\Delta = 100\ \chi = 100$	$\Delta = 10\ \chi = 100$

Depending on the size of the instances, raising Δ over a certain point ($\Delta = 50$ for class A instances and $\Delta = 20$ for class A instances) does not seem to make much sense because it makes the algorithm unable to perform enough iterations.

Concerning the parameter χ that controls the size of the path sets $\mathcal{P}_k$ computed for each commodity $k \in R$, both PEM and F&B obtained the best results using quite big values of χ. On the other hand, algorithm RTS rarely had significant benefits when increasing the value of the parameter χ above 30, even on the bigger class B instances. This is probably due to the fact that at the end of each iteration algorithm RTS recomputes the path set $\mathcal{P}_k$ for each commodity $k \in R$ with respect to the best solution achieved in the previous iteration.

As a result of these investigations we decided to use in the final computational tests reported in Section 2.8.3 the values of the parameters Δ and χ that give, on average, the best results on each of the two classes of instances (i.e. $\Delta \leqslant 20$ for class B instances, $\Delta \geqslant 50$ for class A instances). Table 2.3 summarizes the parameter configurations used in these tests by algorithms PEM and F&B on the two classes of instances A and B. The maximum number of iterations NIT is not reported since for both PEM and F&B the value of this parameter was set equal to 50 for all instances. Concerning algorithm RTS we use the following parameter settings for all the instances:

$$\chi = 30, \quad NIT = 30,000, \quad \text{and} |TL| = 100.$$

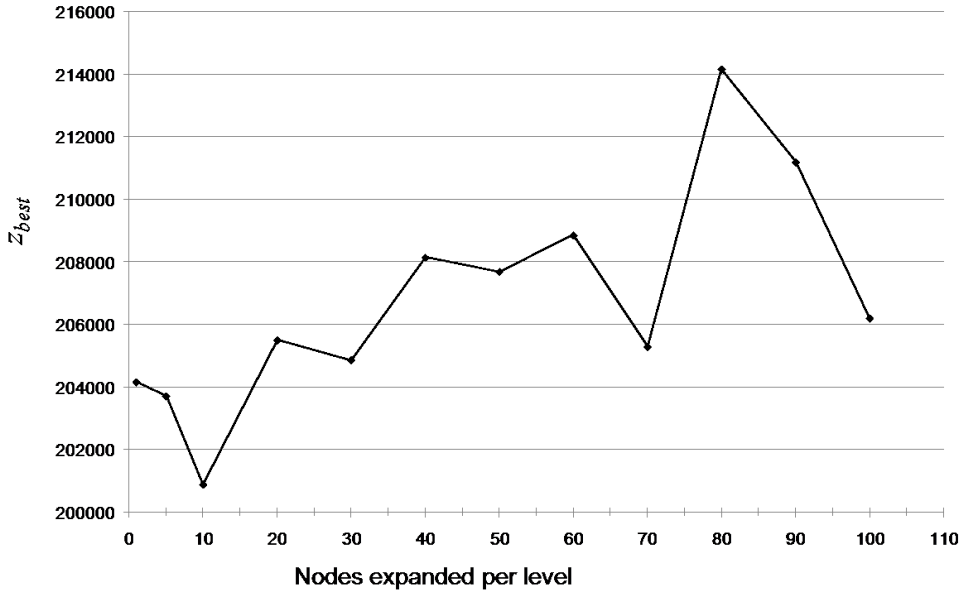

Figure 2.2: Problem 2 of class B.2: best results achieved for increasing values of the parameter Δ

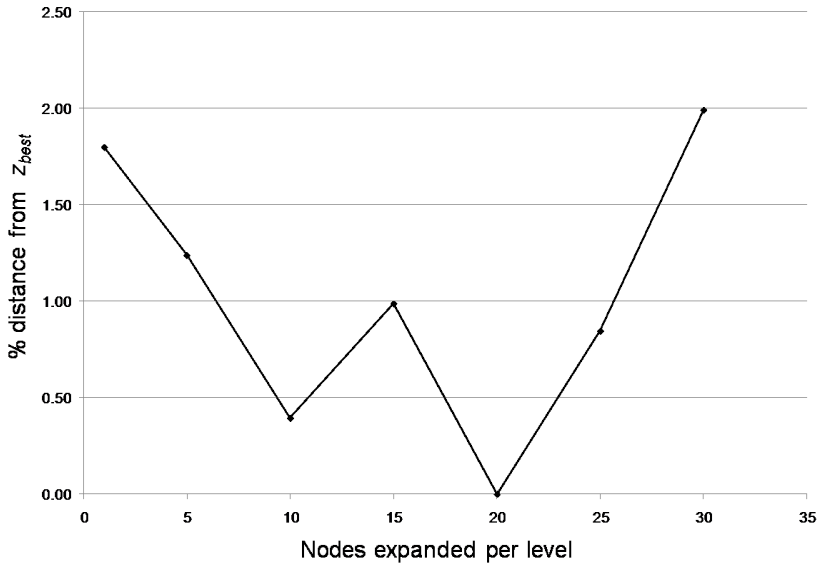

Figure 2.3: Class B.1 instances: percentage distance from the best result achieved for increasing values of the parameter Δ

Table 2.4: F&B: results for different values of parameter Δ on class B instances.

Prob.	F&B($\Delta = 1$)			F&B($\Delta = 5$)			F&B($\Delta = 10$)			F&B($\Delta = 20$)		
	zub	gap	time	zub	gap	time	zub	gap	time	zub	gap	time
B.1 1	279,340	1.66	14	275,705	0.34	213	**274,781**	0.00	1,263	275,381	0.22	3,197
B.1 2	291,269	1.43	16	296,623	3.29	234	**287,163**	0.00	893	290,144	0.00	893
B.1 3	295,853	5.28	17	286,697	2.02	399	285,442	1.58	992	**281,015**	0.00	2,571
B.1 4	285,773	0.98	17	**283,007**	0.00	198	284,547	0.54	754	284,684	0.59	2,695
B.1 5	288,574	1.57	14	290,839	2.37	245	289,003	1.72	735	**284,118**	0.00	3,568
Avg.	288,162	2.18	16	286,574	1.60	258	284,187	0.77	927	**283,068**	0.37	2,912
B.2 1	197,747	2.54	16	200,125	3.78	424	**192,842**	0.00	1,886	197,249	2.29	3,600
B.2 2	204,162	1.63	15	203,715	1.41	281	**200,890**	0.00	605	205,503	2.30	3,600
B.2 3	191,162	1.03	18	190,516	0.69	273	**189,214**	0.00	1,333	191,609	1.27	3,600
B.2 4	175,931	0.31	18	179,121	2.13	268	**175,389**	0.00	1,026	178,965	2.04	3,600
B.2 5	185,760	1.43	21	184,460	0.72	463	183,312	0.10	976	**183,138**	0.00	3,600
Avg.	190,952	1.39	18	191,587	1.74	342	**188,329**	0.02	1,165	191,293	1.58	3,600

2.8.3 Computational results

In this section we compare the performance of the algorithms described in this chapter on the two classes of instances A and B.

The computational results obtained for the instances of class A are reported in Tables 2.5 and 2.6. Table 2.5 compares the exact CPLEX(SF)) and the branch-and-cut of Atamtürk and Rajan [6] described in section 2.3. The table reports the LP-relaxation values of formulations NBP and SNBP, the results achieved by CPLEX 10.1 using formulations NBP and SNBP (columns CPLEX(F) and CPLEX(SF) respectively) and the results obtained by the branch-and-cut of Atamtürk and Rajan [6] (column B&C-AR). In CPLEX(F) and CPLEX(SF) we activated both the node heuristic and RINS heuristic. For each exact method Table 2.5 shows the percentage gap (column Endgap) between the best upper bound $\mathbf{z}_{UB}$ and the best lower bound z_{LB} at termination (i.e. $\texttt{Endgap} = 100 \times (\mathbf{z}_{UB} - \mathbf{z}_{LB})/\mathbf{z}_{LB}$) and the total computing time in seconds (column Time). Utilizing Dongarra [52], we estimated that the Sun Ultra 5 Workstation used by Atamtürk and Rajan [6] is approximately 10 times slower than the Pentium 4 3.2 Ghz used to run all our algorithms.

In Table 2.6 we compare the results obtained by the new heuristic algorithms with those obtained using CPLEX(SF). Table 2.6 reports, for each algorithm, the

following columns:

- $\mathbf{z}_{UB}$: cost of the best solution found;
- Gap: percentage gap between the cost of the best solution (say, $\mathbf{z}_{best}$) and the cost of the solution found by the algorithm (i.e. $100 \times (\mathbf{z}_{UB} - \mathbf{z}_{best}/\mathbf{z}_{best})$;
- Time: total computing time. One hour time limit was set for all heuristics and PEM never terminated before this time limit.

The last line of each block of instances reports the percentage average gap and the average computing times of each method.

Tables 2.7 and 2.8 show the results obtained by CPLEX(SF) and the new heuristics on the instances of class B within a one hour time limit. The final solution obtained by CPLEX(SF) within the imposed time limit was improved by running the CPLEX solution polishing heuristic for an extra hour of computing time.

Table 2.5 clearly indicates that formulation (SNBP) is stronger than NBP as it concerns the lower bounds given by the LP-relaxation, and often permits an improved performance of the exact procedures CPLEX(SF) with respect to CPLEX(F). The results of Table 2.5 do not allow to compare the two exact methods CPLEX(SF) and B&C-AR as the Pentium 4 used by CPLEX(SF) is faster than the Sun workstation used by B&C-AR. However, it is surprising that using CPLEX with a straightforward formulation and two simple sets of valid inequalities it is possible to solve the same instances solved by the very sophisticated branch and cut algorithm of Atamtürk and Rajan [6].

The results of Table 2.6 show that, in most cases, the best upper bound for the instances of class A is achieved by CPLEX(SF) with a time limit of 3,600 seconds. However this is not the case for all the instances. On the instances having a base capacity $u_e = 4$ the heuristic RTS always achieves the best solutions. Finally, Tables 2.7 and 2.8 indicate that CPLEX(SF) is not the best method for solving NBP instances having a large number of edges and commodities. In these cases the best solutions are achieved for all instances by algorithm F&B. Moreover algorithm TPH produces in a few seconds better solutions than CPLEX(SF) using two hours of computing time.

Table 2.5: F&B: Class A instances: exact methods

Cap.	Prob.	LP-relaxation of		CPLEX (F)		CPLEX (SF)		B&C-AR	
		NBP	SNBP	Endgap	Time [a]	Endgap	Time [a]	Endgap	Time [b]
	1	36,032.5	36,127.6	1.54	3,600	1.38	3,600	0.90	3,600
	2	19,217.5	19,266.5	0.00	85	0.00	93	0.00	117
4	3	29,025.0	29,177.0	0.00	115	0.00	140	0.00	728
	4	35,125.0	35,175.0	0.62	3,600	0.68	3,600	1.10	3,600
	5	63,735.0	63,886.0	0.03	3,600	0.00	22	0.00	72
Averages		36,627.0	36,726.4	0.44		0.41		0.40	
	1	29,520.0	30,101.5	2.72	3,600	2.09	3,600	9.30	3,600
	2	15,755.0	16,290.7	0.00	594	0.00	651	0.00	3,600
25	3	23,800.0	24,946.7	0.00	658	0.00	10	0.00	125
	4	28,825.0	29,080.0	2.17	3,600	2.00	3,600	7.90	3,600
	5	52,210.0	52,783.5	0.00	141	0.00	24	0.00	1,878
Averages		30,022.0	30,640.5	0.98		0.82		3.44	
	1	23,007.5	24,515.4	4.60	3,600	4.77	3,600	24.80	3,600
	2	12,292.5	12,960.0	0.00	141	0.00	133	0.00	1,890
60	3	18,575.0	21,235.1	0.00	118	0.00	15	0.00	289
	4	22,525.0	23,418.1	3.46	3,600	3.23	3,600	10.70	3,600
	5	40,685.0	43,285.1	0.00	91	0.00	4	0.00	106
Averages		23,417.0	25,082.7	1.61		1.60		7.10	
	1	19,100.0	23,122.1	14.33	3,600	13.22	3,600	24.90	3,600
	2	10,215.0	12,705.7	0.00	298	0.00	337	0.10	3,600
120	3	15,440.0	21,394.1	0.00	12	0.00	3	0.00	67
	4	18,745.0	20,559.5	4.78	3,600	7.03	3,600	9.50	3,600
	5	33,770.0	38,665.9	0.00	14	0.00	4	0.00	152
Averages		19,454.0	23,289.4	3.82		4.05		6.90	

(a): Seconds of a Pentium 4 at 3.2 GHz

(b): Seconds of a Sun Ultra 5 workstation

Table 2.6: Class A instances: heuristic algorithms

Cap.	Prob.	CPLEX (SF)			PEM [(a)]		F&B			RTS			TPH [(b)]	
		z_{UB}	Gap	Time	z_{UB}	Gap	z_{UB}	Gap	Time	z_{UB}	Gap	Time	z_{UB}	Gap
4	1	36,975	0.20	3,600	36,935	0.09	37,155	0.69	1,243	**36,900**	0.00	336	37,070	0.46
	2	**19,640**	0.00	93	**19,640**	0.00	**19,640**	0.00	230	**19,640**	0.00	231	19,775	0.69
	3	**29,400**	0.00	140	29,425	0.09	29,425	0.09	128	**29,400**	0.00	124	29,455	0.19
	4	35,560	0.07	3,600	35,580	0.13	35,735	0.56	804	**35,535**	0.00	617	35,920	1.08
	5	**64,150**	0.00	22	64,155	0.01	64,190	0.06	313	**64,150**	0.00	298	64,185	0.05
Averages		37,145	0.05	1,491	37,147	0.06	37,229	0.28	544	**37,125**	0.00	321	37,281	0.49
25	1	**31,870**	0.00	3,600	32,130	0.82	32,635	2.40	1,933	33,495	5.10	246	32,840	3.04
	2	**16,780**	0.00	651	16,975	1.16	17,015	1.40	184	17,000	1.31	353	17,435	3.90
	3	**26,175**	0.00	10	**26,175**	0.00	26,185	0.04	129	**26,175**	0.00	127	26,410	0.90
	4	**30,205**	0.00	3,600	30,465	0.78	31,060	2.75	510	30,230	0.08	378	31,300	3.54
	5	**53,905**	0.00	24	54,140	0.44	53,910	0.01	315	**53,905**	0.00	327	54,360	0.84
Averages		**31,787**	0.00	1,577	31,977	0.64	32,161	1.32	614	32,161	1.30	286	32,469	2.45
60	1	**27,075**	0.00	3,600	27,915	3.10	28,145	3.95	1,242	30,265	11.78	224	28,960	6.96
	2	**14,195**	0.00	133	14,200	0.04	15,080	6.23	145	14,200	0.04	347	15,170	6.87
	3	**23,165**	0.00	15	**23,165**	0.00	**23,165**	0.00	129	**23,165**	0.00	114	**23,165**	0.00
	4	**25,050**	0.00	3,600	25,925	3.49	26,945	7.56	518	26,825	7.09	543	26,920	7.47
	5	**45,840**	0.00	4	46,265	0.93	45,945	0.23	317	46,335	1.08	430	46,345	1.10
Averages		**27,065**	0.00	1,470	27,494	1.51	27,856	3.60	470	28,158	4.00	332	28,112	4.48
120	1	**29,465**	0.00	3,600	29,960	1.68	30,765	4.41	1,237	30,950	5.04	326	31,025	5.29
	2	**15,755**	0.00	337	16,410	4.16	**15,755**	0.00	144	16,410	4.16	313	15,970	1.36
	3	**24,515**	0.00	3	**24,515**	0.00	**24,515**	0.00	129	**24,515**	0.00	120	**24,515**	0.00
	4	**25,060**	0.00	3,600	26,300	4.95	26,540	5.91	514	27,010	7.78	466	27,930	11.45
	5	**42,215**	0.00	4	42,220	0.01	42,245	0.07	316	**42,215**	0.00	279	42,230	0.04
Averages		**27,402**	0.00	1,509	27,881	2.16	27,964	2.08	468	28,220	3.40	301	28,334	3.63
Overall		**30,850**	0.01	1,512	31,125	1.09	31,303	1.82	524	31,416	2.17	310	31,549	2.76

(a): Reaches a time limit of 3,600 sec. in all problems

(b): Terminates within 2 sec. in all problems

Table 2.7: Class B.1 instances: heuristic algorithms

Prob.	CPLEX(SF) [(a)]			PEM [(b)]		F&B			RTS			TPH		
	z_{UB}	Endgap	Gap	z_{UB}	Gap	z_{UB}	Gap	Time	z_{UB}	Gap	Time	z_{UB}	Gap	Time
1	320,281	32.94	16.56	293,058	6.65	**274,781**	0.00	1,263	298,703	8.71	3,600	294,034	7.01	31
2	337,136	32.43	17.40	303,799	5.79	**287,163**	0.00	893	327,668	14.11	3,371	314,051	9.36	32
3	309,701	27.71	8.50	300,875	5.41	**285,442**	0.00	992	322,367	12.94	3,600	306,463	7.36	31
4	331,173	32.91	16.39	298,088	4.76	**284,547**	0.00	754	303,266	6.58	3,600	307,136	7.94	27
5	327,312	31.02	13.26	302,164	4.55	**289,003**	0.00	735	325,027	12.46	3,245	312,829	8.24	32
Avg	325,121	31.40	14.42	299,597	5.43	**284,187**	0.00	927	315,406	10.96	3,483	306,903	7.98	31

(a): Reaches a time limit of 3,600 sec. in all problems and uses additional 3,600 sec. for running CPLEX polishing heuristic
(b): Reaches a time limit of 3,600 sec. in all problems

Table 2.8: Class B.2 instances: heuristic algorithms

Prob.	CPLEX(SF) [(a)]			PEM [(b)]		F&B			RTS			TPH		
	z_{UB}	Endgap	Gap	z_{UB}	Gap	z_{UB}	Gap	Time	z_{UB}	Gap	Time	z_{UB}	Gap	Time
1	225,785	31.74	17.08	207,824	7.77	**192,842**	0.00	1,886	217,037	12.55	3,600	210,122	8.96	30
2	232,121	31.41	15.55	218,781	8.91	**200,890**	0.00	605	226,979	12.99	2,842	219,064	9.05	33
3	216,794	31.13	14.58	196,450	3.82	**189,214**	0.00	1,333	208,583	10.24	3,434	205,648	8.69	29
4	197,212	29.77	12.44	188,224	7.32	**175,389**	0.00	1,026	203,076	15.79	3,344	191,218	9.03	32
5	220,353	34.87	20.21	194,955	6.35	**183,312**	0.00	976	207,458	13.17	3,600	195,481	6.64	33
Avg	218,453	31.78	15.97	201,247	6.83	**188,329**	0.00	1,165	212,627	12.95	3,364	204,307	8.47	31

(a): Reaches a time limit of 3,600 sec. in all problems and uses additional 3,600 sec. for running CPLEX polishing heuristic
(b): Reaches a time limit of 3,600 sec. in all problems

2.9. Summary

In this chapter we proposed four new heuristic algorithms for the non-bifurcated capacitated network design problem (NBP), called TPH, PEM, F&B and RTS and we described a formulation of the NBP, called SNBP, incorporating only a subset of two well known classes of valid inequalities for the NBP. The results achieved by solving formulation SNBP by means of the integer programming solver CPLEX 10.1 show that CPLEX, using the simple inequalities described in Section 2.3, is competitive with a recent branch-and-cut algorithm described in Atamtürk and Rajan [6]. We reported computational results on two classes of NBP instances (called class A and B) and compared the results achieved by the new heuristic algorithms against those obtained by CPLEX solving formulation SNBP.

The results show that CPLEX using 1 hour of computing time, achieves solutions that are, on average, 1-3% better than those achieved by the new heuristics on the small instances of class A but is outperformed by the new algorithms on the bigger class B instances. On this last class, CPLEX using 2 hours of computing time, obtains solutions that are 14-15% worse than those obtained by algorithm F&B in less than 20 minutes. Moreover, on class B instances, algorithm TPH achieves solutions which are, on average, 8% better than CPLEX in half a minute of computing time.

Chapter 3

The period routing problem

This chapter presents an exact algorithm for solving strategic and tactical multi-period vehicle routing problems that can be modeled as Period Vehicle Routing Problems (PVRPs). The PVRP is defined on a time horizon of several days and consists in assigning appropriate combinations of delivery days to customers, and in designing a set of delivery routes for every day of the planning period. The objective is to service all customers assigned to each day while minimizing the overall routing cost. We describe an integer programming formulation of the PVRP that is used to derive different lower bounds and an exact solution method. Computational results on test instances from the literature and on new sets of test instances show the effectiveness of the proposed method.

3.1. Introduction

The Period Vehicle Routing Problem (PVRP) is the problem of designing a set of routes for a homogeneous fleet of vehicles located at a central depot for each day of a given p-day period. These routes service customers with known demands. Each vehicle performs at most one route per day and at most m_k vehicles are available at day k. Each route starts and finishes at the depot and services a total customer demand that is smaller than or equal to the vehicle capacity Q.

Each customer i must be visited on f_i different days during the period and

requires q_i units of product every time he is visited. The f_i visits of customer i can only occur in one of a given number of allowable day-combinations. For example, a customer may require to be visited twice during a 5-day period and that these visits can take place on one of the following day-combinations: Monday-Thursday, Monday-Friday or Tuesday-Friday.

A solution to the PVRP simultaneously assigns a day-combination to each customer and designs the vehicle routes for each day so that each customer is visited the required number of times, the number of routes on each day does not exceed the number of vehicles available on that day and the total cost of the routes over the p-day period is minimized.

3.1.1 Special cases of the PVRP

The PVRP considered in this chapter contains as special cases the single-depot Capacitated Vehicle Routing Problem (CVRP), the Multi-Depot Vehicle Routing Problem (MDVRP) and the Tactical Planning Vehicle Routing Problem (TPVRP). These problems are now briefly defined.

a) The Capacitated Vehicle Routing Problem (CVRP)

The PVRP becomes the single depot CVRP when the planning period is of one day only and every customer must be visited exactly once from the depot. Moreover, any PVRP instance where every customer has frequency 1 and can be visited on any day of the period can be transformed into a single day CVRP with $m = m_1 + ... + m_p$ vehicles located at the depot. Any optimal solution of the resulting CVRP instance can be converted into an optimal solution of the original PVRP by splitting the CVRP routes over the p days of the period so that at most m_k routes are assigned to day k.

b) The Multi-Depot Vehicle Routing Problem (MDVRP)

The MDVRP is an extension of the CVRP where a customer can be serviced from p depots, and inter-depot routes are not allowed [see 38]. The MDVRP

can be viewed as a special case of a one day PVRP where vehicles are located at several depots but each vehicle route starts and ends at the same depot.

c) Tactical Planning Vehicle Routing Problem (TPVRP)

The PVRP in its general form is a *strategic* model as, in practice, the routes of a solution of a p-day period are operated unchanged for several months. On the other hand, several tactical distribution problems where technical personnel visits to customers are to be planned over a given p-day period can be modeled as PVRPs. This problem, called Tactical Planning VRP (TPVRP), assigns a visit day over a short term horizon (say five days) to each customer of a set of customer requests, and designs the routes of each day in order to minimize the sum of routing and customer service costs. Depending on the application area also other specific constraints could be needed to model the problem.

In the TPVRP, routes of the first day of the period are executed while new customer requests are recorded. The plan is updated in a rolling horizon process by adding one day to the end of the period and solving a new TPVRP on the new portfolio obtained by removing customers visited and adding the new customer requests. The customer service cost is the sum of the service costs associated with the visit days assigned to the customers. The service cost for visiting a customer on the first day of its day window is zero, but visiting a customer on any other day of the period involves a service cost that is proportional to the number of days of delay with respect to the first day. Any solution having all customers serviced on their first allowable day has service cost equal to zero. However, this solution can either be (i) infeasible, as it could require too many vehicles on some day of the period or (ii) too expensive in terms of routing costs.

Practical applications of the TPVRP with specific constraints arise in different fields, such as food and beverage distribution (where routes must satisfy capacity constraints) and maintenance of logistics activities in the field force

planning (where routes are usually uncapacitated, but have duration limited to a working day).

3.1.2 Literature review

The PVRP has many practical applications in the grocery industry [see 29], the soft drink industry (vending machines), the automotive industry (parts distribution) and industrial gases distribution and refuse collection [see 110]. This problem also admits several variants in terms of the objectives, the specific constraints and the type of solutions that are sought [see 93]. All papers on the PVRP reported in the literature present heuristic methods. However, since no optimal solutions nor lower bounds are known, the quality of the solutions found by these heuristics is unknown.

Early heuristics were proposed by Russel and Igo [110]. More sophisticated heuristic algorithms were presented by Christofides and Beasley [31], Beasley and Tan [20], Russel and Gribbin [109], Chao et al. [30], Gaudioso and Paletta [58] and Cordeau et al. [38].

The tabu search proposed by Cordeau et al. [38] is capable of solving the PVRP as well as the Periodic Traveling Salesman Problem (PTSP) and the MDVRP. This algorithm is based on a heuristic procedure, called GENI [see 60], that is used to remove or insert customers from routes. In order to insert a new customer i in a route, this procedure selects from the route two neighbor vertices of i that are close to it, and evaluates 4-opt modifications of the tour around these two vertices to insert i. The tabu search algorithm starts by constructing a (not necessarily feasible) initial solution. First customers are arbitrarily ordered, and a feasible day combination is randomly assigned to each of them. For each day $k = 1, \ldots, P$ of the period the algorithm considers all customers assigned to day k in the specified order starting from those closest to the depot, and tries to group them in a route of day k. Whenever a customer cannot be inserted in the current route while leaving the route feasible, a new route is created for day k and successive customers will be inserted in this route. This ensures that only the

last route of each day can be infeasible. During the search the algorithm allows in fact infeasible solutions to be generated, but penalizes solution infeasibility in the objective function. The neighborhood of a solution is constructed by evaluating all solutions that can be reached using two possible moves: (i) remove a customer from a route and insert him into another or (ii) assign to a customer a different day-combination and either remove him from its current route or insert him into a different route that is allowed by the new day-combination. Therefore, moves can be identified by triplets $(customer, route, day)$, and each solution is identified by an ordered set of such triplets. Each move performed is made tabu for a fixed number of iterations, but an aspiration criteria is employed to partially revoke tabus, that is, the algorithm allows tabu moves that can lead to an improving solution. A diversification criterion is used to discourage frequent moves that lead to infeasible solutions. This algorithm is compared on 32 PVRP instances from the literature with the heuristics proposed in [20, 31, 109, 30]. The results show that the proposed tabu search algorithm obtains solutions of better quality than other algorithms from the literature on 24 out of 32 instances.

More recently new heuristic algorithms were proposed by Alegre et al. [2] and Hemmelmayr et al. [70].

Alegre et al. [2] present a scatter search algorithm for solving a problem of periodic pick-up of raw materials for a manufacturer of auto parts. In this problem the time horizon under consideration can be as long as 90 days, and the the assignment of visit days to customers is particularly critical. The proposed algorithm uses a two-phase approach that first assigns orders to days and then constructs the routes for each day. The scatter search procedure manipulates day assignments only and, once given a day assignment for all customers, uses a deterministic heuristic to obtain a feasible solution by solving the corresponding vehicle routing problems. Scatter search is characterized by the use of a Reference Set of solutions that are combined at each step of the search to generate new solutions and systematic rules that are used to update the Reference Set. Computational results over the same instances considered by Cordeau et al. [38] show

that in 21 out of 32 instances the results obtained by this algorithm are at least as good as those obtained with the other heuristics from the literature.

Hemmelmayr et al. [70] propose a heuristic algorithm for both the PVRP and the PTSP based on Variable Neighborhood Search (VNS). The algorithm constructs an initial solution by randomly assigning a day-combination to each customer and constructing the routes for each day using the Clarke and Wright savings algorithm [35]. A number of different neighborhoods are used. These are obtained by applying move operators (remove a segment of a route and insert it into another), cross-exchange operators (exchange two segments between two routes) and change-combination operators (change day-combinations of a limited number of customers). The algorithm extracts new solutions from the different neighborhoods of the current solution and tries to improve them using 3-opt as a local search procedure. Each solution generated in this way undergoes an acceptance test that evaluates the improvement in the objective function with respect to the current solution using a scheme that is inspired by Simulated Annealing. The first solution accepted becomes the new current solution, and the whole process is started again using the corresponding neighborhoods. In contrast to the classical VNS schema also non-improving or even infeasible solutions can be accepted when exploring the neighborhoods of the current solution. Improving solutions are always accepted, whereas inferior solutions are accepted with a probability that depends on a given temperature parameter and the difference between the costs of the new solution (possibly including penalty costs if it is infeasible) and the current solution. Computational results show that the VNS algorithm is competitive with those of Cordeau et al. [38] and Alegre et al. [2]. With respect to solution quality this algorithm outperforms the other heuristics on instances where customers require higher frequency of visits and it is able to improve the previously best known solutions on 8 out of 32 problems.

Mourgaya and Vanderbeck [94] described a column generation based heuristic for tactical planning VRPs that restricts its attention to scheduling visits and assigning them to vehicles, but leaves sequencing decisions for an underlying

operational model. A recent survey on the PVRP and its extensions covering different modeling and solution methods can be found in the chapter of Francis et al. [56].

Very few publications on the TPVRP can be found in the optimization literature. Applications to field force planning and optimization in the areas of services building equipment and water distribution can be found in Tang et al. [116] and Bostel et al. [26], respectively. In Tang et al. [116] a tabu search heuristic is developed to solve a planned-maintenance scheduling problem arising in the maintenance of services building equipment such as heating, ventilation and air conditioning systems and escalators. Bostel et al. [26] describe a memetic heuristic and a column generation based heuristic for a multiperiod planning and routing problem of technical personnel that arises in the water distribution sector. In this problem daily technician schedules and routes must be determined in order to meet customer visit requirements.

To our knowledge, no exact methods have been proposed in the literature for both the PVRP and the TPVRP. Exact algorithms for the CVRP are due to Baldacci et al. [14], Lysgaard et al. [87], Fukasawa et al. [57] and Baldacci et al. [16], whereas the only exact methods published for the MDVRP are due to Laporte et al. [81, 82] and Baldacci and Mingozzi [13].

3.1.3 Contributions

In this chapter, we describe an exact solution method for the PVRP based on a set partitioning-like integer formulation of the problem and three relaxations that are used by five different bounding procedures to derive valid lower bounds.

The first relaxation converts the PVRP into a variant of the single-day period CVRP where each customer i must be visited f_i times, routes must satisfy customer-to-customer objections in addition to the capacity constraints, and a route can be in solution more than once. From this relaxation we derive two bounding procedures that generalize two of the bounding methods proposed by

Baldacci et al. [16]. The second and third relaxations are specific to the PVRP and are used to derive new bounding procedures.

Each lower bound corresponds to the cost of a different near-optimal dual solution of the LP-relaxation of the integer formulation. The final lower bound and the corresponding dual solution are used to generate a reduced integer problem containing only the variables having a reduced cost smaller than the difference between a known upper bound and the lower bound. The optimal PVRP solution is obtained by solving the reduced integer problem using a general purpose integer programming solver.

The main contributions of this chapter are the new bounding procedures and an exact algorithm for the PVRP. As noted above, neither lower bounds nor exact algorithms have been proposed in the literature for the PVRP. Further, extensive computational results on PVRP test instances from the literature and on newly generated TPVRP test instances show that the lower bounds obtained are tight and that the exact algorithm can solve for the first time several test instances involving up to 199 customers.

The remainder of this chapter is organized as follows. Section 3.2 describes the PVRP and introduces the notation used in this chapter. Section 3.3 describes a mathematical formulation of the PVRP and three relaxations that are used to derive valid lower bounds. Section 3.4 presents the exact algorithm. Sections 3.5, 3.6 and 3.7 describe the bounding procedures based on the three relaxations given in Section 3.3. Section 3.8 reports computational results on test instances from the literature and on new sets of test instances. Finally, Section 3.9 contains some concluding remarks.

3.2. Description of the PVRP and its special cases

The PVRP is defined on a complete undirected graph $G = (V', E)$ with a planning horizon of p days where the set V' contains $n+1$ vertices. The set V' is partitioned as $V' = \{0\} \cup V$, where vertex 0 represents the depot and the subset $V = \{1, \ldots, n\}$

represents n customers. A fleet of m_k identical vehicles of capacity Q is available on day k at the depot to supply the customers.

Each customer $i \in V$ specifies a *service frequency* f_i, a set C_i of allowable day-combinations of f_i visit days and a quantity q_i of product that customer i must receive every time he is visited. The customer set V is further partitioned as $V = V^1 \cup V^2$, where V^1 contains all customers having frequency equal to one (i.e., $V^1 = \{i \in V : f_i = 1\}$) and V^2 contains the customers having frequency greater than or equal to two (i.e., $V^2 = \{i \in V : f_i \geqslant 2\}$).

The visit days of a day-combination are represented by a column of a (0-1) matrix $[a_{ks}]$ of p rows, where $a_{ks} = 1$ if and only if day k is a required visit day in day-combination s. Hereafter, we assume that C_i is the index set of those columns of matrix $[a_{ks}]$ corresponding to allowable day-combinations of customer $i \in V$. $P = \{1, \ldots, p\}$ denotes the set of days of the planning period and $V_k \subset V$ the subset of customers that can be visited on day $k \in P$ (i.e., $V_k = \{i \in V : \sum_{s \in C_i} a_{ks} \geqslant 1\}$). With each day $k \in P$ is associated a non-negative cost matrix $[d_{ij}^k]$, where d_{ij}^k represents the cost for traversing edge $\{i, j\}$ on day k.

A *feasible* route on a day $k \in P$ is a simple circuit in G passing through the depot and a subset of customers of V_k and such that the sum of the customer demands is less than or equal to Q. Each vehicle can perform at most one route for each day and the cost of a route on a day $k \in P$ is given by the sum of the costs d_{ij}^k of the edges traversed by the route.

The PVRP consists in assigning to each customer a day-combination and designing at most m_k routes for each day k of the planning period so that each customer $i \in V$ is visited f_i times and the sum of the route costs is minimized.

The special cases of the PVRP discussed in Section 3.1.1 can be described as follows.

a) The CVRP corresponds to the PVRP where $p = 1$ and $f_i = 1$ for every customer $i \in V$.

b) Any MDVRP instance with p depots can be converted into an equivalent

PVRP instance as follows. Let $[\hat{d}_{ij}]$ be a $(n+p \times n+p)$ symmetric cost matrix, where $\hat{d}_{n+k\,i}$ is the travel cost for going from depot $k = 1, \ldots, p$ to customer $i \in V$. Each day k of the period corresponds to a depot k where m_k vehicles of capacity Q are stationed. Every customer $i \in V$ has frequency $f_i = 1$. The (0-1) matrix $[a_{ks}]$ is the identity matrix of order $(p \times p)$ and each day-combination of customer i corresponds to a depot that can service customer i, that is, C_i represents the subset of depots that can service customer $i \in V$. The edge cost matrix $[d^k_{ij}]$ is defined as $d^k_{ij} = \hat{d}_{n+k\,j}$, if $i = 0$; $d^k_{ij} = \hat{d}_{ij}$, otherwise, $\forall k \in P$ and $\forall\{i, j\} \in E$.

c) The TPVRP is defined on a complete undirected graph as the PVRP. Let $\hat{d}_{ij}$ be the travel cost associated with each edge $\{i, j\} \in E$. With each customer $i \in V$ is associated a frequency $f_i = 1$ and a day window $[e_i, l_i]$, with $e_i \geqslant 1$ and $l_i \leqslant p$, where e_i and l_i represent the first and the last day of the period during which customer i can be visited, respectively. Moreover, let $\tau_i(k)$, $k \in [e_i, l_i]$, be the *service cost* of customer $i \in V$ on day k. Any TPVRP instance can be converted into an equivalent PVRP instance as follows. Let matrix $[a_{ks}]$ be the $(p \times p)$ identity matrix. Associate with each customer $i \in V$ the set of allowable day-combinations defined as $C_i = \{s : s \in [e_i, l_i]\}$. Define the edge cost matrix $[d^k_{ij}]$ as $d^k_{ij} = \hat{d}_{ij} + \tau_i(k)/2 + \tau_j(k)/2$, $\forall k \in P$, $\forall\{i, j\} \in E$ (we assume that $\tau_0(k) = 0$, $\forall k \in P$). The cost of a route on day k of the PVRP instance defined above is equal to the sum of the routing costs and the service costs of the customers visited by the route in the original TPVRP instance. Thus, the objective of the resulting PVRP problem is to minimize the sum of routing and service costs.

3.3. Mathematical formulation and relaxations

In this section we describe a set partitioning-like formulation of the PVRP and three relaxations that are used to derive valid lower bounds on the PVRP.

Let $\mathcal{R}^k$ be the index set of all routes of day $k \in P$ visiting the customers in V_k and let $\mathcal{R}^k_i \subseteq \mathcal{R}^k$ be the index set of the route subset covering customer $i \in V_k$. We

use R_ℓ^k and c_ℓ^k to indicate the subset of customers and the cost of route $\ell \in \mathcal{R}^k$ on day $k \in P$, respectively.

Let y_{is} be a (0-1) binary variable which is equal to 1 if and only if customer $i \in V^2$ is assigned to day-combination $s \in C_i$. Define a (0-1) binary variable x_ℓ^k which is equal to 1 if and only if route $\ell \in \mathcal{R}^k$ of day $k \in P$ is in solution. The PVRP can be formulated as follows:

$$(F) \quad z(F) = \min \quad \sum_{k\in P} \sum_{\ell\in\mathcal{R}^k} c_\ell^k x_\ell^k \tag{3.1}$$

$$\text{s.t.} \quad \sum_{k\in P} \sum_{\ell\in\mathcal{R}_i^k} x_\ell^k = f_i \qquad (\forall i \in V), \tag{3.2}$$

$$\sum_{\ell\in\mathcal{R}_i^k} x_\ell^k - \sum_{s\in C_i} a_{ks} y_{is} = 0 \quad (\forall i \in V^2, \forall k \in P), \tag{3.3}$$

$$\sum_{\ell\in\mathcal{R}^k} x_\ell^k \leqslant m_k \qquad (\forall k \in P), \tag{3.4}$$

$$x_\ell^k \in \{0,1\} \qquad (\forall \ell \in \mathcal{R}^k, \forall k \in P), \tag{3.5}$$

$$y_{is} \in \{0,1\} \qquad (\forall s \in C_i, \forall i \in V^2). \tag{3.6}$$

Constraints (3.2) impose that each customer i is visited exactly f_i times. In particular, as a consequence of the definition of customer subsets V_k and of the route sets $\mathcal{R}^k$, $k \in P$, every customer $i \in V^1$ is visited exactly once in one of its allowable days. Constraints (3.3) impose that every customer $i \in V^2$ is visited exactly f_i times during the f_i days of the day-combination assigned to the customer. Constraints (3.4) force the solution to contain at most m_k routes on each day $k \in P$. Finally, constraints (3.5) and (3.6) are the integrality constraints for the decision variables x_ℓ^k and y_{is}, respectively.

Notice that constraints (3.2) and (3.3) imply that exactly one day combination $s \in C_i$ is assigned to each customer $i \in V^2$, that is, $\sum_{s\in C_i} y_{is} = 1$, $\forall i \in V^2$. In fact, adding constraints (3.3) associated with customer $i \in V^2$ on the p-days of the period and considering that $\sum_{k\in P} a_{ks} = f_i$, $\forall s \in C_i$, we obtain $\sum_{k\in P} \sum_{\ell\in\mathcal{R}_i^k} x_\ell^k = \sum_{s\in C_i} f_i y_{is}$. From the latter expression and equations (3.2) we derive $\sum_{s\in C_i} f_i y_{is} = f_i$, that is, $\sum_{s\in C_i} y_{is} = 1$.

Formulation F can be easily reduced when the PVRP corresponds to the MDVRP. In this case we have $V^2 = \emptyset$ as $f_i = 1$, $\forall i \in V$. Since the customer-depot objections are implicitly imposed in the construction of the route sets $\mathcal{R}^k$, $\forall k \in P$, formulation F does not require constraints (3.3) and variables y_{is} and constraints (3.6) are not needed. Therefore, the mathematical formulation of the MDVRP is given by expressions (3.1), (3.2), (3.4) and (3.5). Similar reductions of formulation F apply for both the CVRP and the TPVRP.

Problem F is not practical to solve even for moderate size instances. In the following we describe three relaxations of problem F that are used to derive different lower bounds.

In the following LF denotes the LP-relaxation of problem F.

3.3.1 Relaxation RF

Relaxation RF corresponds to an integer problem that is derived from problem F by ignoring constraints (3.3) and (3.6) and by reducing the number of variables by means of the following observations.

Let us associate with each edge $\{i, j\} \in E$ the cost $\overline{d}_{ij} = \min_{k \in P}\{d^k_{ij}\}$. It is quite easy to observe that with respect to costs $\overline{d}_{ij}$ the cost of the least cost route in G visiting any subset of customers S is a lower bound on the cost c^k_ℓ of any route $\ell \in \mathcal{R}^k$, $k \in P$, such that $R^k_\ell = S$.

Let I be a $(n \times n)$ symmetric (0-1) *compatibility* matrix, where $I_{ij} = 0$ if and only if there exists at least a day $k \in P$ in which both i and j can be visited (i.e., $i, j \in V_k$, for some $k \in P$). Let $\overline{\mathcal{R}}$ be the index set of all least cost routes in G with respect to the modified edge costs $\overline{d}_{ij}$ that satisfy, in addition to the capacity constraints, the following constraint:

$$\sum_{i \in \overline{R}_\ell} \sum_{j \in \overline{R}_\ell} I_{ij} = 0, \tag{3.7}$$

where $\overline{R}_\ell$ is the subset of customers visited by route ℓ.

We denote by $\overline{c}_\ell$ the cost of route $\ell \in \overline{\mathcal{R}}$ and by $\overline{\mathcal{R}}_i \subseteq \overline{\mathcal{R}}$ the index subset of the routes visiting customer $i \in V$. Problem RF is to select at most $\overline{m} = \sum_{k \in P} m_k$

routes from $\overline{\mathcal{R}}$, where each route can be in the solution more than once, so that each customer $i \in V$ is visited exactly f_i times. Let $\overline{x}_\ell$ be a non-negative integer variable representing the number of times the route $\ell \in \overline{\mathcal{R}}$ is in the solution. Problem RF is the following.

$$(RF) \quad z(RF) = \min \quad \sum_{\ell \in \overline{\mathcal{R}}} \overline{c}_\ell \overline{x}_\ell \tag{3.8}$$

$$\text{s.t.} \quad \sum_{\ell \in \overline{\mathcal{R}}_i} \overline{x}_\ell = f_i \qquad (\forall i \in V), \tag{3.9}$$

$$\sum_{\ell \in \overline{\mathcal{R}}} \overline{x}_\ell \leqslant \overline{m}, \tag{3.10}$$

$$\overline{x}_\ell \geqslant 0 \text{ integer} \quad (\forall \ell \in \overline{\mathcal{R}}). \tag{3.11}$$

Any solution $(\mathbf{x}', \mathbf{y}')$ of problem F of cost $z'(F)$ can be transformed into a feasible RF solution of cost smaller than or equal to $z'(F)$. In fact, any route $\ell \in \mathcal{R}^k$, $k \in P$, corresponds to a route $\ell' \in \overline{\mathcal{R}}$ of cost $\overline{c}_{\ell'} \leqslant c^k_\ell$ because from the definition of the edge costs $\overline{d}_{ij}$ we have $\overline{d}_{ij} \leqslant d^k_{ij}$, $\forall \{i, j\} \in E$. Thus, $z(RF)$ is a valid lower bound on the PVRP.

Problem RF cannot be solved in practice. However, we can compute valid lower bounds on $z(RF)$ by finding near-optimal solutions of the dual, called DRF, of the LP-relaxation of problem RF. Let $\mathbf{w} = (w_0, w_1, \ldots, w_n)$ be a vector of $n + 1$ dual variables, where variables w_i, $\forall i \in V$, are associated with constraints (3.9) and w_0 is associated with constraint (3.10). The dual problem DRF is as follows.

$$(DRF) \quad z(DRF) = \max \quad \sum_{i \in V} f_i w_i + \overline{m} w_0 \tag{3.12}$$

$$\text{s.t.} \quad \sum_{i \in \overline{R}_\ell} w_i + w_0 \leqslant \overline{c}_\ell \quad (\forall \ell \in \overline{\mathcal{R}}), \tag{3.13}$$

$$w_i \in \mathbb{R} \qquad (\forall i \in V), \tag{3.14}$$

$$w_0 \leqslant 0. \tag{3.15}$$

In Section 3.5 we describe two bounding procedures called H^1 and H^2that produce two valid lower bounds LH1 and LH2, respectively. H^1 is based on q-route

relaxation [see 33] while H^2 is a dual ascent method based on column generation and on feasible routes. As in a q-route a customer can be visited more than once, we have that LH1 $\leqslant$ LH2. To improve the computational efficiency of H^2it is convenient to execute in sequence H^1 and H^2so that the DRF solution achieved by H^1 can be used to initialize the master problem of H^2.

3.3.2 Relaxation LF

A second relaxation of the PVRP is based on the dual problem of LF, called DF.

Associate dual variables v_i, $i \in V$, with constraints (3.2), u_{ik}, $i \in V^2$, $k \in P$, with constraints (3.3) and σ_k, $k \in P$, with constraints (3.4). Let us partition the subset of customers R_ℓ^k visited by each route $\ell \in \mathcal{R}^k$ into two subsets A_ℓ^k and B_ℓ^k, where $A_\ell^k = R_\ell^k \cap V^1$ and $B_\ell^k = R_\ell^k \cap V^2$. Problem DF is as follows.

$$(DF) \quad z(DF) = \max \quad \sum_{i \in V} f_i v_i + \sum_{k \in P} m_k \sigma_k \tag{3.16}$$

$$\text{s.t.} \quad \sum_{i \in R_\ell^k} v_i + \sum_{i \in B_\ell^k} u_{ik} + \sigma_k \leqslant c_\ell^k \quad (\forall \ell \in \mathcal{R}^k, \forall k \in P), \tag{3.17}$$

$$\sum_{k \in P} a_{ks} u_{ik} \geqslant 0 \quad (\forall s \in C_i, \forall i \in V^2), \tag{3.18}$$

$$u_{ik} \in \mathbb{R} \quad (\forall i \in V^2, \forall k \in P), \tag{3.19}$$

$$v_i \in \mathbb{R} \quad (\forall i \in V), \tag{3.20}$$

$$\sigma_k \leqslant 0 \quad (\forall k \in P). \tag{3.21}$$

Both problems LF and DF are impractical to solve. However, a valid lower bound on the PVRP can be computed as a near optimal DF solution without generating all constraints (3.17).

It is quite easy to observe that the set of DRF solutions is contained in the set of DF solutions. In fact, any DRF solution $\mathbf{w}'$ of cost $z'(DRF)$ corresponds to a DF solution $(\mathbf{u}', \mathbf{v}', \boldsymbol{\sigma}')$ of cost $z'(DF) = z'(DRF)$ where $\mathbf{u}' = \mathbf{0}$, $v_i' = w_i'$, $\forall i \in V$, and $\sigma_k' = w_0'$, $\forall k \in P$. Thus, we have $z(DF) \geqslant z(DRF)$.

In Section 3.6 we describe two dual ascent heuristics for solving DF, called H^3 and H^4, that produce two near optimal DF solutions of cost LH3 and LH4, respectively. Both H^3 and H^4 are extensions to DF of H^1 and H^2. H^3 is based on the q-route relaxation, while H^4 is a column generation method based on feasible routes.

It can be shown that the following relation holds: LH3 $\leqslant$ LH4, LH1 $\leqslant$ LH3 and LH2 $\leqslant$ LH4. No dominance relation exists between LH2 and LH3 since H^2 is based on feasible routes and H^3 on q-routes. Procedure H^4 must be executed either after H^2 or H^3in order to be computationally efficient as it needs a near-optimal DF solution to initialize the master problem.

3.3.3 Relaxation $\overline{\text{LF}}$

A better relaxation than LF, called $\overline{\text{LF}}$, is obtained by adding to LF the following extensions of two well-known valid inequalities designed for the CVRP.

a) *Generalized Capacity Constraints*. Let $\overline{P} \subseteq P$ and let $\hat{q}_i$, $i \in V$, be a lower bound on the total demand delivered to customer i during days in $\overline{P}$. The value $\hat{q}_i$ can be computed as follows. For each $i \in V$:

$$\hat{q}_i = \begin{cases} 0, & \text{if } f_i = 1 \text{ and } \{k \in P : i \in V_k\} \not\subseteq \overline{P}, \\ q_i, & \text{if } f_i = 1 \text{ and } \{k \in P : i \in V_k\} \subseteq \overline{P}, \\ q_i \min_{s \in C_i}\{\sum_{k \in \overline{P}} a_{ks}\}, & \text{if } f_i \geqslant 2. \end{cases}$$

Let $\mathcal{S} = \{S : S \subseteq V, |S| \geqslant 2\}$ be the set of all customer subsets. The generalized capacity constraints derive from the observation that the total demand delivered to a subset of customers $S \in \mathcal{S}$ within the set of days $\overline{P}$ must be greater than or equal to $\sum_{i \in S} \hat{q}_i$. Thus, any feasible F solution must satisfy the following inequalities:

$$\sum_{k \in \overline{P}} \sum_{\ell \in \mathcal{R}^k(S)} x_\ell^k \geqslant \left\lceil \sum_{i \in S} \hat{q}_i / Q \right\rceil, \quad \forall S \in \mathcal{S}, \tag{3.22}$$

where $\mathcal{R}^k(S) = \{\ell \in \mathcal{R}^k : R_\ell^k \cap S \neq \emptyset\}$.

b) *Clique Inequalities*. Let $H = (\mathcal{R}, \overline{E})$ be the *conflict graph* associated with the routes of sets $\mathcal{R}^k$, $\forall k \in P$. Node i of graph H represents route $\ell(i)$ of day $k(i)$ (i.e., $\ell(i) \in \mathcal{R}^{k(i)}$). The edge set $\overline{E}$ contains every pair $\{i, j\}$, $i < j$, such that the set of conflicting customers $S = R^{k(i)}_{\ell(i)} \cap R^{k(j)}_{\ell(j)} \neq \emptyset$ and one of the following two conditions is satisfied:

   (a) $k(i) = k(j)$;

   (b) $k(i) \neq k(j)$ or at least one conflicting customer either has frequency equal to one or cannot be serviced on both days $k(i)$ and $k(j)$ according to the customer day combinations (i.e., either $\min_{h \in S}\{f_h\} = 1$ or $a_{k(i)s} + a_{k(j)s} \leqslant 1$, $\forall s \in C_h$, for some $h \in S$).

   Let $\mathcal{C}$ be the set of all cliques of graph H. Then, any feasible F solution must satisfy the following inequalities:

$$\sum_{i \in C} x^{k(i)}_{\ell(i)} \leqslant 1, \ \forall C \in \mathcal{C}. \tag{3.23}$$

Let $z(\overline{LF})$ be the optimal solution cost of $\overline{LF}$. Relaxation $\overline{LF}$ is solved by means of a standard column and cut generation method, called CG, that is described in Section 3.7. The initial master problem is generated by using the dual solution $(\mathbf{u}, \mathbf{v}, \boldsymbol{\sigma})$ computed by the bounding procedures H^1, H^2, H^3 and H^4, as described in the following Section 3.4.1. The master problem is then solved using the simplex algorithm. At each iteration a limited subset of inequalities (3.22) and (3.23) that are violated by the current fractional solution is added to the master (see Section 3.7).

We denote by LCG the cost $z(\overline{LF})$ of the final $\overline{LF}$ solution achieved by CG.

The computational performance of CG strongly depends on the quality of the DF solution used to initialize the master problem. Thus, it is worth executing CG after procedures H^1, H^2 or H^3, H^4.

3.4. An exact method for solving the PVRP

The exact algorithm for the PVRP, described in this section, is based on the exact methods proposed by Baldacci et al. [16] for the CVRP and by Baldacci and Mingozzi [13] for the Heterogeneous Vehicle Routing Problem (HVRP). The algorithm consists in generating a reduced problem $\widehat{F}$ that is obtained from F by replacing each set $\mathcal{R}^k$ with the subset $\widehat{\mathcal{R}}^k$, $k \in P$, and by adding two subsets of inequalities (3.22) and (3.23), respectively. The subsets $\widehat{\mathcal{R}}^k$, $k \in P$, are generated in such a way that any optimal $\widehat{F}$ solution is also optimal for F.

Problem $\widehat{F}$ is solved by means of an integer linear programming solver. The main component of the algorithm is the bounding method that combines different bounding procedures based on the three relaxations described in the previous section.

3.4.1 Computing the lower bound

A valid lower bound LCG on the PVRP can be obtained by executing in sequence procedures H^1, H^2, H^3, H^4 and CG, where each procedure starts from the dual solution found by the previous one. However, this method can be time consuming. This method can be improved without affecting the quality of the final lower bound by removing from the sequence either H^1 and H^2 or H^3 and H^4, according to the type of PVRP instance considered.

In our computational experience we observed that $LH2$ is almost equal to $LH4$ for PVRP instances having the following characteristics:

(a) the edge costs of graph G do not depend from the day of the period. In this case a cost d_{ij} is associated with each edge $\{i,j\} \in E$ and $d^k_{ij} = d_{ij}$, $\forall k \in P$, $\forall \{i,j\} \in E$;

(b) the customer frequencies and the day-combinations are such that every customer can be serviced on any day of the period (i.e., $V_k = V$, $\forall k \in P$).

For these instances LH4 is not significantly greater than lower bound LH2, and it is worth executing the sequence of bounding procedures H^1, H^2 and CG. The initial master of CG is generated using the DF dual solution of cost LH2 produced by H^2.

On the other hand, we observed that for PVRP instances not satisfying at least one of the two conditions (a) and (b) both LH3 and LH4 are better than LH1 and LH2, respectively. Whenever condition (a) is not satisfied LH3 is significantly greater than LH2 and it is computationally convenient to execute in sequence H^3, H^4 and CG. Notice that conditions (a) and (b) are satisfied by all PVRP instances proposed in the literature, but they are not satisfied by the PVRP instances corresponding to the MDVRP and the TPVRP.

3.4.2 Finding an optimal solution

The exact method performs the following steps.

1. Compute lower bound LCG by executing either the sequence of procedures H^1, H^2 and CG or H^3, H^4 and CG according to the type of PVRP instances as described in Section 3.4.1. Let $\widehat{S}$ and $\widehat{C}$ be the subsets of constraints (3.22) and (3.23) that are saturated by the final $\overline{LF}$ solution achieved by CG.

2. Generate the route subsets $\widehat{\mathcal{R}}^k$, $k \in P$, containing all routes whose reduced costs with respect to the dual solution of $\overline{LF}$ achieved by CGis smaller than the gap $z(UB)-LCG$, where $z(UB)$ is a valid upper bound on the PVRP.

3. Define the reduced problem $\widehat{F}$ that is derived from F replacing each $\mathcal{R}^k$ with the subset $\widehat{\mathcal{R}}^k$, $k \in P$, generated at step 2 and adding the subsets of constraints (3.22) and (3.23) that are saturated by the optimal $\overline{LF}$ solution obtained by CG.

4. Solve problem $\widehat{F}$ using an integer programming solver (e.g., CPLEX [43]).

In generating the route subsets $\widehat{\mathcal{R}}^k$, $k \in P$, we use procedure GENROUTE described in Baldacci et al. [16] and in Baldacci and Mingozzi [13].

The effectiveness of the proposed exact method strongly depends on the quality of the final lower bound LCG achieved by CG. As LCG improves the reduced costs of the routes of an optimal PVRP solution get smaller and the size of subsets $\widehat{\mathcal{R}}^k$, $\forall k \in P$, tends to get smaller making problem $\widehat{F}$ easier to solve.

3.5. Lower bounds based on relaxation RF

In this section we describe the bounding procedures H^1 and H^2. Procedures H^1 and H^2 compute two lower bounds LH1 and LH2 on the PVRP that correspond to the costs of two feasible solutions of the dual problem DRF described in Section 3.3.1. Procedures H^1 and H^2 are extensions to relaxation RF of two methods originally proposed by Baldacci et al. [16] for the CVRP and are based on the following theorem.

Theorem 2 *Associate penalties* $\lambda_i \in \mathbb{R}$, $\forall i \in V$, *with constraints* (3.9) *and* $\lambda_0 \leqslant 0$ *with constraint* (3.10). *A feasible* DRF *solution* $\mathbf{w}$ *of cost* $z(DRF(\boldsymbol{\lambda}))$ *is given by the following expressions:*

$$w_i = q_i \min_{\ell \in \overline{\mathcal{R}}_i} \{(\overline{c}_\ell - \lambda(\overline{R}_\ell))/q(\overline{R}_\ell)\} + \lambda_i, \ \forall i \in V, \quad \textit{and} \quad w_0 = \lambda_0, \tag{3.24}$$

where $q(\overline{R}_\ell) = \sum_{i \in \overline{R}_\ell} q_i$ *and* $\lambda(\overline{R}_\ell) = \sum_{i \in \overline{R}_\ell} \lambda_i$.

Proof. See Baldacci et al. [16].

A lower bound on problem RF is given by $\max_\lambda \{z(DRF(\boldsymbol{\lambda}))\}$ and can be computed using subgradient optimization. Let $\ell(i) \in \overline{\mathcal{R}}$ be the index of the route producing w_i in expressions (3.24) and let ρ_{ji} be the number of times that customer $j \in V$ is visited by route $\ell(i)$. It can be shown that a valid subgradient of the function $z(DRF(\boldsymbol{\lambda}))$ at point $\boldsymbol{\lambda}$ is given by the vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_n)$ that is computed as follows:

$$\theta_j = f_j - \sum_{i \in V} f_i \rho_{ji} q_i / q(\overline{R}_{\ell(i)}), \quad \forall j \in V, \quad \text{and} \quad \theta_0 = \overline{m} - \sum_{i \in V} q_i / q(\overline{R}_{\ell(i)}). \tag{3.25}$$

The following theorem shows that any feasible DRF solution provides a feasible solution of the dual problem DF of the same cost.

Theorem 3 *Let* $\mathbf{w}$ *be a feasible* DRF *solution of cost* $z(\mathrm{DRF})$*. A solution* $(\mathbf{u}, \mathbf{v}, \boldsymbol{\sigma})$ *of problem* DF *of cost* $z(\mathrm{DF}) = z(\mathrm{DRF})$ *is given by the following expressions:*

$$\begin{aligned} u_{ik} = &\quad 0, \quad \forall i \in V^2, \forall k \in P, \\ v_i = &\quad w_i, \quad \forall i \in V, \\ \sigma_k = &\quad w_0, \quad \forall k \in P. \end{aligned} \tag{3.26}$$

Proof: It is easy to see that the solution $(\mathbf{u}, \mathbf{v}, \boldsymbol{\sigma})$ defined by expressions (3.26) satisfies constraints (3.19), (3.20) and (3.21) and that $z(\mathrm{DF}) = z(\mathrm{DRF})$. Consider a route $\ell \in \mathcal{R}^k$ for a given $k \in P$. From inequalities (3.17) and expressions (3.26) we have:

$$\sum_{i \in A_\ell^k} v_i + \sum_{i \in B_\ell^k} u_{ik} + \sigma_k = \sum_{i \in \overline{R}_{\ell'}} w_i + w_0, \tag{3.27}$$

where ℓ' is the index of the route in $\overline{\mathcal{R}}$ such that $\overline{R}_{\ell'} = R_\ell^k$. As $\overline{c}_{\ell'} \leqslant c_\ell^k$ and $\mathbf{w}$ is a feasible DRF solution, then from inequalities (3.13) and (3.27) we obtain:

$$\sum_{i \in A_\ell^k} v_i + \sum_{i \in B_\ell^k} u_{ik} + \sigma_k \leqslant c_\ell^k. \tag{3.28}$$

This proves that solution $(\mathbf{u}, \mathbf{v}, \boldsymbol{\sigma})$ satisfies constraints (3.17) of problem DF. Moreover, because $u_{ik} = 0, \forall i \in V, \forall k \in P$, constraints (3.18) of problem DF are trivially satisfied. Finally, it is easy to see that $z(\mathrm{DF}) = z(\mathrm{DRF})$. □

3.5.1 Procedure H^1

In procedure H^1 we relax the requirement that each route in $\overline{\mathcal{R}}$ is a simple cycle in G and we extend the set $\overline{\mathcal{R}}$ to contain all the q-routes. This relaxation allows us to compute expressions (3.24) in pseudo-polynomial time as follows.

Associate with each edge $\{i, j\} \in E$ the modified cost $\tilde{d}_{ij} = \overline{d}_{ij} - \frac{1}{2}\lambda_i - \frac{1}{2}\lambda_j$. Compute for each customer $i \in V$ and each load value $q = q_i, \ldots, Q$ the modified cost $\phi(q, i)$ of the least cost cycle (not necessarily simple), called q-route, passing through the depot and customer i and such that the total load is equal to q. The values $\phi(q, i)$ can be computed using dynamic programming in time $O(Qn^2)$ [see

33] by relaxing constraints (3.7) but imposing $\overline{d}_{ij} = \infty, \forall\{i,j\} \in E$ such that $I_{ij} = 1$. It is quite easy to see that $\phi(q,i) \leqslant \min\{\overline{c}_\ell - \lambda(\overline{R}_\ell) : \ell \in \overline{\mathcal{R}}_i, q(\overline{R}_\ell) = q\}$. Therefore, a solution to expressions (3.24) is given by setting:

$$w_i = q_i \min_{q_i \leqslant q \leqslant Q} \{\phi(q,i)/q\} + \lambda_i, \ \forall i \in V, \text{ and } w_0 = \lambda_0. \tag{3.29}$$

H^1 performs $Maxt1$ iterations of subgradient optimization to compute LH1$=$ $\max_{\boldsymbol{\lambda}}\{z(DRF(\boldsymbol{\lambda}))\}$, where at each iteration the penalties $\lambda_i, \forall i \in V$, are modified using the subgradient vector $\boldsymbol{\theta}$ computed according to expressions (4.28). For a step-by-step description of procedure H^1see Baldacci et al. [16].

Let $\mathbf{w}^1$ be the final DRF solution of cost LH1 achieved by H^1 with respect to the penalty vector $\boldsymbol{\lambda}^1$. We denote by $(\mathbf{u}^1, \mathbf{v}^1, \boldsymbol{\sigma}^1)$ the DF solution obtained from $\mathbf{w}^1$ by setting $\mathbf{u}^1 = \mathbf{0}$, $v_i^1 = w_i^1, \forall i \in V$, $\sigma_k^1 = w_0^1, \forall k \in P$.

3.5.2 Procedure H^2

Procedure H^2 is an iterative method that uses column generation to solve equations (3.24) and subgradient optimization to compute LH2$=$ $\max_{\boldsymbol{\lambda}}\{z(DRF(\boldsymbol{\lambda}))\}$. H^2 is initialized by setting $\boldsymbol{\lambda} = \boldsymbol{\lambda}^1$ and LH2 $= 0$ and by generating a subset $\widehat{\mathcal{R}} \subseteq \overline{\mathcal{R}}$ containing the Δ^{min}-routes of minimum reduced cost with respect to the DF solution $\mathbf{w}^1$ obtained by H^1 (where Δ^{min} is a parameter defined a-priori). H^2 executes an a-priori defined number $Maxt2$ of macro iterations. On each macro iteration H^2 performs the following two steps:

1. *Solve the Master Problem*. The master problem is obtained by replacing in problem DRF the set $\overline{\mathcal{R}}$ with the subset $\widehat{\mathcal{R}}$. A near-optimal solution $\tilde{\mathbf{w}}$ of cost $\tilde{z}$ of the master problem is obtained by an iterative method that performs $Maxt3$ iterations. At each iteration it uses expressions (3.24) and subgradient optimization to modify the penalty vector $\boldsymbol{\lambda}$ in order to maximize the cost $\tilde{z}$ of the master dual solution $\tilde{\mathbf{w}}$.

2. Generate the largest subset $\mathcal{N}$ of routes having minimum reduced cost with respect to the dual master solution $\tilde{\mathbf{w}}$ and such that $|\mathcal{N}| \leqslant \Delta^a$, where Δ^a is a

parameter defined a-priori. If $\mathcal{N} = \emptyset$ and $\tilde{z}$ is greater than LH2, then update LH2$= \tilde{z}$, $\mathbf{w}^2 = \tilde{\mathbf{w}}$ and $\boldsymbol{\lambda}^2 = \boldsymbol{\lambda}$; otherwise, update $\widehat{\mathcal{R}} = \widehat{\mathcal{R}} \cup \mathcal{N}$.

The initial route subset $\widehat{\mathcal{R}}$ and the subset $\mathcal{N}$ are generated using procedure GENROUTE [see 16, 13].

We denote by $(\mathbf{u}^2, \mathbf{v}^2, \boldsymbol{\sigma}^2)$ the DF solution obtained from $\mathbf{w}^2$ by setting $\mathbf{u}^2 = \mathbf{0}$, $v_i^2 = w_i^2$, $\forall i \in V$, $\sigma_k^2 = w_0^2$, $\forall k \in P$.

3.6. Lower bounds based on relaxation LF

In this section we describe the bounding procedures H^3 and H^4 for computing the lower bounds LH3 and LH4 as the cost of two near-optimal DF solutions. Procedures H^3 and H^4 do not require the generation of the entire route sets $\mathcal{R}^k$, $k \in P$ and are based on the following theorem that is a generalization of Theorem 2 (see Section 3.5).

Theorem 4 *Associate with constraints* (3.2) *the penalties* $\mu_i \in \mathbb{R}$, $\forall i \in V$, *with constraints* (3.3) *the penalties* $\lambda_{ik} \in \mathbb{R}$, $\forall i \in V^2$, $\forall k \in P$, *and with constraints* (3.4) *the penalties* γ_k, $\forall k \in P$. *For each route* $\ell \in \mathcal{R}^k$, $k \in P$, *define* $\mu(R_\ell^k) = \sum_{i \in R_\ell^k} \mu_i$, $\lambda(R_\ell^k) = \sum_{i \in B_\ell^k} \lambda_{ik}$ *and* $q(R_\ell^k) = \sum_{i \in R_\ell^k} q_i$. *Compute:*

$$b_{ik} = q_i \min_{\ell \in \mathcal{R}_i^k} \{(c_\ell^k - \mu(R_\ell^k) - \lambda(R_\ell^k) - \gamma_k)/q(R_\ell^k)\} + \mu_i + \lambda_{ik}, \quad \forall i \in V, \forall k \in P. \tag{3.30}$$

A feasible DF *solution* $(\mathbf{u}, \mathbf{v}, \boldsymbol{\sigma})$ *of cost* $z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma}))$ *can be obtained as follows:*

$$\left.\begin{array}{lll} v_i = \min_{k \in P}\{b_{ik}\}, & \forall i \in V^1, & (a) \\ v_i = \dfrac{1}{f_i} \min_{s \in C_i} \left\{ \sum_{k \in P} a_{ks} b_{ik} \right\}, & \forall i \in V^2, & (b) \\ u_{ik} = b_{ik} - v_i, & \forall i \in V^2,\ \forall k \in P, & (c) \\ \sigma_k = \gamma_k, & \forall k \in P. & (d) \end{array}\right\} \tag{3.31}$$

Proof: First we show that the values assigned to $\mathbf{u}$ satisfy inequalities (3.18). Consider a day-combination $s' \in C_i$ of customer $i \in V^2$. From expressions (3.31.b) we have $f_i v_i \leqslant \sum_{k\in P} a_{ks} b_{ik}$, $\forall s \in C_i$, and from expressions (3.31.c) we derive $\sum_{k\in P} a_{ks'} u_{ik} = \sum_{k\in P} a_{ks'} b_{ik} - f_i v_i \geqslant 0$.

By substituting solution $(\mathbf{u}, \mathbf{v}, \boldsymbol{\sigma})$ given by expressions (3.31) in the left-hand-side of the dual constraints (3.17) of a given route $\bar{\ell} \in \mathcal{R}^k$ of day k and considering that $v_i \leqslant b_{ik}$, $k \in P$, $i \in V^1$ because of expressions (3.31.a), we obtain:

$$\sum_{i\in R^k_{\bar{\ell}}} v_i + \sum_{i\in B^k_{\bar{\ell}}} u_{ik} + \sigma_k = \sum_{i\in R^k_{\bar{\ell}}} v_i + \sum_{i\in B^k_{\bar{\ell}}} (b_{ik} - v_i) + \gamma_k \leqslant \sum_{i\in R^k_{\bar{\ell}}} b_{ik} + \gamma_k. \tag{3.32}$$

Since $\bar{\ell} \in \mathcal{R}^k_i$ for every $i \in R^k_{\bar{\ell}}$, from expressions (3.30) we have:

$$b_{ik} \leqslant q_i(c^k_{\bar{\ell}} - \mu(R^k_{\bar{\ell}}) - \lambda(R^k_{\bar{\ell}}) - \gamma_k)/q(R^k_{\bar{\ell}}) + \mu_i + \lambda_{ik}, \quad \forall i \in R^k_{\bar{\ell}}. \tag{3.33}$$

From inequalities (3.33) we obtain the following inequality:

$$\sum_{i\in R^k_{\bar{\ell}}} b_{ik} + \gamma_k \leqslant \sum_{i\in R^k_{\bar{\ell}}} q_i \frac{(c^k_{\bar{\ell}} - \mu(R^k_{\bar{\ell}}) - \lambda(R^k_{\bar{\ell}}) - \gamma_k)}{q(R^k_{\bar{\ell}})} + \mu(R^k_{\bar{\ell}}) + \lambda(R^k_{\bar{\ell}}) + \gamma_k = c^k_{\bar{\ell}}. \tag{3.34}$$

Finally, inequalities (3.32) and (3.34) show that the dual constraint (3.17) is satisfied by the duals $(\mathbf{u}, \mathbf{v}, \boldsymbol{\sigma})$ given by expressions (3.31), $\forall \bar{\ell} \in \mathcal{R}^k$, $\forall k \in P$. □

Both procedures H^3 and H^4 use subgradient optimization to solve $\max_{\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\gamma}}\{z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma})\})$. A valid subgradient of function $z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma})\})$ at point $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma})$ can be computed as follows.

Let $\ell(i, k) \in \mathcal{R}^k$ be the index of the route producing the value b_{ik} in expression (3.30), $\forall i \in V$, $\forall k \in P$, and let k_i and s_i be the day and the day-combination producing v_i in expressions (3.31.a) and (3.31.b), respectively.

Let δ^j_{ik} be the number of times that customer j is visited by the route $R^k_{\ell(i,k)}$ and let $\bar{q}_{ik} = q(R^k_{\ell(i,k)})$. We denote by U_k the subset of customers visited on day k, that is, $U_k = \{i \in V^2 : a_{ks_i} = 1\} \cup \{i \in V^1 : k_i = k\}$. Then, a valid subgradient of $z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma}))$ at point $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma})$ is given by the following vectors $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ whose components, corresponding to constraints (3.2), (3.3) and (3.4), respectively, are

computed as follows:

$$\left.\begin{aligned}
\beta_{jk} &= \sum_{i\in U_k} \delta^j_{ik} q_i/\overline{q}_{ik} - a_{ks_j}, && \forall j \in V^2,\ \forall k \in P,\\
\alpha_j &= \begin{cases} \displaystyle\sum_{k\in P}\sum_{i\in U_k} \delta^j_{ik} q_i/\overline{q}_{ik} - 1, & \text{if } j \in V^1,\\ \displaystyle\sum_{k\in P}\sum_{i\in U_k} \delta^j_{ik} q_i/\overline{q}_{ik} - f_j, & \text{if } j \in V^2, \end{cases}\\
\theta_k &= \sum_{i\in U_k} q_i/\overline{q}_{ik} - m_k, && \forall k \in P.
\end{aligned}\right\} \tag{3.35}$$

The penalty vectors $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ and $\boldsymbol{\gamma}$ are modified as follows:

$$\left.\begin{aligned}
\lambda_{ik} &= \lambda_{ik} - step\ \beta_{ik}, && \forall i \in V^2, \forall k \in P,\\
\mu_i &= \mu_i - step\ \alpha_i, && \forall i \in V,\\
\gamma_k &= \min\{0, \gamma_k - step\ \theta_k\}, && \forall k \in P,
\end{aligned}\right\} \tag{3.36}$$

where $step = \epsilon(z(UB) - z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma})))/(\sum_{k\in P}\sum_{i\in V^2} \beta^2_{ik} + \sum_{i\in V} \alpha^2_i + \sum_{k\in P} \theta^2_k)$, and ϵ is a positive constant.

3.6.1 Procedure H^3

As in procedure H^1 we extend the route sets $\mathcal{R}^k$, $\forall k \in P$, to contain all q-routes. This relaxation allows us to compute in pseudo-polynomial time a lower bound $\overline{b}_{ik} \leqslant b_{ik}$, $\forall i \in V$, $\forall k \in P$, as follows. Given the penalty vectors $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, define the *modified edge costs* $\{\tilde{d}^k_{ij}\}$ as follows:

$$\tilde{d}^k_{ij} = d^k_{ij} - \frac{1}{2}\pi_{ik} - \frac{1}{2}\pi_{jk}, \quad \forall \{i,j\} \in E,\ \forall k \in P, \tag{3.37}$$

where $\pi_{ik} = \lambda_{ik} + \mu_i$, $\forall i \in V^2$, $\forall k \in P$, and $\pi_{ik} = \mu_i$, $\forall i \in V^1$, $\forall k \in P$. Let $\phi(k, q, i)$ be the cost with respect to the modified edge costs $\{\tilde{d}^k_{ij}\}$ of a least cost q-route $\Omega(k, q, i)$ passing through the depot and vertex $i \in V_k$ on day $k \in P$, and such that the total demand of the customers visited is equal to q ($q_i \leqslant q \leqslant Q$). $\phi(k, q, i)$ provides a lower bound on the cost $c^k_\ell - \mu(R^k_\ell) - \lambda(R^k_\ell)$ of any feasible route $\ell \in \mathcal{R}^k_i$ of load $q(R^k_\ell) = q$. The values $\phi(k, q, i)$ can be computed using the

method proposed in Christofides et al. [33] for each day $k \in P$. Thus, values $\overline{b}_{ik}$ can be computed as follows:

$$\overline{b}_{ik} = q_i \min_{q_i \leqslant q \leqslant Q} \left\{ \frac{\phi(k, q, i)}{q} \right\} + \mu_i + \lambda_{ik}, \quad \forall i \in V, \ \forall k \in P. \tag{3.38}$$

Let $\overline{q}_{ik}$ be the value of q giving the minimum in expression (3.38). As $\overline{b}_{ik} \leqslant b_{ik}$, $\forall i \in V$ and $\forall k \in P$, Theorem 4 remains valid if in expression (3.31) each b_{ik} is replaced with $\overline{b}_{ik}$, $i \in V$, $k \in P$.

Computing lower bound LH3

Procedure H^3 uses subgradient optimization to compute LH3=$\max_{\boldsymbol{\lambda},\boldsymbol{\mu},\boldsymbol{\gamma}}\{z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma})\}$) and a corresponding DF solution $(\mathbf{u}^3, \mathbf{v}^3, \boldsymbol{\sigma}^3)$.

H^3 starts by initializing $LH3 = 0$, $\lambda_{ik} = 0$, $\forall i \in V^2$, $\forall k \in P$, $\mu_i = \lambda_i^2$, $\forall i \in V$, and $\sigma_k = \lambda_0^2$, $\forall k \in P$. We assume $\boldsymbol{\lambda}^2 = \mathbf{0}$, $\boldsymbol{\mu}^2 = \mathbf{0}$ in case neither H^1 nor H^2 is executed before H^3.

At each iteration H^3 performs the following two steps.

1. Compute the modified edge costs $\{\tilde{d}_{ij}^k\}$ according to expressions (3.37) and the values $\phi(k, q, i)$, $\forall k \in P$, $\forall i \in V$, $q_i \leqslant q \leqslant Q$. Compute the values $\overline{b}_{ik}$ using expressions (3.38) and the corresponding DF solution $(\mathbf{u}, \mathbf{v}, \boldsymbol{\sigma})$ of cost $z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma}))$ using expressions (3.31), where b_{ik} is replaced with $\overline{b}_{ik}$.

   If $z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma}))$ is greater than LH3, then update LH3=$z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma}))$, $\mathbf{u}^3 = \mathbf{u}$, $\mathbf{v}^3 = \mathbf{v}$, $\boldsymbol{\sigma}^3 = \boldsymbol{\sigma}$, $\boldsymbol{\lambda}^3 = \boldsymbol{\lambda}$, $\boldsymbol{\mu}^3 = \boldsymbol{\mu}$ and $\boldsymbol{\gamma}^3 = \boldsymbol{\gamma}$.

2. Update penalty vectors $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ and $\boldsymbol{\gamma}$ using expressions (3.35) and (3.36), where $\overline{q}_{ik}$ represents the value of q giving the minimum in expressions (3.38) and δ_{ik}^j is the number of times that j is visited by the q-route $\Omega(k, \overline{q}_{ik}, i)$. Notice that since a q-route is not necessarily simple, we may have $\delta_{ik}^j > 1$ for some customer j. This does not affect the correctness of expressions (3.35) in computing the subgradient of $z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma}))$ at point $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma})$.

Procedure H^3 terminates after $Maxt1$ iterations, where $Maxt1$ is a parameter defined a-priori.

3.6.2 Procedure H^4

Procedure H^4 is a column generation method that computes LH4 as the cost of a near optimal solution $(\mathbf{u}^4, \mathbf{v}^4, \boldsymbol{\sigma}^4)$ of the dual problem DF. H^4 differs from standard column generation methods as the master problem is solved heuristically using expressions (3.30) and (3.31) and subgradient optimization.

Procedure H^4 is initialized by setting $\boldsymbol{\lambda} = \boldsymbol{\lambda}^3$, $\boldsymbol{\mu} = \boldsymbol{\mu}^3$, $\boldsymbol{\gamma} = \boldsymbol{\gamma}^3$ and $LH4 = 0$ and by generating for each $k \in P$ the subset $\widehat{\mathcal{R}}^k \subseteq \mathcal{R}^k$ of the Δ^{min} routes having minimum reduced cost with respect to the DF solution $(\mathbf{u}^3, \mathbf{v}^3, \boldsymbol{\sigma}^3)$ obtained from procedure H^3, where Δ^{min} is a parameter defined a-priori. H^4 executes $Maxt2$ macro iterations, where $Maxt2$ is an a-priori defined number. On each macro iteration H^4 performs the following two steps.

1. *Solve the master problem*. The master problem is obtained from LF by replacing each $\mathcal{R}^k$ with $\widehat{\mathcal{R}}^k$, $\forall k \in P$. The dual solution of the master problem is computed by means of an iterative procedure based on subgradient optimization. Given the vectors $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ and $\boldsymbol{\gamma}$, the dual solution $(\mathbf{u}, \mathbf{v}, \boldsymbol{\sigma})$ of cost $z(DF(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma}))$ is computed using expressions (3.30) and (3.31), where $\mathcal{R}^k$ is replaced with $\widehat{\mathcal{R}}^k$, $\forall k \in P$. The initial values of $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ and $\boldsymbol{\gamma}$ are set equal to the best values achieved at the previous macro iteration and are iteratively updated as described in Section 3.6 using expressions (3.35) and (3.36). Let $(\mathbf{u}^*, \mathbf{v}^*, \boldsymbol{\sigma}^*)$ be the best dual solution of the master, of cost z^*, achieved after $Maxt3$ iterations.

2. *Check if the master solution* $(\mathbf{u}^*, \mathbf{v}^*, \boldsymbol{\sigma}^*)$ *is a feasible* DF *solution*. For each $k \in P$, generate the largest subset $\mathcal{N}^k \subset \mathcal{R}^k \setminus \widehat{\mathcal{R}}^k$ containing the routes having minimum reduced cost with respect to the dual master solution $(\mathbf{u}^*, \mathbf{v}^*, \boldsymbol{\sigma}^*)$ and such that $|\mathcal{N}^k| \leqslant \Delta^a$, where Δ^a is a parameter defined a-priori. We have two cases:

   a) if $\mathcal{N}^k = \emptyset$ $\forall k \in P$ and $z^* >$ LH4, update $LH4 = z^*$, $\mathbf{u}^4 = \mathbf{u}^*$, $\mathbf{v}^4 = \mathbf{v}^*$ and $\boldsymbol{\sigma}^4 = \boldsymbol{\sigma}^*$.

   b) if $\mathcal{N}^k \neq \emptyset$ for some $k \in P$, update $\widehat{\mathcal{R}}^k = \widehat{\mathcal{R}}^k \cup \mathcal{N}^k$, $\forall k \in P$.

The initial route subsets $\widehat{\mathcal{R}}^k$, $\forall k \in P$, and the subsets $\mathcal{N}^k$, $\forall k \in P$, are generated using procedure GENROUTE.

3.7. Lower bounds based on relaxation $\overline{LF}$

Procedure CG is a cut and column generation method that computes lower bound LCG as the cost of an optimal solution to problem $\overline{LF}$.

The initial master problem is obtained from $\overline{LF}$ by replacing the sets $\mathcal{R}^k$, $\forall k \in P$, with the route subsets $\widehat{\mathcal{R}}^k$, $\forall k \in P$, that are generated using procedure GENROUTE. For each day $k \in P$ the initial route subset $\widehat{\mathcal{R}}^k \subset \mathcal{R}^k$ contains the largest subset of routes of minimum reduced cost with respect to the best DF solution computed by the previous bounding procedures such that $|\widehat{\mathcal{R}}^k| \leqslant \Delta^{min}$. In addition, we set $\overline{\mathcal{S}} = \emptyset$ and $\overline{\mathcal{C}} = \emptyset$. At each iteration CG performs the following steps:

1. *Solve problem* $\overline{LF}$ using the simplex and let $(\overline{x}, \overline{y})$ be the optimal solution of $\overline{LF}$.

2. *Generate the route subsets* $\mathcal{N}^k$, $\forall k \in P$. Use procedure GENROUTE to generate the largest route subsets $\mathcal{N}^k \subset \mathcal{R}^k \setminus \overline{\mathcal{R}}^k$, $\forall k \in P$, containing the routes having minimum reduced cost with respect to the optimal dual solution of $\overline{LF}$ and satisfying $|\mathcal{N}^k| \leqslant \Delta^a$, $\forall k \in P$.

3. *Separate violated inequalities*. Generalized capacity constraints (3.22) and clique inequalities (3.23) are separated as follows.

   - *Generalized Capacity Constraints*. Let $E(R^k_\ell)$ be the subset of edges covered by a route R^k_ℓ and let $\eta^{k\ell}_{ij}$ be the number of times edge $\{i, j\}$ is traversed by route R^k_ℓ. If R^k_ℓ is a single customer route we set $\eta^{k\ell}_{ij} = 2$; otherwise, we set $\eta^{k\ell}_{ij} = 1$ for each edge $\{i, j\} \in E(R^k_\ell)$. In both cases we set $\eta^{k\ell}_{ij} = 0$, $\forall \{i, j\} \in E \setminus E(R^k_\ell)$. Given the $\overline{LF}$ solution $(\overline{x}, \overline{y})$ of the master and a subset $\overline{P} \subseteq P$, associate with each edge $\{i, j\} \in E$ a weight ω_{ij} computed as follows:

$$\omega_{ij} = \sum_{k \in \overline{P}} \sum_{\ell \in \mathcal{R}^k} \eta^{k\ell}_{ij} \overline{x}^k_\ell, \quad \forall \{i, j\} \in E. \tag{3.39}$$

Note that ω_{ij} can be strictly greater than one (i.e., an edge can be traversed more than once over the days in $\overline{P}$). It is quite simple to notice that any feasible PVRP solution must satisfy the following constraints:

$$\sum_{\{i,j\}\in\delta(S)} \omega_{ij} \geqslant 2 \left\lceil \sum_{i\in S} \hat{q}_i/Q \right\rceil, \quad \forall S \in \mathcal{S}, \tag{3.40}$$

where $\delta(S) = \{\{i,j\} \in E : i \in S, j \notin S \text{ or } i \notin S, j \in S\}$ and the values $\hat{q}_i$, $\forall i \in V$, are computed as described in Section 3.3.3. It can be shown that if the LF solution $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ violates inequalities (3.40) for some S, then it also violates inequalities (3.22) for the same S. Therefore, given a subset $\overline{P}$ we add to $\overline{\mathcal{S}}$ all sets S corresponding to a violated inequality (3.40). If $f_i = 1, \forall i \in V$, inequalities (3.40) are separated using the package CVRPSEP (Lysgaard [85]) as in this case $\omega_{ij} \leqslant 1$ for each edge $\{i,j\} \in E$. Otherwise, inequalities (3.40) are separated using an adaptation of the *greedy randomized algorithm* described in Augerat et al. [7] for finding violated CVRP capacity constraints. After having performed a number of preliminary computational experiments using different strategies for selecting the subsets $\overline{P}$, we decided to set $\overline{P} = P$ in the separation of generalized capacity constraints. Moreover, in order to keep our linear program compact the separation procedure is forced to detect at most 70 violated inequalities (3.40).

- *Clique Inequalities*. Let $H(\overline{\mathbf{x}}) = (\mathcal{R}(\overline{\mathbf{x}}), \overline{E}(\overline{\mathbf{x}}))$ be the subgraph of the conflict graph H induced by the $\overline{LF}$ solution $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$, where $\mathcal{R}(\overline{\mathbf{x}}) = \{\ell \in \widehat{\mathcal{R}}^k : \overline{x}^k_\ell > 0, k \in P\}$ and $\overline{x}^k_\ell$ is the weight of vertex $\ell \in \mathcal{R}(\overline{\mathbf{x}})$. Separating inequalities (3.23) that are violated by the current solution $(\overline{\mathbf{x}}, \overline{\mathbf{y}})$ requires to find all the maximal cliques in $H(\overline{\mathbf{x}})$ of weight grater than one. This problem is solved as described in Baldacci et al. [16] using the CLIQUER 1.1 package [see 96]. Moreover, every new clique is lifted by checking if some of the $\hat{h}$ least reduced cost routes in the sets $\widehat{\mathcal{R}}^k$, $\forall k \in P$, can be used to extend the clique. The resulting lifted clique inequality is then added to the set $\overline{\mathcal{C}}$. In our computational results we set $\hat{h} = 2,000$.

4. If $\mathcal{N}^k = \emptyset$, $\forall k \in P$, and no violated constraints (3.22) and (3.23) are found, then CG terminates; otherwise, the master problem is updated by setting $\overline{\mathcal{R}} = \overline{\mathcal{R}} \cup \mathcal{N}$ and by adding to the master problem all violated constraints found at step 3.

In the following we denote by LCG the cost of the final $\overline{LF}$ solution achieved by CG.

3.8. Computational results

This section presents computational results for the exact method described in Section 3.4 and the bounding procedures described in sections 3.5, 3.6 and 3.7.

The algorithms described in this chapter were coded in Fortran 77, compiled with Intel Visual Fortran 10.1 compiler and linked with the C source codes of the packages CVRPSEP [see 85] and CLIQUER [see 96]. CPLEX 11.0 was used as the LP solver in procedure CG and as the integer programming solver in the exact method. The experiments were performed on a Fujitsu Siemens Primergy TX200S3 server (Intel Xeon E5310 processor at 1.6 GHz with 8 Gb of RAM).

We considered two sets of instances. The first set is composed of instances from the literature and the second one contains randomly generated instances simulating TPVRP problems arising in real world application sectors such as the beverage and food industries. In the following we give a detailed description of the two classes of instances.

(a) *PVRP instances from the literature*. We considered the entire set of PVRP test instances from the literature involving up to 153 customers. The instances were proposed by Christofides and Beasley [31], Russel and Igo [110], Russel and Gribbin [109] and Chao et al. [30]. These instances are publicly available at `http://neumann.hec.ca/chairedistributique/data/pvrp/` and the best upper bounds were taken from Hemmelmayr et al. [70]. The relevant data for these instances are summarized in Table 3.1 that shows under column m the maximum number of vehicles available in each day of the period and under column $z(UB)$ the best upper bound known in the literature.

(b) *TPVRP instances*. We derived twenty TPVRP instances from five CVRP instances, namely E-n51-k5, E-n76-k10, E-n101-k8, M-n121-k7, M-n151-k12 and M-n200-k16 (available at `http://branchandcut.org/VRP/data`), involving up to 199 customers. For each of the five CVRP instances we generated four TPVRP instances with the same vertex coordinates and customer demands of the original CVRP instance but choosing the vehicle capacity Q in the set $\{80, 100, 140, 160\}$. The data for the other instances were defined as follows.

i) p was set equal to five and the frequency f_i of every customer i was set equal to one.

ii) The set of n customers was partitioned into four subsets $S_1 = \{1, \ldots, \alpha_1\}$, $S_2 = \{\alpha_1 + 1, \ldots, \alpha_2\}$, $S_3 = \{\alpha_2 + 1, \ldots, \alpha_3\}$ and $S_4 = \{\alpha_3 + 1, \ldots, n\}$, where $\alpha_1 = \lfloor 0.15\ n \rfloor$, $\alpha_2 = \lfloor 0.30\ n \rfloor$ and $\alpha_3 = \lfloor 0.60\ n \rfloor$. For each customer i the day-window $[e_i, l_i]$ has a maximum width of three days and was defined as follows: (a) $e_i = l_i = 1$, if $i \in S_1$; (b) $e_i = l_i = \alpha$, where α is randomly chosen from the set $\{2, 3, \ldots, p\}$, if $i \in S_2$; (c) $e_i = 1$ and $l_i = e_i + \alpha$, where α is randomly chosen from the set $\{1, 2\}$, if $i \in S_3$; (d) $e_i = \beta$ and $l_i = e_i + \alpha$, where β and α are randomly chosen from the sets $\{2, 3, \ldots, p - 1\}$ and $\{0, 1\}$, respectively, if $i \in S_4$. Customer subsets S_1 and S_2 contain customers that must be visited on a specific day of the period: customers in S_1 must be visited on the first day of the period while customers in S_2 must be visited on a successive day. Both customer subsets S_3 and S_4 contain customers having a width of the day-window greater than one. For customers in S_3 the initial day of the day-window is the first day of the period, whereas for customers in S_4 the initial day of the day-window is a day successive to the first day of the period.

iii) Let $Q_1 = \sum_{i \in S_1} q_i$ and $Q_2 = \sum_{i \in S_3} q_i$ be the total demand of the customers in the sets S_1 and S_2, respectively. Notice that customers in S_1 must be visited on day 1 of the day period. Therefore, the number of

Table 3.1: Data of the PVRP instances from the literature

Name	$\|V\|$	n	$\|V^1\|$	$\|V^2\|$	p	Q	m	z(UB)	Name	$\|V\|$	n	$\|V^1\|$	$\|V^2\|$	p	Q	m	z(UB)
p01	52	51	51	0	2	160	3	524.61	p17	41	40	16	24	4	20	4	1,597.75
p02	51	50	17	33	5	160	3	1,322.87	p18	77	76	32	44	4	30	4	3,136.69
p03	51	50	50	0	5	160	1	524.61	p19	113	112	48	64	4	40	4	4,834.34
p04	76	75	75	0	2	140	5	835.26	p21	61	60	24	36	4	20	6	2,170.61
p05	76	75	30	45	5	140	6	2,027.99	p22	115	114	48	66	4	30	6	4,193.95
p06	76	75	75	0	10	140	1	835.45	p24	52	51	36	15	6	20	3	3,687.46
p07	101	100	100	0	2	200	4	826.14	p25	52	51	36	15	6	20	3	3,777.15
p08	101	100	40	60	5	200	5	2,034.15	p26	52	51	36	15	6	20	3	3,795.32
p09	101	100	100	0	8	200	1	826.14	p27	103	102	72	30	6	20	6	21,956.00
p10	101	100	40	60	5	200	4	1,593.45	p28	103	102	72	30	6	20	6	22,305.34
p11	140	139	103	36	5	235	4	779.06	p29	103	102	72	30	6	20	6	22,639.85
p14	21	20	8	12	4	20	2	954.81	p30	154	153	108	45	6	20	9	74,464.26
p15	39	38	16	22	4	30	2	1,862.63	p31	154	153	108	45	6	20	9	76,552.25
p16	57	56	24	32	4	40	2	2,875.24	p32	154	153	108	45	6	20	9	78,072.88

vehicles available on each day of the period was set equal to $m = \lceil (Q_1 + 0.25Q_2)/Q \rceil + 1$.

iv) The edge costs $\hat{d}_{ij}$ were computed as $\hat{d}_{ij} = \lfloor euc_{ij} + 0.5 \rfloor$, $\forall \{i, j\} \in E$, where euc_{ij} is the Euclidean distance between vertices i and j.

v) The service cost $\tau_i(k)$ for visiting customer $i \in V$ on day $k \in [e_i, l_i]$ was computed as follows:

$$\tau_i(k) = \begin{cases} 0, & \text{if } k = e_i, \\ (1 + 0.5(k - e_i - 1))q_i(2euc_{0i}/Q), & \text{if } e_i < k \leqslant l_i. \end{cases} \tag{3.41}$$

In order to test the effectiveness of the service cost functions in improving the customer service quality we also considered the case where $\tau_i(k) = 0$, $\forall i \in V$, $\forall k \in [e_i, l_i]$.

In computing the lower bounds for all the PVRP instances proposed in the lit-

erature we executed in sequence procedures H^1, H^2 and CG as all these instances satisfy conditions (a) and (b) described in Section 3.4.1. The new TPVRP instances do not satisfy these conditions, therefore we used the sequence H^3, H^4 and CG to compute the lower bounds. The upper bounds of the new TPVRP instances were computed using our implementation of the tabu search heuristic algorithm proposed by Cordeau et al. [38].

Based on the results of several preliminary experiments to identify good parameter settings for our method we decided to use the following settings in solving all test problems:

- in H^1, $Maxt1 = 300$;
- in H^2, $Maxt2 = 25$, $Maxt3 = 400$, $\Delta^{min} = 5,000$ and $\Delta^a = 200$;
- in H^3, $Maxt1 = 300$ and $\epsilon = 2.0$; in H^4, $Maxt2 = 25$, $Maxt3 = 200$, $\epsilon = 1.5$, $\Delta^{min} = 5,000$, $\Delta^a = 200$;
- in CG, $\Delta^{min} = 700$ and $\Delta^a = 200$;
- in order to avoid out-of-memory errors we impose that the size of the final route set does not exceed the limit of $300,000$ routes (i.e., $\sum_{k\in P} |\widehat{\mathcal{R}}^k| \leqslant 300,000$).

Tables 3.2, 3.3 and 3.4 of this section report the following columns:

$z(UB)$: best upper bound value available in the literature;

z^* : cost of the optimal solution or cost of the best solution found by the exact method; values printed in bold indicate that the solution cost found is less than the best known upper bound;

%LHx : percentage ratio of lower bound LHx, $x = 1, 2, 3, 4$, computed as 100.0 LHx/z^*;

t_{H^x} : time in seconds spent by the bounding procedure H^x, $x = 1, 2, 3, 4$;

%LCG : percentage ratio of the final lower bound LCG computed as 100.0 LCG/z^*;

t_{CG} : time in seconds spent by procedure CG;

t_{BP} : total time in seconds spent by the sequence of bounding procedures used for computing the final lower bound LCG. In Table 3.2, $t_{BP} = t_{H^1} + t_{H^2} + t_{CG}$, while in Tables 3.3 and 3.4, $t_{BP} = t_{H^3} + t_{H^4} + t_{CG}$;

$|\mathcal{R}|$: total number of routes in the final route subsets $\widehat{\mathcal{R}}^k$, $\forall k \in P$, i.e., $|\mathcal{R}| = \sum_{k \in P} |\mathcal{R}^k|$;

t_{CPX} : total computing time in seconds spent by CPLEX to solve problem $\hat{F}$;

t_{EM} : total computing time in seconds of the exact method; t_{EM} is the sum of t_{BP}, t_{CPX} and of the time spent by procedure GENROUTE for generating the final route subsets $\widehat{\mathcal{R}}_k$, $\forall k \in P$;

For the PVRP instances and the TPVRP instances a time limit of 14,400 seconds and of 7,200 seconds was imposed to CPLEX, respectively. The last line of tables 3.2, 3.3 and 3.4 reports the average percentage ratios and the average running times in seconds of each bounding procedure over all instances.

Tables 3.2, 3.3 and 3.4 report the lower bounds obtained by the bounding procedures and the results of the exact method for the two sets of instances considered. In particular, tables 3.4 and 3.3 refer to the new TPVRP instances with and without service costs, respectively.

Table 3.2 shows that the exact method is able to solve to optimality 14 out of the 28 PVRP instances from the literature and to improve the best known upper bound of problems p06, p27, p28, p29 and p31. Instance p11 is particularly difficult and procedure H^2 failed to compute the lower bound within the memory limit imposed to GENROUTE.

Tables 3.3 and 3.4 show that the lower bound achieved on both the variants of the TPVRP is on average within one percent of optimality. The exact method was able to solve to optimality 28 out of the 40 instances. All the TPVRP instances involving up to 100 customers were solved optimality.

Finally, Table 3.5 compares the best solutions found by the exact algorithm with and without service costs. Columns z^r and z^s of this table show the total

routing cost and the total service cost, respectively, whereas column z^{r+s} reports the sum $z^r + z^s$. The purpose of this table is to compare the TPVRP solutions achieved when ignoring service costs in the model (columns under the heading "without service costs") with those obtained when accounting for service costs in the objective function (columns under the heading "with service costs"). To this end, column $z^r + z^s$, under the heading "without service costs", reports the cost of the best solutions found by the algorithm ignoring service costs plus the total service cost that would be incurred when operating that solution. Moreover, column #day reports the total number of days of delay with respect to the initial day of customers' day-windows. The last line of the table reports for each column the sum of the entries over rows.

The table clearly shows that the inclusion of service costs induces solutions of better quality with respect to the customer service level at the expense of a marginal increase in the routing costs. Indeed, when taking into account service costs in the model, a reduction of 33% on the total number of days of delay could be achieved with an extra routing cost of about 3%.

It is also worth mentioning that the exact method proposed in this chapter, when applied to CVRP and MDVRP instances, produces the same computational results as those reported in Baldacci et al. [16] and Baldacci and Mingozzi [13].

3.9. Summary

We presented an exact algorithm for the Period Routing Problem (PVRP) for which neither lower bounds nor exact algorithms have been proposed in the literature so far. The exact algorithm is based on a set partitioning-like formulation of the problem and uses five types of bounding procedures based on different relaxations of the PVRP. The bounding procedures reduce the number of variables in the formulation so that the resulting problem can be practically solved by an integer programming solver.

We also considered a special case of the PVRP, called Tactical Planning VRP

(TPVRP), arising in food and beverage distribution systems and in field force planning.

We reported computational results on both a set of PVRP test instances from the literature and a new set of TPVRP test instances. The results obtained show that the proposed exact method is able to achieve, on average, lower bounds within one percent of optimality. The algorithm was able to solve to optimality several PVRP instances from the literature for the first time and to improve some of the best known upper bounds. Moreover, TPVRP instances with up to 100 customers could be consistently solved to optimality.

Table 3.2: Computational results for the PVRP instances from the literature

					Bounding Procedure										Exact Method		
Name	|V|	p	z(UB)	z^*	LH1	%LH1	t_{H^1}	LH2	%LH2	t_{H^2}	LCG	%LCG	t_{CG}	t_{BP}	$\|\mathcal{R}\|$	t_{CPX}	t_{EM}
p01	52	2	524.61	524.61	516.69	98.5	2.9	520.91	99.3	1.4	524.61	100.0	0.1	4.4	0	0.0	4.4
p02	51	5	1,322.87	1,322.87 (a)	1,248.63	94.4	4.0	1,299.48	98.2	3.8	1307.33	98.8	390.4	398.2	300,000	14,400.0	14,811.0
p03	51	5	524.61	524.61	516.69	98.5	2.9	520.94	99.3	1.6	524.61	100.0	0.1	4.7	0	0.0	4.7
p04	76	2	835.26	835.26	815.52	97.6	4.8	816.99	97.8	2.9	830.87	99.5	73.7	81.3	19,399	201.3	335.6
p05	76	5	2,027.99	2,027.99 (a)	1,912.88	94.3	5.3	1,995.93	98.4	3.2	2005.53	98.9	149.2	157.7	300,000	14,400.0	14,568.8
p06	76	10	835.45	**835.26**	815.50	97.6	4.8	816.98	97.8	2.7	830.83	99.5	73.7	81.2	27,644	509.7	670.3
p07	101	2	826.14	826.14 (a)	792.79	96.0	18.6	797.88	96.6	8.5	804.39	97.4	367.2	394.3	300,000	14,400.0	14,921.4
p08	101	5	2,034.15	2,034.15 (a)	1,934.74	95.1	21.9	1,994.00	98.0	473.5	2000.10	98.3	2109.9	2,605.3	300,000	14,400.0	17,111.5
p09	101	8	826.14	826.14 (a)	792.79	96.0	18.6	797.88	96.6	8.5	804.39	97.4	367.2	394.3	300,000	14,400.0	14,921.4
p10	101	5	1,593.45	1,593.45 (a)	1,533.53	96.2	18.3	1,546.98	97.1	9.9	1556.43	97.7	1190.7	1,218.9	300,000	14,400.0	15,656.7
p11	140	5	779.06	779.06 (a)	685.47	88.0	40.0	- (b)	-	-	-	-	-	40.0	-	-	40.0
p14	21	4	954.81	954.81	853.05	89.3	0.1	954.81	100.0	0.7	954.81	100.0	0.0	0.7	0	0.0	0.7
p15	39	4	1,862.63	1,862.63	1,773.97	95.2	0.3	1,862.60	100.0	3.3	1862.60	100.0	0.0	3.5	0	0.0	3.5
p16	57	4	2,875.24	2,875.24	2,791.08	97.1	0.7	2,875.24	100.0	6.1	2875.24	100.0	0.0	6.8	0	0.0	6.8
p17	41	4	1,597.75	1,597.75	1,474.02	92.3	0.2	1,594.22	99.8	2.1	1597.75	100.0	0.3	2.6	0	0.0	2.6
p18	77	4	3,136.69	3,136.69 (a)	2,873.61	91.6	0.8	3,088.75	98.5	26.1	3088.90	98.5	433.8	460.7	300,000	14,400.0	15,226.8
p19	113	4	4,834.34	4,834.34 (a)	4,567.15	94.5	2.1	4,766.78	98.6	1529.5	4766.91	98.6	120.1	1,651.7	300,000	14,400.0	16,124.4
p21	61	4	2,170.61	2,170.61	2,008.47	92.5	0.3	2,162.61	99.6	2.0	2169.29	99.9	142.8	145.1	27,005	17.3	163.5
p22	115	4	4,193.95	4,193.95 (a)	3,850.36	91.8	1.6	4,160.31	99.2	322.7	4169.73	99.4	15.0	339.3	300,000	14,400.0	14,829.5
p24	52	6	3,687.46	3,687.46	3,171.85	86.0	0.3	3,680.58	99.8	1.6	3680.63	99.8	0.5	2.3	5,628	59.2	61.6
p25	52	6	3,777.15	3,777.15	3,334.85	88.3	0.2	3,753.36	99.4	2.3	3753.38	99.4	1.7	4.2	18,390	218.9	223.2
p26	52	6	3,795.32	3,795.32	3,437.07	90.6	0.2	3,792.76	99.9	1.5	3792.80	99.9	0.1	1.9	1,500	1.8	3.8
p27	103	6	21,956.00	**21,912.85** (a)	16,989.43	77.5	0.8	21,786.26	99.4	9.7	21787.50	99.4	71.1	81.5	300,000	14,400.0	14,519.5
p28	103	6	22,305.34	**22,246.69**	17,798.65	80.0	0.7	22,210.97	99.8	6.0	22211.10	99.8	5.8	12.5	141,246	12,334.7	12,348.8
p29	103	6	22,639.85	**22,543.75**	19,370.57	85.9	0.8	22,531.65	99.9	5.9	22531.73	99.9	1.5	8.2	7,680	6,450.1	6,458.8
p30	154	6	74,464.26	74,464.26 (a)	49,586.02	66.6	1.5	73,236.52	98.4	86.4	73248.41	98.4	699.1	787.0	300,000	14,400.0	15,218.0
p31	154	6	76,552.25	**76,322.04** (a)	59,579.98	78.1	1.5	75,756.97	99.3	138.8	75757.34	99.3	46.6	186.8	300,000	14,400.0	14,598.3
p32	154	6	78,072.88	78,072.88 (a)	57,841.64	74.1	1.5	77,184.59	98.9	82.8	77185.20	98.9	38.2	122.5	300,000	14,400.0	14,537.4
						90.1	5.6		98.9	101.6		99.2	233.3	328.5			

(a): optimality not proved

(b): H^2 terminates prematurely due to memory overflow in procedure GENROUTE

Table 3.3: Computational results for the TPVRP instances without service costs

						Bounding Procedure										Exact Method		
Name	$\|V\|$	Q	m	$z(UB)$	z^*	LH3	%LH3	t_{H^3}	LH4	%LH4	t_{H^4}	LCG	%LCG	t_{CG}	t_{BP}	$\|\mathcal{R}\|$	t_{CPX}	t_{EM}
E-n51-k5	51	80	4	899.0	899.0	849.7	94.5	2.4	889.7	99.0	2.9	895.5	99.6	0.6	5.8	8,483	0.6	6.5
E-n76-k10	76	80	6	1,369.0	1,369.0	1,331.7	97.3	4.0	1,357.5	99.2	3.6	1,361.0	99.4	0.6	8.1	10,146	0.8	9.1
E-n101-k8	101	80	5	1,527.0	1,527.0	1,461.2	95.7	6.9	1,506.2	98.6	8.0	1,520.8	99.6	16.2	31.2	20,454	9.3	93.9
M-n151-k12	151	80	7	2,131.6	**2,121.0**	2,065.7	97.4	15.7	2,096.9	98.9	9.9	2,109.6	99.5	20.0	45.6	43,396	287.1	394.6
M-n200-k16	200	80	9	2,804.8	**2,777.0**	2,708.2	97.5	27.6	2,753.1	99.1	10.9	2,767.8	99.7	36.9	75.4	56,570	191.1	464.2
E-n51-k5	51	100	3	835.0	835.0	768.0	92.0	3.4	823.4	98.6	4.7	833.7	99.8	0.8	8.9	8,784	0.1	10.0
E-n76-k10	76	100	5	1,234.0	1,234.0	1,169.2	94.8	5.7	1,216.2	98.6	7.5	1,228.6	99.6	4.3	17.5	13,152	1.8	30.4
E-n101-k8	101	100	5	1,341.0	1,341.0	1,293.0	96.4	10.4	1,324.3	98.8	9.5	1,336.6	99.7	28.0	47.9	20,722	9.1	288.2
M-n151-k12	151	100	6	1,856.0	1,856.0	1,801.5	97.1	24.2	1,837.1	99.0	10.2	1,847.7	99.6	35.0	69.4	50,588	246.9	430.3
M-n200-k16	200	100	8	2,449.2	**2,413.0** (a)	2,316.7	96.0	38.7	2,371.1	98.3	12.5	2,384.9	98.8	38.2	89.4	300,000	7,200.0	7,451.2
E-n51-k5	51	140	3	784.0	784.0	684.3	87.3	6.3	762.9	97.3	11.0	783.1	99.9	497.3	514.6	16,683	0.6	929.8
E-n76-k10	76	140	4	1,070.0	1,070.0	1,009.5	94.3	11.5	1,053.2	98.4	11.0	1,063.2	99.4	15.5	38.0	19,632	6.0	58.8
E-n101-k8	101	140	4	1,188.0	1,188.0	1,104.1	92.9	22.1	1,170.3	98.5	11.4	1,181.0	99.4	313.9	347.5	31,296	78.7	3,154.6
M-n151-k12	151	140	5	1,629.7	**1,612.0** (a)	1,527.7	94.8	42.1	1,574.9	97.7	21.3	1,588.1	98.5	393.3	456.6	300,000	7,200.0	8,104.3
M-n200-k16	200	140	6	2,035.1	**2,027.0** (a)	1,910.9	94.3	63.9	1,976.9	97.5	40.9	1,995.5	98.4	546.8	651.6	300,000	7,200.0	8,374.9
E-n51-k5	51	160	3	762.0	762.0	662.4	86.9	8.3	745.1	97.8	10.4	761.5	99.9	571.8	590.5	16,741	0.6	1,292.8
E-n76-k10	76	160	4	1,022.0	1,022.0	961.7	94.1	15.8	1,006.7	98.5	11.3	1,015.6	99.4	36.1	63.3	19,214	5.0	125.8
E-n101-k8	101	160	3	1,153.6	**1,141.0**	1,052.1	92.2	28.8	1,130.3	99.1	20.8	1,136.7	99.6	553.2	602.8	23,099	9.9	2,006.7
M-n151-k12	151	160	4	1,550.1	**1,531.0** (a)	1,446.3	94.5	51.9	1,500.2	98.0	54.2	1,511.8	98.7	570.6	676.7	300,000	7,200.0	8,480.0
M-n200-k16	200	160	5	1,958.6	**1,946.0** (a)	1,793.9	92.2	77.0	1,862.2	95.7	100.2	1,878.0	96.5	620.3	797.4	300,000	7,200.0	8,593.8
							94.1	23.3		98.3	18.6		99.3	215.0	256.9			

(a): optimality not proved

Table 3.4: Computational results for the TPVRP instances with service costs

						Bounding Procedure										Exact Method		
Name	$\|V\|$	Q	m	$z(UB)$	z^*	LH3	%LH3	t_{H^3}	LH4	%LH4	t_{H^4}	LCG	%LCG	t_{CG}	t_{BP}	$\|\mathcal{R}\|$	t_{CPX}	t_{EM}
E-n51-k5	51	80	4	989.0	988.98	943.0	95.4	2.4	987.1	99.8	2.8	989.0	100.0	1.5	6.7	0	0.0	6.7
E-n76-k10	76	80	6	1,574.1	1,574.10	1,463.2	93.0	4.1	1,565.7	99.5	3.7	1,573.4	100.0	0.7	8.4	0	0.0	8.4
E-n101-k8	101	80	5	1,791.8	1,791.78	1,602.9	89.5	7.1	1,764.8	98.5	9.3	1,782.9	99.5	18.1	34.4	20,752	12.9	58.0
M-n151-k12	151	80	7	2,648.7	2,648.73	2,287.1	86.3	16.0	2,613.4	98.7	11.2	2,633.7	99.4	51.1	78.3	73,913	1,094.3	1,211.9
M-n200-k16	200	80	9	3,553.1	**3,507.50** (a)	2,924.3	83.4	28.3	3,459.1	98.6	13.3	3,478.1	99.2	40.5	82.0	300,000	7,200.0	10,000.6
E-n51-k5	51	100	3	938.0	937.96	857.8	91.5	3.5	932.1	99.4	3.8	938.0	100.0	0.6	7.9	0	0.0	7.9
E-n76-k10	76	100	5	1,418.7	1,418.71	1,313.1	92.6	5.8	1,403.6	98.9	6.8	1,415.5	99.8	2.1	14.7	11,384	0.7	15.4
E-n101-k8	101	100	5	1,517.9	1,517.89	1,423.9	93.8	10.6	1,501.8	98.9	9.0	1,511.4	99.6	14.7	34.4	20,996	13.4	54.8
M-n151-k12	151	100	6	2,259.5	**2,246.07** (a)	2,013.5	89.6	24.7	2,209.9	98.4	11.9	2,224.9	99.1	48.9	85.5	300,000	7,200.0	7,851.8
M-n200-k16	200	100	8	2,906.0	**2,877.23** (a)	2,520.5	87.6	39.4	2,799.2	97.3	13.5	2,815.5	97.9	44.0	96.9	300,000	7,200.0	7,953.7
E-n51-k5	51	140	3	869.2	869.24	753.9	86.7	6.4	843.1	97.0	10.9	866.7	99.7	481.2	498.5	16,744	1.2	639.3
E-n76-k10	76	140	4	1,203.9	1,203.91	1,130.6	93.9	11.6	1,190.6	98.9	10.8	1,198.4	99.5	3.2	25.5	14,541	1.9	27.5
E-n101-k8	101	140	4	1,330.7	1,330.74	1,223.2	91.9	22.2	1,315.9	98.9	13.9	1,328.7	99.8	583.2	619.3	20,892	1.8	956.1
M-n151-k12	151	140	5	1,876.4	**1,867.06** (a)	1,706.9	91.4	42.7	1,818.5	97.4	47.7	1,831.6	98.1	162.3	252.8	300,000	7,200.0	8,123.0
M-n200-k16	200	140	6	2,317.5	2,317.54 (a)	2,092.0	90.3	64.4	2,284.4	98.6	29.0	2,289.9	98.8	711.3	804.7	300,000	7,200.0	9,374.6
E-n51-k5	51	160	3	839.1	839.05	721.9	86.0	8.4	812.8	96.9	10.0	835.6	99.6	605.8	624.2	17,003	2.8	1012.1
E-n76-k10	76	160	4	1,151.7	1,151.66	1,077.7	93.6	16.0	1,141.7	99.1	11.6	1,151.7	100.0	5.7	33.3	0	0.0	33.3
E-n101-k8	101	160	3	1,292.4	1,292.41	1,166.6	90.3	28.9	1,270.2	98.3	22.3	1,278.6	98.9	1,069.7	1,120.9	267,839	6,805.2	10,666.0
M-n151-k12	151	160	4	1,772.3	1,772.27 (a)	1,616.0	91.2	52.8	1,734.6	97.9	90.4	1,740.7	98.2	786.0	929.2	300,000	7,200.0	8,774.6
M-n200-k16	200	160	5	2,241.5	2,241.47 (a)	1,958.9	87.4	78.3	2,158.4	96.3	101.3	2,172.0	96.9	550.8	730.3	300,000	7,200.0	8,844.6
							90.3	23.7		98.4	21.7		99.2	259.1	304.4			

(a): optimality not proved

Table 3.5: Comparison of the solutions obtained for the TPVRP instances

				without service costs				with service costs			
Name	|V|	Q	m	z^r	z^s	z^{r+s}	#days	z^r	z^s	z^{r+s}	#days
E-n51-k5	51	80	4	899.00	192.00	1,091.00	25	934.00	54.98	988.98	9
E-n76-k10	76	80	6	1,369.00	340.25	1,709.25	37	1,451.00	123.10	1,574.10	19
E-n101-k8	101	80	5	1,527.00	438.33	1,965.33	45	1,627.01	164.78	1,791.78	26
M-n151-k12	151	80	7	2,121.00	627.43	2,748.43	66	2,183.01	465.73	2,648.73	61
M-n200-k16	200	80	9	2,777.00	892.05	3,669.05	106	2,841.00	666.50	3,507.50	93
E-n51-k5	51	100	3	835.00	178.33	1,013.33	29	874.00	63.96	937.96	14
E-n76-k10	76	100	5	1,234.00	294.53	1,528.53	37	1,271.00	147.71	1,418.71	22
E-n101-k8	101	100	5	1,341.00	315.50	1,656.50	45	1,408.00	109.89	1,517.89	23
M-n151-k12	151	100	6	1,856.00	540.85	2,396.85	69	1,918.00	328.07	2,246.07	54
M-n200-k16	200	100	8	2,413.00	565.27	2,978.27	80	2,550.00	327.23	2,877.23	57
E-n51-k5	51	140	3	784.00	114.79	898.79	25	803.00	66.24	869.24	15
E-n76-k10	76	140	4	1,070.00	168.12	1,238.12	32	1,087.00	116.91	1,203.91	21
E-n101-k8	101	140	4	1,188.00	266.46	1,454.46	50	1,218.00	112.74	1,330.74	26
M-n151-k12	151	140	5	1,612.00	346.20	1,958.20	69	1,682.00	185.06	1,867.06	39
M-n200-k16	200	140	6	2,027.00	385.18	2,412.18	78	2,080.00	237.54	2,317.54	55
E-n51-k5	51	160	3	762.00	91.41	853.41	23	791.00	48.05	839.05	16
E-n76-k10	76	160	4	1,022.00	165.41	1,187.41	30	1,050.00	101.66	1,151.66	21
E-n101-k8	101	160	3	1,141.00	181.02	1,322.02	39	1,191.00	101.41	1,292.41	26
M-n151-k12	151	160	4	1,531.00	325.44	1,856.44	73	1,575.00	197.27	1,772.27	45
M-n200-k16	200	160	5	1,946.00	396.33	2,342.33	91	2,006.00	235.47	2,241.47	62
				29,455.00	6,824.89		1,049	30,540.01	3,854.29		704

Chapter 4

The pickup and delivery problem with time windows

The Pickup and Delivery Problem with Time Windows (PDPTW) is a generalization of the Vehicle Routing Problem with Time Windows where a set of identical vehicles located at a central depot must be optimally routed to service a set of transportation requests subject to capacity, time window, pairing and precedence constraints. In this chapter, we present an exact algorithm for the PDPTW which is based on a set partitioning formulation with additional cuts. We describe a bounding procedure that finds a near optimal dual solution of the LP-relaxation of the integer formulation by combining two dual ascent heuristics and a column-and-cut generation procedure. The final dual solution is used to generate a reduced problem containing only the routes whose reduced costs are smaller than the gap between a known upper bound and the lower bound achieved. If the resulting problem has moderate size it is solved by an integer programming solver; otherwise, a branch-and-cut-and-price algorithm is used to close the integrality gap. Extensive computational results over the main instances from the literature show that the proposed exact method is faster than the best exact method presented in the literature so far and that it can solve to optimality for the first time several test instances.

4.1. Introduction

The Pickup and Delivery Problem with Time Windows (PDPTW) is the problem of designing a number of routes for a fleet of m identical vehicles stationed at a central depot to service the transportation requests of n customers. Each customer request asks that a given load is transported from a pickup location to an associated delivery location by the same vehicle. Each vehicle route starts and finishes at the depot according to the time window of the depot and services a subset of the customer requests. In order to be feasible each route must be simple and must satisfy the following constraints: (i) the delivery location of each request serviced must be visited after the associated pickup location by the same vehicle (*pairing and precedence constraints*), (ii) the total load carried by the vehicle after leaving any pickup location cannot exceed the vehicle capacity (*capacity constraints*) and (iii) all locations must be visited within their time windows (*time window constraints*). The objective is to design a set of at most m feasible routes of minimum cost so that each customer request is serviced by exactly one route. However, in some applications it is required to minimize first the number of vehicles used and then the route costs.

The PDPTW is a generalization of the Vehicle Routing Problem with Time Windows (VRPTW) in that the VRPTW can be seen as a special case of the PDPTW where the pickup locations of all requests coincide with the depot. Since the PDPTW generalizes the VRPTW it is $\mathcal{NP}$-hard.

There is a variety of practical situations that can be modeled as pickup and delivery problems. Some examples are sealift and airlift of cargo and troops, pickup and delivery for overnight carriers and urban services, less-than truckload transportation and school buses routing and scheduling. In some applications it is required that the total time between a pickup stop and the corresponding delivery stop does not exceed a given threshold. In this case the problem is called the Dial-a-Ride Problem (DARP) and finds important applications in door-to-door transportation services for the elderly and the disabled.

4.1.1 Literature Review

In this section we give an overview of the solution methods presented in the literature for solving the PDPTW considered in this chapter. Several other variants of this problem have been studied. For reviews on different pickup and delivery problems the reader is referred to the works of Savelsbergh and Sol [112], Desaulniers et al. [50] and Cordeau et al. [39]. Recent surveys on pickup and delivery problems containing a classification of the different variants together with a discussion on the corresponding exact, heuristic, and metaheuristic solution methods can be found in Cordeau et al. [40], Gribkovskaia and Laporte [67] and Parragh et al. [100, 101]. As this chapter presents an exact algorithm for the PDPTW we review the latest exact solution methods proposed in the literature for the PDPTW.

Exact algorithms for the PDPTW have been proposed by Dumas et al. [54], Savelsbergh and Sol [112], Lu and Dessouky [84], Ropke et al. [107] and Ropke and Cordeau [106].

The method of Dumas et al. [54] is the first branch-and-price algorithm proposed in the literature for the PDPTW. This algorithm is based on a set partitioning formulation of the problem where each column corresponds to a vehicle route and each constraint is associated with a transportation request. The pricing subproblem used is a *Shortest Path Problem with Time Windows, Capacity, and Pickup and Delivery* that is solved by a dynamic programming algorithm where states correspond to paths starting from the depot. The labels associated with each path allow the algorithm to enforce time window, capacity, pairing and precedence constraints, but do not guarantee that only elementary routes are generated. Different methods for eliminating states within this dynamic programming procedure are also proposed. These methods are based on the concepts of dominance (i.e., avoid generating routes that cannot be part of an optimal solution) and non-post-feasibility (i.e., eliminate paths that cannot be extended to feasible routes). Various preprocessing rules for tightening time windows and removing arcs that cannot be part of a feasible solution are also proposed. More-

over, the algorithm makes use of a partial network, obtained from the original graph by heuristically removing nodes and arcs, to attempt generating negative reduced cost routes while reducing the computing time when solving the pricing problem. The branching strategy adopted by this algorithm chooses a fractional variable corresponding to a route visiting an ordered set of pickups $S = (0, i_1, i_2, \ldots, i_k, i_{k+1} = 0)$ (0 representing the depot), and creates $|S| + 2$ branches. Each branch $h = 1, \ldots, |S| + 2$ imposes that the pickups $i_1, \ldots, i_h$ are visited in the same order as in S but the vertex i_{h+1} cannot be visited immediately after i_h. Two additional branches are also required, imposing respectively that i_1 cannot be visited just after the depot and that all vertices in S must be visited in the same order. This method was tested on instances involving up to 55 requests and proved successful in solving tightly constrained instances with a limited number of requests per route.

Savelsbergh and Sol [112] describe a branch-and-price method based on a set partitioning formulation that differs from that of Dumas et al. [54] in several respects. They use heuristics to solve the pricing subproblem before trying to solve it to optimality and maintain a column pool that contains variables having a reduced cost not greater than a predefined threshold. The pricing problem is solved only when no columns of negative reduced cost can be found in this pool. At the beginning of each iteration the reduced costs of columns in the pool are updated with respect to the current dual solution, and those columns whose reduced cost exceeds the threshold are eliminated. Then, if no negative reduced cost column exists in the pool the pricing problem is solved using heuristics and the new columns are added to the pool. If again no column having negative reduced cost can be found in the pool, then the pricing problem is solved exactly. The branching schema of this algorithm combines two strategies. The first one is the same used by Dumas et al. [54], while the second one is based on a set of assignment variables $\{z_i^k\}$, defined for each request i and vehicle k, that represent the fraction of request i serviced by each vehicle k. This latter strategy chooses a fractional variable z_i^k and creates two branches imposing $z_i^k = 0$ and $z_i^k = 1$,

respectively. Notice that this strategy can be seen as special case of Ryan and Foster's branching schema (Ryan and Foster [111]) since, given two requests i and j, a branch where $z_i^k = 1$ and $z_j^k = 0$ imposes that i and j cannot be visited by the same route. A primal constructive heuristic that is based on the assignment variables $\{z_i^k\}$ is used at each node of the enumerative tree to compute an upper bound on the associated problem. A set of test instances with up to 30 requests are randomly generated to test the effectiveness of the proposed exact algorithm. On 17 out of 40 instances the algorithm is able to solve the problem at the root node, whereas for the remaining instances the integrality gap at the root node is under 1%. All the instances are solved to optimality using in the worst case $55,266$ seconds of computing time of an IBM/RS6000, model 500.

Lu and Dessouky [84] consider a variant of the PDPTW where the vehicle fleet can be heterogeneous and vehicles can be based at different depots. They specifically address the case where time windows are not tight and present a branch-and-cut algorithm based on a two-index formulation. This formulation involves a polynomial number of constraints but uses in addition to the usual flow variables a set of binary precedence variables. These variables specify for each vertex pair i, j whether vertex i must precede j in a route, or viceversa, and are used to enforce pairing and precedence constraints. Using precedence variables, four classes of valid inequalities are derived to strengthen the LP-relaxation of the mathematical formulation. Even if the number of constraints in the proposed formulation is polynomial, the branch-and-cut algorithm is started with a limited number of constraints from the original formulation, and violated constraints are then added at each node together with violated valid inequalities. The algorithm uses two branching strategies. The first strategy branches on the flow variables. The second one relies on precedence variables and imposes if a given pair of vertices has to be adjacent or not in a route. Using precedence variables, the latter strategy computes for each vertex i the number S_i of vertices preceding i in the route. S_i represents the position of vertex i within the route. Then, it chooses a vertex pair i, j such that $|S_i - S_j|$ is minimum and creates two branches imposing

$|S_i - S_j| \leqslant 1$ and $|S_i - S_j| > 1$, respectively. Values $\{S_i\}$ are also used at each node of the enumerative tree by a heuristic procedure to compute an upper bound on the problem. The algorithm is tested on two classes of instances. A first class is derived by the Solomon data set for the VRP, and a second class is randomly generated using a method similar to that of Savelsbergh and Sol [112]. In both data sets the vehicle fleet is homogeneous. Computational results show that the algorithm can solve to optimality instances with up to 5 vehicles and 25 requests within 3 hours of computing time on a SUN Fire 4800 system (12 900-MHZ CPUs).

Ropke et al. [107] present a branch-and-cut algorithm that is an improved version of the one proposed by Cordeau [36] for the DARP. This paper describes two different formulations based on two-index variables and involve an exponential number of constraints. Several families of valid inequalities are also used to strengthen the LP-relaxation of such formulations. The second formulation provides slight better results and is more compact in terms of the number of variables, but imposes capacity, time window, pairing and precedence constraints using three families of inequalities involving an exponential number of constraints. Capacity constraints are enforced using *rounded capacity inequalities* [see e.g. 95] that impose a lower bound on the number of times a vehicle must enter and leave a set of vertices in order to service all the corresponding nodes. Time window constraints are imposed by defining the set $\mathcal{P}$ of all paths that are infeasible with respect to time windows. Then, for any path $P \in \mathcal{P}$ involving a set of arcs $A(P)$, an *infeasible path elimination constraint* [see e.g. 5] is imposed stating that at most $|A(P)| - 1$ arcs in $A(P)$ can be in solution. Finally, a set of *precedence constraints* [see 108] are used to impose pairing and precedence constraints by stating that at least one arc must enter each vertex set S that contains the starting depot and at least one delivery whose corresponding pickup is not in S. The algorithm also uses additional classes of valid inequalities called *successor* and *predecessor inequalities* and *lifted generalized order constraints* that were introduced by Cordeau [36] for the DARP. Three new classes of inequalities for the PDPTW are introduced in this paper: *strengthened capacity constraints*, obtained by strengthening rounded

capacity constraints and *strengthened infeasible path constraints* and *fork constraints*, both expressing different conditions to forbid infeasible paths. Finally the set of *reachability constraints* [see 86] is adapted to the PDPTW by extending the sets of conflicting nodes taking into account precedence and capacity constraints. The proposed branch-and-cut algorithm is evaluated on two classes of instances containing PDPTW problems involving up to 75 requests and DARP problems with up to 96 requests. The computational results show that fork and reachability constraints are the most useful in terms of the improvement of the integrality gap at the root node. The algorithm can optimally solve PDPTW instances with up to 75 requests and outperforms the algorithm of Cordeau [36] on the DARP instances.

Ropke and Cordeau [106] present a new branch-and-cut-and-price algorithm for the PDPTW in which lower bounds are computed by solving through column generation the linear programming relaxation of a set partitioning formulation strengthened by valid inequalities. Within the column generation algorithm two pricing subproblems are experimented, based on elementary and non-elementary shortest path computations. The first pricing problem, called SP1, corresponds to an *Elementary Shortest Path Problem with Time Windows, Capacity, and Pickup and Delivery*; while the second one, called SP2, is the same used by Dumas et al. [54]. SP2 is faster than SP1 but produces worst lower bounds because it generates routes that may contain cycles. The paper also discusses the use of different classes of valid inequalities introduced for the two-index and three-index formulations by Ropke et al. [107] and Cordeau [36] and their usefulness when added to the set partitioning formulation. A method for modifying the cost matrix used in the pricing algorithm is also proposed to allow the use of the dominance criteria within the pricing algorithms after adding valid inequalities to the master problem. The authors prove that strengthened precedence constraints, fork inequalities and reachability inequalities are implied by the LP-relaxation of the set partitioning formulation when using SP1 as the pricing problem. Therefore these inequalities are not embedded in the proposed branch-and-cut-algorithm when using SP1. On the other hand, strengthened infeasible path inequalities,

rounded capacity inequalities (here strengthened with respect to those used by Ropke et al. [107]) and *2-path inequalities* (proposed by Kohl et al. [78] for the VRPTW and extended here to the PDPTW) are not redundant in the set partitioning formulation using SP1 and are used by this algorithm. The proposed branch-and-cut-algorithm uses different combinations of heuristics in order to reduce the computational effort for solving the pricing subproblem: only when none of the heuristics is able to provide negative reduced cost routes the exact dynamic programming procedure is executed. Two different branching strategies are employed. The first one computes the number κ of routes leaving the depot and then creates two branches imposing $m = \lceil \kappa \rceil$ and $m = \lfloor \kappa \rfloor$, respectively. If κ is integral, however, the current solution may still be fractional. If this is the case, a second branching strategy is used that branches on the outflow $\delta^+(S)$ of a set S of vertices (i.e. on the number of routes leaving S), imposing $\delta^+(S) = \lceil \delta^+(S) \rceil$ or $\delta^+(S) = \lfloor \delta^+(S) \rfloor$. Computational experiments show that the 2-path cuts are the most successful of the valid inequalities tested and most often, using the pricing problem SP1, yield better results. A comparison with the method of Ropke et al. [107] shows that the proposed algorithm outperforms the branch-and-cut method of Ropke et al. [107] and that it can solve three instances with 500 requests having very tight time windows. At our knowledge the algorithm of Ropke and Cordeau [106] is currently the best solution method available in the literature for the PDPTW in terms of the size of the instances that can be solved to optimality .

It is worth mentioning that, as opposed to the vehicle routing problem, there are few clearly defined benchmark problems for the pickup and delivery problem. Indeed, most of the exact methods presented in the literature have been tested on different data sets that are not publicly available, making it difficult to compare the effectiveness of the proposed algorithms. The only instances that are publicly available are those used by Ropke and Cordeau [106] which correspond to two data sets described in Section 4.7.

4.1.2 Contributions

In this chapter we present an exact algorithm for solving the PDPTW with two different objective functions. The first objective asks to minimize the total routing cost, whereas the second asks to minimize the sum of the fixed costs of the vehicles used and of the routing costs. The exact method is based on a set partitioning-like formulation F of the PDPTW where each column corresponds to a route. We propose a bounding method that computes a solution of the dual of the LP-relaxation of F using in sequence three dual ascent procedures, called H^1, H^2 and H^3, each one exploring a different structure of the problem. Each procedure produces a different dual solution of the LP-relaxation of F that is used to initialize the following procedure. Procedures H^1 and H^2 are based on two different relaxations of the PDPTW that do not require the generation of the entire route set. Procedure H^1 relaxes the route feasibility constraints, and procedure H^2 combines a dual ascent heuristic with column generation method and produces a lower bound of the same quality of the value of the LP-relaxation of F without being affected by the typical degeneration of the classical simplex-based column generation methods. H^3 is a classical column generation method based on the simplex algorithm that tries to close the integrality gap by adding to F additional cuts corresponding to a subset of the clique inequalities. The effectiveness of H^2 and H^3 relies on an efficient dynamic programming algorithm for generating columns of negative reduced cost that uses different bounding functions to reduce the state space graph. The exact method attempts to generate all variables of formulation F having a reduced cost smaller then the gap between the lower bound achieved and a known upper bound. When the number of such variables is sufficiently small the resulting problem is solved by an integer linear programming solver; otherwise, the problem is solved by a branch-and-cut-and-price algorithm. The computational results show that the proposed lower bound is superior to the lower bounds presented in the literature. The exact algorithm outperforms the exact method of Ropke and Cordeau [106] and can solve to optimality 15 instances that cannot be solved by this latter method.

The remainder of the chapter is organized as follows. In Section 4.2 we describe the PDPTW and its objective functions considered in this chapter and a set partitioning-like formulation F. Section 4.3 describes three relaxations of formulation F and Section 4.4 describes three bounding procedures based on these relaxations. Section 4.5 describes the exact algorithms for solving the PDPTW. The algorithms for solving the pricing problem are given in Section 4.6 and computational results on test instances from the literature are reported in Section 4.7. Finally, Section 4.8 contains concluding remarks.

4.2. Problem Description and Mathematical Formulation

In this section we describe two variants of the PDPTW which differ for the objective function to be minimized and we give a set partitioning formulation of the PDPTW.

4.2.1 Description of the PDPTW and its objective functions

The PDPTW is defined on a complete digraph $G = (V, A)$, where $V = \{0, 1, \ldots, 2n\}$ is a set of $2n + 1$ vertices and A is the arc set. Vertex 0 represents the depot and the vertex subsets $P = \{1, \ldots, n\}$ and $D = \{n + 1, \ldots, 2n\}$ contain the pickup and delivery vertices of n transportation requests, respectively. With each request $i = 1, \ldots, n$ is associated a pickup vertex $i \in P$ and a delivery vertex $n+i \in D$. In the following the set P will also be used to represent the set of the n transportation requests.

With each vertex $i \in V$ is associated a load q_i such that $q_i \geqslant 0$, $\forall i \in P$, and $q_i = -q_{i-n}$, $\forall i \in D$. Moreover, with each vertex $i \in V$ are associated a non-negative service time s_i and a time window $[a_i, b_i]$, where a_i and b_i represent the earliest and latest time to visit vertex i, respectively. If a vehicle arrives at vertex i at time $\tau_i < a_i$, then it must wait until time a_i before servicing vertex i. We

assume $q_0 = 0$ and $d_0 = 0$. With each arc $(i, j) \in A$ are associated a travel cost d_{ij} and a travel time t_{ij} that includes the service time s_i at vertex i. We assume that both travel costs and times satisfy the triangle inequality.

A fleet of m identical vehicles of capacity Q located at the depot 0 can be used to perform routes. A *route* is a circuit $R = (i_0 = 0, i_1, \ldots, i_h, i_{h+1} = 0)$ in G starting and ending at the depot 0, that represents the trip of a vehicle servicing the subset of requests $S(R) = V(R) \cap P$, where $V(R)$ is the subset of vertices visited by route R. In order to be *feasible* a route R must be simple and must satisfy the following constraints: (i) R visits the delivery vertex $n+i$ after having visited the pickup vertex $i \in S(R)$, (ii) the total load $\sum_{r=1}^{k} q_{i_r}$ of the vehicle leaving vertex i_k, $k = 1, \ldots, h$, does not exceed Q, and (iii) the arrival time τ_{i_k} at each vertex i_k is within the associated time window $[a_{i_k}, b_{i_k}]$, $k = 0, \ldots, h+1$. The arrival time τ_{i_k} is recursively computed by setting $\tau_{i_0} = a_0$ and $\tau_{i_k} = \max\{a_{i_k}, \tau_{i_{k-1}} + t_{i_{k-1} i_k}\}$, $k = 1, \ldots, h+1$. The cost of a route R is equal to the sum of the travel costs of the arcs traversed.

The PDPTW consists in designing at most one route for each vehicle so that each customer request is serviced by one route and a given objective function is minimized. One objective, hereafter called o1, is to minimize the sum of the route costs. A second one, hereafter called o2, involves a fixed cost W associated with each vehicle and is to minimize the sum of fixed costs and route costs. If W is very large (i.e., larger than the route cost of any feasible solution), then objective o2 minimizes first the number of vehicles used and second the sum of the route costs. In the following we denote by PDPTW-o1 and PDPTW-o2 the variants of the PDPTW where the objective is to minimize the objective functions o1 and o2, respectively. In the following section we give a mathematical formulation which is valid for both variants of the PDPTW as any PDPTW-o2 instance can be transformed into a PDPTW-o1 instance by adding the fixed cost W to each outgoing arc from the depot.

4.2.2 Set Partitioning Formulation

Let $\mathcal{R}$ be the index set of all feasible routes and let $\mathcal{R}_i \subseteq \mathcal{R}$ be the index subset of the routes servicing request $i \in P$. Each route $\ell \in \mathcal{R}$ has an associated cost c_ℓ. In the following we will use R_ℓ and $S(R_\ell)$ (or simply S_ℓ) to denote the ordered sequence of vertices and the subset of requests visited by the route $\ell \in \mathcal{R}$, respectively. Let x_ℓ be a $(0-1)$ binary variable that is equal to 1 if and only if route ℓ is in solution. The PDPTW can be formulated as follows.

$$(F) \quad z(F) = \min \sum_{\ell \in \mathcal{R}} c_\ell x_\ell \tag{4.1}$$

$$s.t. \sum_{\ell \in \mathcal{R}_i} x_\ell = 1, \quad \forall i \in P, \tag{4.2}$$

$$\sum_{\ell \in \mathcal{R}} x_\ell \leqslant m, \tag{4.3}$$

$$x_\ell \in \{0,1\}, \quad \forall \ell \in \mathcal{R}. \tag{4.4}$$

Constraints (4.2) specify that each request $i \in P$ must be serviced by exactly one route; constraint (4.3) imposes that at most m routes are in solution. We denote by LF the LP-relaxation of problem F.

Relaxation LF provides a tight lower bound on the PDPTW-o1. However it can be very weak for those PDPTW-o2 instances where the fixed cost W is large. In Section 4.5 we describe an exact method for the PDPTW-o1 that is based on relaxation LF, and an exact method for the PDPTW-o2. This latter method decomposes a PDPTW-o2 instance into $m_{UB} - m_{LB} + 1$ PDPTW-o1 problems obtained from the original PDPTW-o2 by setting $W = 0$, where m_{UB} and m_{LB} represent an upper bound and a lower bound on the number of vehicles required, respectively. Both exact algorithms use three bounding procedures that compute different lower bounds on the PDPTW-o1 by solving three relaxations of problem F which do not require the a-priori computation of the entire route set $\mathcal{R}$.

4.3. Relaxations of the PDPTW-o1

In this section we describe three different relaxations of the mathematical formulation described in the previous section that are used to derive different lower bounds on the PDPTW-o1.

4.3.1 Relaxation LR

The first relaxation of the PDPTW-o1 is based on the observation that any feasible PDPTW-o1 solution is composed of at most m routes, each ending in a different delivery vertex. LR corresponds to an integer problem that is derived from problem F by relaxing in a Lagrangian fashion constraints (4.2) using penalties $\mu_i \in \mathbb{R}$, $\forall i \in P$, and imposing that each route in solution must visit a different delivery vertex before returning to the depot. Let $\mathcal{D}_i \subseteq \mathcal{R}_i$ be index subset of the routes such that $i \in D$ is the last vertex visited before returning to the depot. Notice that $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$, $\forall i, j \in D, i \neq j$ and $\mathcal{R} = \cup_{i \in D} \mathcal{D}_i$. For a given penalty vector $\boldsymbol{\mu} \in \mathbb{R}^n$, the following integer problem $LR(\boldsymbol{\mu})$ provides a valid lower bound on the PDPTW-o1.

$$(LR(\boldsymbol{\mu})) \quad z(LR(\boldsymbol{\mu})) = \min \sum_{i \in D} \sum_{\ell \in \mathcal{D}_i} \left(c_\ell - \sum_{j \in S_\ell} \mu_j \right) x_\ell + \sum_{i \in P} \mu_i \tag{4.5}$$

$$s.t. \sum_{\ell \in \mathcal{D}_i} x_\ell \leqslant 1, \quad \forall i \in D, \tag{4.6}$$

$$\sum_{\ell \in \mathcal{R}} x_\ell \leqslant m, \tag{4.7}$$

$$x_\ell \in \{0, 1\}, \quad \forall \ell \in \mathcal{R}. \tag{4.8}$$

Notice that constraints (4.6) are redundant in the original problem F, but they help strengthening the relaxation $LR(\boldsymbol{\mu})$. Problem $LR(\boldsymbol{\mu})$ cannot be solved directly as it requires the entire route set $\mathcal{R}$. In Section 4.4.1 we describe a Lagrangian ascent procedure, called H^1, that solves a relaxation of problem $LR(\boldsymbol{\mu})$ where the routes are not necessarily feasible.

Procedure H^1 computes a lower bound LH1 on the PDPTW-o1 by solving the Lagrangian dual problem $LH1 = \max_{\mu} [LR(\mu)]$ using subgradient optimization.

4.3.2 Relaxation LF

The second relaxation is based on the dual problem, called DLF, of LF. Associate dual variables $u_i \in \mathbb{R}$, $\forall i \in P$, with constraints (4.2) and a dual variable $u_0 \leqslant 0$ with constraint (4.3). Let $\mathbf{u} = (u_0, u_1, \ldots, u_n)$ be the vector of the dual variables. Problem DLF is as follows.

$$(DLF) \quad z(DLF) = \max \sum_{i \in P} u_i + m u_0 \tag{4.9}$$

$$s.t. \sum_{i \in S_\ell} u_i + u_0 \leqslant c_\ell, \quad \forall \ell \in \mathcal{R}, \tag{4.10}$$

$$u_0 \leqslant 0 \text{ and } u_i \in \mathbb{R}, \forall i \in P. \tag{4.11}$$

Problem DLF is impractical to solve because it involves an exponential number of constraints. However, it is possible to find a near optimal DLF solution without generating all constraints (4.10), yet any feasible DLF solution provides a valid lower bound on the PDPTW-o1.

In Section 4.4.1, we will show that the $LR(\mu)$ solution of cost LH1 obtained by procedure H^1 can be transformed into a feasible solution $\mathbf{u}^1$ of problem DLF. In Section 4.4.2 we describe a dual ascent heuristic for solving DLF, called H^2, that produces a near optimal DLF solution $\mathbf{u}^2$ of cost LH2. H^2 is a column generation method executed after H^1 and uses the dual solution $\mathbf{u}^1$ produced by H^1 to generate the initial master problem. H^2 differs from standard column generation methods as it uses Lagrangian relaxation and subgradient optimization to solve the master problem.

4.3.3 Relaxation LSF

Since problem F is a generalization of the set partitioning problem (SP), any class of valid inequalities for SP can be used to strengthen relaxation LF. However, it

is well known that several difficulties arise when cuts are added within a column generation framework because the added inequalities can destroy the structure of the pricing problem.

A well known class of valid inequalities defining facets of the set partitioning polytope are the clique inequalities [see 12, 72]. Clique inequalities are known to significantly improve relaxation LF in the case of the Capacitated Vehicle Routing Problem (CVRP) [see 16]. However, after adding clique inequalities to problem LF it is not easy to efficiently compute the reduced cost of the routes when using a pricing problem based on shortest paths computations because reduced costs do not only depend on the arcs traversed.

A feasible approach, used by Baldacci et al. [16] for solving the CVRP, is to generate routes of non-decreasing negative reduced cost with respect to the duals of constraints (4.2) and (4.3) ignoring the duals associated with the clique inequalities. Once solved the pricing problem, the generated route subset is post-processed by adding to the reduced cost of each route the dual variable of all clique inequalities involving it. As a result, all routes whose actual reduced cost turns out to be non negative are discarded. Notice that, since the pricing problem generates a superset of the routes having negative reduced cost, special care must be taken in order to avoid generating duplicated routes. This method was very effective in solving the CVRP, but it can be time consuming especially when a large number of clique inequalities are added, each one involving a large number of variables.

To overcome this drawback we used a subset of clique inequalities whose dual variables can be easily taken into account when solving the pricing problem. Let $\mathcal{C} = \{C \subseteq P : |C| = 3\}$ be the set of all request triplets and let $\mathcal{R}(C) \subseteq \mathcal{R}$ be the subset of routes servicing at least two requests in C, i.e., $\mathcal{R}(C) = \{\ell \in \mathcal{R} : |S_\ell \cap C| \geqslant 2\}$. It is quite obvious that the following inequalities are valid:

$$\sum_{\ell \in \mathcal{R}(C)} x_\ell \leqslant 1, \quad \forall C \in \mathcal{C}. \tag{4.12}$$

Inequalities (4.12) are a subset of the clique inequalities and a special case of the

Subset Row inequalities introduced by Jepsen et al. [76].

We denote by LSF the problem obtained by adding to LF all inequalities (4.12). Relaxation LSF is solved by means of a column and cut generation method, called H^3, that is described in Section 4.4.3. H^3 computes a lower bound LH3 as the cost $z(LSF)$ of an optimal LSF solution.

4.4. Bounding Procedures for the PDPTW-o1

In this section we describe the three bounding procedures, H^1, H^2 and H^3, that are based on the three different relaxations of the PDPTW-o1 described in Section 4.3. Procedure H^1 is based on a method originally proposed by Baldacci et al. [15] for the time constrained vehicle routing problem but uses an original relaxation of the route set $\mathcal{R}$. Procedures H^2 and H^3 are extensions to the PDPTW-o1 of two methods proposed by Baldacci et al. [16] for the CVRP but use a new route generator (described in Section 4.6) specifically designed for the PDPTW.

4.4.1 Bounding Procedure H^1

Procedure H^1 solves the Lagrangian problem $LR(\boldsymbol{\mu})$, described in Section 4.3.1, by relaxing the requirement that each route in $\mathcal{R}$ must be feasible. H^1 is based on a relaxation of feasible routes called (t,i)-*routes*. A (t,i)-route is a not necessarily simple circuit in G that starts and ends at the depot and such that: (i) i is the last vertex visited before returning to the depot and (ii) the arrival time τ_j at each vertex j visited is within the associated time window $[a_j, b_j]$. In other words, (t,i)-routes correspond to a relaxation of feasible routes $\mathcal{D}_i$ that may be non elementary and may violate capacity, pairing and precedence constraints.

Let $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_{2n})$ be a vector of real penalties associated with the set of vertices $P \cup D$ and define for each arc $(i,j) \in A$ a modified arc cost $d'_{ij} = d_{ij} - \lambda_j$ (we assume $\lambda_0 = 0$). Using the modified arc costs $\{d'_{ij}\}$ let ϕ_i be the cost of a least

cost (t,i)-route. The values ϕ_i, $\forall i \in D$, can be computed in pseudo-polynomial time using the (t,i)-path functions described in Section 4.6.

It is easy to see that, by setting $\mu_i = \lambda_i + \lambda_{i+n}$, $\forall i \in P$, the values $\phi(i)$ satisfy the following inequalities:

$$\phi_i \leqslant \min_{\ell \in \mathcal{D}_i} [c_\ell - \mu(S_\ell)] \quad \forall i \in D. \tag{4.13}$$

For a given $\boldsymbol{\lambda}$, we denote by $z(LR'(\boldsymbol{\lambda}))$ the problem of choosing $m' \leqslant m$ (t,i)-routes of minimum cost ending in m' different vertices of D. Let ξ_i be an integer $(0,1)$ variable that is equal to 1 if and only if the (t,i)-route of cost ϕ_i is in solution. Problem $LR'(\boldsymbol{\lambda})$ is as follows.

$$(LR'(\boldsymbol{\lambda})) \quad z(LR'(\boldsymbol{\lambda})) = \min \sum_{i \in D} \phi_i \xi_i + \sum_{i \in P \cup D} \lambda_i \tag{4.14}$$

$$s.t. \sum_{i \in D} \xi_i \leqslant m, \tag{4.15}$$

$$\xi_i \in \{0,1\}, \quad \forall i \in D. \tag{4.16}$$

It is quite easy to observe that $LR'(\boldsymbol{\lambda})$ is a relaxation of problem $LR(\boldsymbol{\mu})$, where $\mu_i = \lambda_i + \lambda_{i+n}$, $\forall i \in P$. Therefore, the cost of an optimal $LR'(\boldsymbol{\lambda})$ solution provides a valid lower bound on the PDPTW-o1.

$LR'(\boldsymbol{\lambda})$ is a single constraint $0-1$ integer problem that can be solved by inspection as follows. Let the delivery vertices in D be ordered such that $\phi_{i_1} \leqslant \phi_{i_2} \leqslant \ldots \leqslant \phi_{i_n}$. An optimal solution $\boldsymbol{\xi}^*$ to problem $LR'(\boldsymbol{\lambda})$ is given by setting:

$$\xi^*_{i_k} = \begin{cases} 1, & \text{if } \phi_{i_k} < 0 \text{ and } k \leqslant m \\ 0, & \text{otherwise} \end{cases} \qquad k = 1, \ldots, n. \tag{4.17}$$

Computing Lower Bound LH1

Let a_{ij} be the number of times that vertex $i \in P \cup D$ is visited by the (t,j)-route of cost ϕ_j. A valid subgradient $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_{2n})$ of the function $z(LR'(\boldsymbol{\lambda}))$ at point $\boldsymbol{\lambda}$ is given by setting:

$$\delta_i = \sum_{j \in D} a_{ij} \xi^*_j - 1, \forall i \in P \cup D. \tag{4.18}$$

H^1 is an iterative procedure that performs $Maxt1$ iterations and uses subgradient optimization to compute $LH1 = \max_{\lambda}\{z(\overline{LR}(\boldsymbol{\lambda}))\}$.

At each iteration, procedure H^1 solves problem $\overline{LR}(\boldsymbol{\lambda})$ by computing the least cost (t, i)-route of cost ϕ_i, $\forall i \in D$, with respect to the modified arc costs $\{d'_{ij}\}$, and derives an optimal $LR'(\boldsymbol{\lambda})$ solution $\boldsymbol{\xi}^*$ using expressions (4.17). Then, H^1 computes the subgradient vector $\boldsymbol{\delta}$ and updates the penalty vector $\boldsymbol{\lambda}$ as:

$$\lambda_i = \lambda_i - \epsilon \frac{z(UB) - z(LR'(\boldsymbol{\lambda}))}{\sum_{k \in P \cup D}(\delta_k - 1)^2}(\delta_i - 1), \forall i \in P \cup D, \tag{4.19}$$

where ϵ is a positive constant and $z(UB)$ is an upper bound on the PDPTW-o1.

The following theorem shows that any optimal $LR'(\boldsymbol{\lambda})$ solution $\boldsymbol{\xi}^*$ provides a feasible DLF solution $\mathbf{u}$ of the same cost.

Theorem 4.1 *For a given vector* $\boldsymbol{\lambda}$*, let* $\boldsymbol{\xi}^*$ *be an optimal* $LR'(\boldsymbol{\lambda})$ *solution, and let* $D^* = \{i \in D : \xi^*_i = 1\}$*. Defining* $\mu_i = \lambda_i + \lambda_{n+i}$, $\forall i \in P$*, a feasible* DLF *solution* $\mathbf{u}$ *of cost* $z(LR'(\boldsymbol{\lambda}))$ *is given by:*

$$u_i = \begin{cases} \mu_i, & \text{if } n + i \notin D^* \\ \mu_i + \phi_{n+i} - \sigma, & \text{if } n + i \in D^* \end{cases}, \forall i \in P, \tag{4.20}$$

$$u_0 = \sigma,$$

where $\sigma = \max_{i \in D^*}\{\phi_i\}$*, if* $|D^*| = m$*;* $\sigma = 0$ *otherwise.*

Proof: Notice that $\sigma \leqslant 0$ since, from the definition of D^*, we have $\phi_i < 0$, $\forall i \in D^*$. Hence, the DLF solution $\mathbf{u}$ defined by expressions (4.20) satisfies constraints (4.11). Now we show that the vector $\mathbf{u}$ given by expressions (4.20) satisfies inequalities (4.10) for any route $\ell \in \mathcal{R}$, that is:

$$\sum_{i \in S_\ell} u_i + u_0 \leqslant c_\ell. \tag{4.21}$$

Let $S^*_\ell = \{i \in S_\ell : n + i \in D^*\}$ and let $\beta(\ell)$ be the last delivery vertex visited by route ℓ. From expressions (4.20), we have:

$$\sum_{i \in S_\ell} u_i + u_0 = \sum_{i \in S_\ell} \mu_i + \sum_{i \in S^*_\ell}(\phi_{n+i} - \sigma) + \sigma. \tag{4.22}$$

We have two cases.

A) $S^*_\ell = \emptyset$. In this case we have to show that $\sum_{i \in S_\ell} \mu_i + \sigma \leqslant c_\ell$. Since $\ell \in \mathcal{D}_{\beta(\ell)}$, from expressions (4.13) we have $\phi_{\beta(\ell)} \leqslant (c_\ell - \sum_{i \in S_\ell} \mu_i)$ and since $\beta(\ell) \notin D^*$ we have $\sigma \leqslant \phi_{\beta(\ell)}$. Combining these two inequalities we obtain $\sigma \leqslant (c_\ell - \sum_{i \in S_\ell} \mu_i)$, or $\sum_{i \in S_\ell} \mu_i + \sigma \leqslant c_\ell$.

B) $S^*_\ell \neq \emptyset$. Let $k^* \in S^*_\ell$ be such that $\phi_{n+k^*} = \min\{\phi_{n+i} : i \in S^*_\ell\}$. Note that $\phi_{n+i} - \sigma \leqslant 0$ as $\sigma \geqslant \phi_{n+i}$, $\forall n + i \in D^*$. Therefore we have:

$$\sum_{i \in S^*_\ell} (\phi_{n+i} - \sigma) \leqslant \phi_{n+k^*} - \sigma. \tag{4.23}$$

From expressions (4.22) and (4.23), we derive the following inequality:

$$\sum_{i \in S_\ell} u_i + u_0 \leqslant \sum_{i \in S_\ell} \mu_i + \sigma + \phi_{n+k^*} - \sigma = \sum_{i \in S_\ell} \mu_i + \phi_{n+k^*}. \tag{4.24}$$

From the definition of k^* we have $\phi_{n+k^*} \leqslant \phi_{\beta(\ell)}$, and because of expression (4.13) we obtain $\phi_{n+k^*} \leqslant \phi_{\beta(\ell)} \leqslant (c_\ell - \sum_{i \in S_\ell} \mu_i)$. Therefore, from inequality (4.24), we derive:

$$\sum_{i \in S_\ell} u_i + u_0 \leqslant \sum_{i \in S_\ell} \mu_i + (c_\ell - \sum_{i \in S_\ell} \mu_i) = c_\ell. \tag{4.25}$$

□

Let $\boldsymbol{\lambda}^*$ be the value that produces the best lower bound LH1 achieved by procedure H^1, i.e., such that $LH1 = z(LR'(\boldsymbol{\lambda}^*))$. We denote by $\mathbf{u}^1$ the DLF solution of cost $z(DLF) = LH1$ given by expressions (4.20) setting $\mu_i = \lambda^*_i + \lambda^*_{n+i}$, $\forall i \in P$.

4.4.2 Bounding Procedure H^2

Procedure H^2 is an extension to the PDPTW of a method originally proposed by Baldacci et al. [16] for the CVRP and it is based on the following theorem.

Theorem 4.2 (Baldacci et al. [16]) *Associate penalties $\mu_i \in \mathbb{R}$, $\forall i \in P$, and $\mu_0 \leqslant 0$ with constraints (4.2) and (4.3), respectively. Let w_i, $\forall i \in P$, be a non-negative weight*

associated with each pickup vertex i. *A feasible* DLF *solution* $\mathbf{u}$ *of cost* $z(\mathrm{DLF}(\boldsymbol{\mu}))$ *can be obtained by means of the following expressions:*

$$u_i = w_i \min_{\ell \in \mathcal{R}_i} [(c_\ell - \mu(S_\ell))/w(S_\ell)] + \mu_i, \quad \forall i \in P, \tag{4.26}$$

$$u_0 = \mu_0 \tag{4.27}$$

where $w(S_\ell) = \sum_{i \in S_\ell} w_i$ *and* $\mu(S_\ell) = \sum_{i \in S_\ell} \mu_i$.

A valid lower bound on problem F is given by $\max_{\boldsymbol{\mu}}\{z(\mathrm{DLF}(\boldsymbol{\mu}))\}$ and can be computed using subgradient optimization. Let $\ell(i) \in \mathcal{R}$ be the index of the route producing u_i in expressions (4.26), and let a_{ij} be the number of times that customer $j \in P$ is visited by the route $\ell(i)$. It can be shown that a valid subgradient of the function $z(\mathrm{DLF}(\boldsymbol{\mu}))$ at point $\boldsymbol{\mu}$ is given by the vector θ computed as follows:

$$\theta_j = \sum_{i \in P} a_{ij} w_i / w(S_{\ell(i)}) - 1, \quad \forall j \in P, \quad \text{and} \quad \theta_0 = \sum_{i \in P} w_i / w(S_{\ell(i)}) - m. \tag{4.28}$$

Procedure H^2 is an iterative method that uses column generation to solve equations (4.26) and subgradient optimization to compute $LH2 = \max_{\boldsymbol{\mu}}\{z(\mathrm{DLF}(\boldsymbol{\mu}))\}$. H^2 is initialized by setting $\boldsymbol{\mu} = \mathbf{u}^1$ and by generating a subset $\overline{\mathcal{R}} \subseteq \mathcal{R}$ containing the Δ^{min} routes of minimum reduced cost with respect to the DLF solution $\mathbf{u}^1$ obtained by H^1, where Δ^{min} is an a-priori defined parameter. Procedure H^2 executes an a-priori defined number $Maxt2$ of macro iterations, each performing the following two steps:

1. *Solve the Master Problem*. The master problem is obtained from LF by replacing $\mathcal{R}$ with $\overline{\mathcal{R}}$. A near-optimal solution $\bar{\mathbf{u}}$ of cost $\bar{\mathbf{z}}$ of the master problem is obtained by an iterative method that performs $Maxt3$ iterations. At each iteration it uses expressions (4.26) and subgradient optimization to modify the penalty vector $\boldsymbol{\mu}$ in order to maximize the cost $\bar{\mathbf{z}}$ of the master dual solution $\bar{\mathbf{u}}$.

2. *Check if the master dual solution* $\bar{\mathbf{u}}$ *satisfies all DLF constraints*. Generate the largest subset $\mathcal{N}$ of the routes having the most negative reduced cost with respect to the dual master solution $\bar{\mathbf{u}}$ and such that $|\mathcal{N}| \leqslant \Delta^a$, where Δ^a is a parameter

defined a-priori. If $\mathcal{N} = \emptyset$ and $\bar{\mathbf{z}}$ is greater than LH2, update LH2$= \bar{\mathbf{z}}$ and $\mathbf{u}^2 = \bar{\mathbf{u}}$; otherwise, update $\overline{\mathcal{R}} = \overline{\mathcal{R}} \cup \mathcal{N}$.

The initial route subset $\overline{\mathcal{R}}$ and the subsets $\mathcal{N}$ are generated using procedure GENR described in Section 4.6. The weights w_i are computed as $w_i = \max[d_{0i}, d_{0n+i}]$, $\forall i \in P$.

4.4.3 Bounding Procedure H^3

Procedure H^3 is a column and cut generation method that computes a lower bound LH3 as the cost of an optimal solution of problem LSF introduced in Section 4.3.3. H^3 generates the initial master problem using the dual solution $\mathbf{u}^2$ given by H^2. The initial master problem $\overline{\text{LSF}}$ is obtained by replacing in LSF the set $\mathcal{R}$ with the subset $\overline{\mathcal{R}}$ containing the routes having the smallest reduced cost with respect to the dual solution $\mathbf{u}^2$ achieved by H^2, and by replacing $\mathcal{C}$ with the subset $\overline{\mathcal{C}} = \emptyset$.

At a given iteration (say t), H^3 computes the optimal primal and dual solutions $\bar{\mathbf{x}}$ and $(\bar{\mathbf{u}}, \bar{\mathbf{v}})$ of $\overline{\text{LSF}}$ using the simplex algorithm, where $\bar{\mathbf{v}}$ is the dual vector associated with constraints (4.12), and performs the following operations:

i) generates the subset $\mathcal{C}' \subseteq \mathcal{C}$ of the θ most violated inequalities (4.12), where θ is an a-priori defined parameter;

ii) generates the largest subset $\mathcal{N}$ of the Δ^q routes having the largest negative reduced cost with respect to $(\bar{\mathbf{u}}, \bar{\mathbf{v}})$.

If $\mathcal{N} = \emptyset$ and $\mathcal{C}' = \emptyset$, then procedure H^3 terminates; otherwise, H^3 updates $\overline{\mathcal{R}} = \overline{\mathcal{R}} \cup \mathcal{N}$ and $\overline{\mathcal{C}} = \overline{\mathcal{C}} \cup \mathcal{C}'$ and performs a new iteration. The initial route subset $\overline{\mathcal{R}}$ and the subset $\mathcal{N}$ are generated using procedure GENR described in Section 4.6. The separation of violated inequalities (4.12) is performed by complete enumeration and is based on the following observation.

Let $\overline{\mathcal{R}}(\bar{\mathbf{x}}) \subseteq \overline{\mathcal{R}}$ be the subset of routes corresponding to variables $\bar{x}_\ell > 0$ in the current master solution $\bar{\mathbf{x}}$ and let $\bar{\omega}_{ij} = \sum_{\ell \in \overline{\mathcal{R}}(\bar{\mathbf{x}}) \cap (\overline{\mathcal{R}}_i \cap \overline{\mathcal{R}}_j)} \bar{x}_\ell$ be the sum of the variables

corresponding to the routes servicing both the vertices $i, j \in P$. Let us associate with each request triplet C the weight $w(C) = \sum_{i,j \in C} \bar{\omega}_{ij}$. It is easy to observe that any request triplet $C = (i_1, i_2, i_3)$ such that $w(C) > 1$ corresponds to a violated inequality (4.12). Violated inequalities (4.12) can be separated exactly in time $O\left(n^3\, n_B\right)$, where $n_B = |\overline{\mathcal{R}}(\bar{\mathbf{x}})|$, simply by computing the values $\bar{\omega}_{ij}$, $i, j = 1, \ldots, n$, $i \neq j$, and by enumerating all the request triplets in P.

Let $\mathcal{C}'$ be the subset of violated inequalities (4.12) separated at a given iteration. In our computational experiments we found computationally convenient to restrict $\mathcal{C}'$ in such a way that each request triplet C has at most one vertex in common with any other triplet in $\mathcal{C}'$. Moreover, we impose that the the set $\mathcal{C}'$ cannot contain more than θ violated inequalities (4.12), where θ is an a-priori defined parameter. At each iteration the set $\mathcal{C}'$ is computed as follows.

1. Initialize $\mathcal{C}' = \emptyset$ and let $(C_1, C_2, \ldots, C_{\hat{k}})$ be the ordered set of all triplets such that $w(C_k) > 1$, $k = 1, \ldots, \hat{k}$ and $w(C_1) \geqslant w(C_2) \geqslant \ldots w(C_{\hat{k}})$.
2. For each $k = 1, \ldots, \hat{k}$ perform the following steps:
   (a) If $|C_k \cap C| \leqslant 1$, $\forall C \in \mathcal{C}'$, then add C_k to $\mathcal{C}'$
   (b) If $|\mathcal{C}'| = \theta$ stop.

The peculiar structure of inequalities (4.12), i.e., being completely defined by a vertex triplet, allows us to easily lift each inequality every time a new set $\mathcal{N}$ of routes is added to $\overline{\mathcal{R}}$. Indeed, to lift an inequality (4.12) associated with a request triple C it suffices to check, for each route $\ell \in \mathcal{N}$, if at least two vertices in C are visited by the route. This being the case the route ℓ belongs to the set $\mathcal{R}(C)$ and the corresponding inequality (4.12) can be extended accordingly.

In the following we denote by $(\mathbf{u}^3, \mathbf{v}^3)$ the optimal dual solution of LSF of cost LH3, obtained by H^3.

4.5. An Exact Algorithm for the PDPTW

In this section we describe an exact algorithm for solving the PDPTW-o1 that combines the method proposed by Baldacci et al. [16] for the CVRP with a branch-and-cut-and-price algorithm. We also describe an exact algorithm for solving the PDPTW-o2 that uses the same bounding procedures developed for the PDPTW-o1.

4.5.1 Solving the PDPTW-o1

The exact method is made up of two main phases. In the first phase we execute in sequence the bounding procedures H^1, H^2 and H^3 to compute an optimal dual solution $(\hat{\mathbf{u}}, \hat{\mathbf{v}})$ of LSF of cost $\hat{z}(LSF)$. In the second phase, the algorithm attempts to generate the subset $\widehat{\mathcal{R}} \subseteq \mathcal{R}$ of all the routes having a reduced cost smaller than or equal than the gap $z(UB) - \hat{z}(DLSF)$, where $z(UB)$ is an upper bound on the PDPTW-o1. If the size of the set $\widehat{\mathcal{R}}$ is below a given threshold (defined as an a-priori defined parameter Δ^{max}), then the algorithm derives a restricted problem $\hat{F}$ from problem F by substituting the set $\mathcal{R}$ with its subset $\widehat{\mathcal{R}}$ and adding to it all the constraints $\overline{\mathcal{C}}$ produced by H^3. $\hat{F}$ is then solved using a general purpose integer programming solver. If $\widehat{\mathcal{R}}$ is greater than Δ^{max}, then the algorithm resorts to branching and turns into a branch-and-cut-and-price algorithm. More precisely, the exact method performs the following steps:

1. Execute in sequence the bounding procedures H^1, H^2 and H^3. Let $(\hat{\mathbf{u}}, \hat{\mathbf{v}}) = (\mathbf{u}^3, \mathbf{v}^3)$, $\hat{z}(DLSF) = LH3$ and $\widehat{\mathcal{C}} = \overline{\mathcal{C}}$.

2. Set $gap = z(UB) - \hat{z}(LSF)$. Generate the largest subset $\widehat{\mathcal{R}}$ of routes whose reduced cost with respect to the dual solution $(\hat{\mathbf{u}}, \hat{\mathbf{v}})$ is smaller than gap and such that $|\widehat{\mathcal{R}}| \leqslant \Delta^{max}$, where Δ^{max} is an a-priori defined parameter.

3. Define the reduced problem $\hat{F}$ derived from F by replacing the route set $\mathcal{R}$ with the set $\widehat{\mathcal{R}}$ and by adding to F all constraints (4.12) corresponding to the triplets in $\hat{\mathcal{C}}$. We have two cases:

a) $|\widehat{\mathcal{R}}| < \Delta^{max}$: solve problem $\hat{F}$ using a general purpose integer programming solver.

b) $|\widehat{\mathcal{R}}| = \Delta^{max}$: solve the problem using the branch-and-cut-and-price algorithm described in the next section, where the lower bound at each node of the enumerative tree is computed using procedure H^3.

4.5.2 Branch-and-Cut-and-Price Algorithm for PDPTW-o1

The branch-and-cut-and-price algorithm described in this section solves problem F by using procedure H^3 at each node of the enumerative tree. Given an LSF solution $\bar{\mathbf{x}}$ involving a set of routes $\overline{\mathcal{R}}$, let $\bar{\zeta}_{ij} = \sum_{\ell \in \overline{\mathcal{R}}_{ij}} x_\ell$, $\forall (i,j) \in A$, where $\overline{\mathcal{R}}_{ij} \subseteq \overline{\mathcal{R}}$ is the subset of routes traversing arc (i,j). When H^3 terminates with a fractional solution $\bar{\mathbf{x}}$ the algorithm selects an arc (i,j) having value $\bar{\zeta}_{ij}$ closest to 0.5 (in case of ties the arc having the smallest modified arc cost $d_{ij} - u_j^3$ is chosen), and creates two branches imposing the disjunction $\overline{\zeta}_{ij} = 0 \vee \overline{\zeta}_{ij} = 1$. The nodes are processed according to a best bound node selection rule. A possible drawback of this simple branching strategy is that it can result in a larger number of nodes explored by the enumerative tree with respect to more sophisticated branching methodologies. On the other hand, it permits to transfer all the branching conditions to the pricing subproblem simply by removing arcs from the graph G and does not require that additional constraints are added to the master problem.

The algorithm uses a pool of routes $\mathcal{R}^0 \subseteq \mathcal{R}$ and a pool of cuts $\mathcal{C}^0$, which are initialized by setting $\mathcal{R}^0 = \widehat{\mathcal{R}}$ and $\mathcal{C}^0 = \hat{\mathcal{C}}$. At each node of the enumerative tree the pools $\mathcal{R}^0$ and $\mathcal{C}^0$ are used to initialize the initial master problem $\overline{LSF}$ of H^3. The route subset $\overline{\mathcal{R}}$ is obtained by extracting from $\mathcal{R}^0$ the largest set of routes satisfying the branching conditions, whereas the subset of triplets $\overline{\mathcal{C}}$ is initialized by setting $\overline{\mathcal{C}} = \mathcal{C}^0$. The route pool $\mathcal{R}^0$ and the cut pool $\mathcal{C}^0$ are enlarged at each node during the execution of H^3 by setting $\mathcal{R}^0 = \mathcal{R}^0 \cup \mathcal{N}$ and $\mathcal{C}^0 = \mathcal{C}^0 \cup \mathcal{C}'$, where $\mathcal{N}$ and $\mathcal{C}'$ are the route subset and the cut subset generated at each iteration of H^3, respectively. Notice that as a result of the branching conditions imposed it is possible that at

a given node suboptimal routes are generated. To overcome this drawback we reject all those routes that are dominated by a route in the pool $\mathcal{R}^0$.

4.5.3 Solving the PDPTW-o2

It is quite easy to observe that any PDPTW-o2 instance can be transformed into a PDPTW-o1 instance by adding the fixed cost W to each outgoing arc from the depot. The resulting problem can then be solved using the exact method described for the PDPTW-o1. This strategy, however, can be time consuming because the relaxation LSF of the resulting PDPTW-o1 problem provides a weak lower bound when W is very large. In this section, we describe an exact method for the PDPTW-o2 which attempts to overcome this drawback.

Let m_{LB} and m_{UB} be a lower and an upper bound on the number of vehicles used in any optimal PDPTW-o2 solution, respectively. The proposed exact method solves at most $m_{UB} - m_{LB} + 1$ PDPTW-o1 instances, each obtained from the original PDPTW-o2 instance by setting $W = 0$ and $m = \kappa$, with $\kappa = m_{LB}, \ldots, m_{UB}$. With each PDPTW-o1 instance involving $m = \kappa$ vehicles we associate a problem $F(\kappa)$ and the corresponding relaxation $LSF(\kappa)$ which are derived from F and LSF by setting $m = \kappa$. The values of m_{LB} and m_{UB} are computed as follows.

Let $z(UB)$ be an upper bound on the original PDPTW-o2 instance. The value m_{UB} can be computed as $m_{UB} = \lfloor z(UB)/W \rfloor$. However, a better estimate of m_{UB} is obtained by computing a valid lower bound r_{LB} on problem $F(\lfloor z(UB)/W \rfloor)$, and setting $m_{UB} = \lfloor \frac{z(UB)-r_{LB}}{W} \rfloor$.

Lower bound m_{LB} can be set equal to the smallest integer κ such that problem $F(\kappa)$ has a feasible solution and can be computed as follows. Let $LH3(\kappa)$ be the optimal solution cost of relaxation $LSF(\kappa)$ of $F(\kappa)$, achieved by procedure H^3 (we assume $LH3(\kappa) = \infty$ in case $LSF(\kappa)$ has no feasible solution). Then, m_{LB} can be set equal to the smallest integer κ such that $LH3(\kappa) < \infty$.

Notice that after computing m_{UB} and m_{LB}, a valid lower bound on the original

PDPTW-o2 problem can be computed as:

$$LB_0 = \min_{m_{LB} \leqslant \kappa \leqslant m_{UB}} \{W \times \kappa + LH3(\kappa)\}. \tag{4.29}$$

The proposed exact method iteratively solves problems $F(\kappa)$, $\kappa = m_{LB}, \ldots, m_{UB}$, but terminates prematurely at iteration $\kappa < m_{UB}$ if $W \times (\kappa - 1) + z^*(F(\kappa - 1)) < W \times \kappa + LH3(\kappa)$, where $z^*(F(\kappa - 1))$ is the optimal solution cost of the integer problem $F(\kappa - 1)$. A step-by-step description of the exact method for solving PDPTW-o2 is given below.

Exact algorithm for solving the PDPTW-o2

1. *Compute the lower bound* r_{LB} *on the routing cost of any PDPTW-o2 solution.*
   Execute in sequence the bounding procedures H^1 and H^2 on the PDPTW-o1 instance derived from the PDPTW-o2 by setting $W = 0$ and $m = \lfloor z(UB)/W \rfloor$. Let $(\mathbf{u}^2, \mathbf{v}^2)$ be the DLF solution of cost LH2 obtained by H^2. Set $r_{LB} = LH2$.

2. *Compute an upper bound* m_{UB} *on the number of vehicles.*
   Define $m_{UB} = \lfloor \frac{z(UB) - r_{LB}}{W} \rfloor$, where $z(UB)$ is a known upper bound on the PDPTW-o2.

3. *Compute the lower bound* m_{LB} *on the number of vehicles.*
   Generate the largest subset $\overline{\mathcal{R}}$ of the Δ^{min} routes having minimum reduced cost with respect to the DLF solution $(\mathbf{u}^2, \mathbf{v}^2)$ achieved at step 1 (Δ^{min} defined a-priori). Compute m_{LB} using the following iterative procedure.

   (3a) Initialize $\kappa = m_{UB} - 1$.

   (3b) Let $F(\kappa)$ be the PDPTW-o1 problem derived from the PDPTW-o2 by setting $W = 0$ and $m = \kappa$. Use the bounding procedure H^3 to solve relaxation $LSF(\kappa)$ of problem $F(\kappa)$ and let $LH3(\kappa)$ be the lower bound achieved. Notice that the initial master of H^3 corresponds to the route subset $\overline{\mathcal{R}}$ computed at step 3.

(3c) If $LH3(\kappa) = \infty$ (i.e., $F(\kappa)$ has no feasible solution) set $m_{LB} = \kappa + 1$ and go to step 4.

(3d) If $LH3(\kappa) < \infty$ set $\kappa = \kappa - 1$. If $\kappa = 0$ set $m_{LB} = 1$ and go to step 4; otherwise return to step (3b).

Compute the lower bound LB_0 on the PDPTW-o2 using expression (4.29).

4. *Find an optimal PDPTW-o2 solution*

For each $\kappa = m_{LB}, \ldots, m_{UB}$, execute the following procedure.

(4a) Define $\hat{\mathbf{u}} = \mathbf{u}^3(\kappa)$, $\hat{\mathbf{v}} = \mathbf{v}^3(\kappa)$, where $(\mathbf{u}^3(\kappa), \mathbf{v}^3(\kappa))$ is the dual solution of $LSF(\kappa)$ of cost $LH3(\kappa)$ computed by H^3 at step (3b). Let $r_{UB}(\kappa)$ be a valid upper bound on $F(\kappa)$ that is computed as described below. Set $\hat{z}(LSF) = LH3(\kappa)$ and $\hat{z}(UB) = r_{UB}(\kappa)$.

(4b) Solve problem $F(\kappa)$ executing steps 2 and 3 of the exact method for the PDPTW-o1 described in Section 4.5.1 and let $z^*(F(\kappa))$ be the optimal $F(\kappa)$ solution achieved. Update $z(UB) = \min\{z(UB), W \times \kappa + z^*(F(\kappa))\}$.

(4c) If $W \times \kappa + z^*(F(\kappa)) \leqslant W \times (\kappa + 1) + LH3(\kappa + 1)$, then stop: the optimal PDPTW-o1 solution found above provides an optimal PDPTW-o2 solution of cost $z(UB)$; otherwise, go to step (4a)

Computing upper bound $r_{UB}(\kappa)$

A valid upper bound $r_{UB}(\kappa)$ on $F(\kappa)$ is given by $r_{UB}(\kappa) = z(UB) - W \times \kappa$. However, this value can be very high for values of κ smaller than m_{UB} for those PDPTW-o2 instances having a very large value of W such as those where the first objective is to minimize the number of vehicles.

A better method for computing $r_{UB}(\kappa)$, when $\kappa < m_{UB}$, consists in computing $r_{UB}(\kappa)$ as an upper bound on the optimal cost of a problem $\hat{F}(\kappa)$ which is derived from $F(\kappa)$ by maximizing objective o1. We compute $r_{UB}(\kappa)$ by solving a Lagrangian relaxation $\widehat{LR}_\kappa(\boldsymbol{\mu})$ of $\hat{F}(\kappa)$ derived from $LR(\boldsymbol{\mu})$ (see Section 4.3.1), replacing "min" with "max" in the objective function (4.5). Since $z(\widehat{LR}_\kappa(\boldsymbol{\mu}))$ is a

valid upper bound on $\hat{F}(\kappa)$ for any $\boldsymbol{\mu}$, we compute $r_{UB}(\kappa) = \min_{\boldsymbol{\mu}}\{z(\widehat{LR}_\kappa(\boldsymbol{\mu}))\}$ using a straightforward adaptation of procedure H^1.

4.6. Generating feasible routes

In this section we describe a dynamic programming procedure, called GENR, that allows us to generate the subsets of feasible routes required by the bounding procedures described in Section 4.4 and by the exact methods described in Section 4.5.

Let $(\mathbf{u}', \mathbf{v}')$ be a dual LSF solution, where dual variables $\mathbf{u}' = (u'_0, u'_1, \ldots, u'_n)$ are associated with constraints (4.2) and (4.3) and variables $\mathbf{v}' = (v'_1, \ldots, v'_{|\mathcal{C}'|})$ are associated with a subset of constraints (4.12) defined by a subset $\mathcal{C}' \subseteq \mathcal{C}$ of triplets. The reduced cost c'_ℓ of a route ℓ with respect to the dual solution $(\mathbf{u}', \mathbf{v}')$ is computed as follows:

$$c'_\ell = c_\ell - \sum_{i \in S_\ell} u'_i - u'_0 - \sum_{C \in \mathcal{C}'_\ell} v'_C,$$

where $\mathcal{C}'_\ell = \{C \in \mathcal{C}' : |C \cap V(R_\ell)| \geqslant 2\}$.

Given the vectors $\mathbf{u}'$ and $\mathbf{v}'$ and two user-defined parameters γ and gap, procedure GENR generates the largest subset $\mathcal{B} \subseteq \mathcal{R}$ satisfying the following conditions:

$$\left.\begin{array}{lll} a) & \max_{\ell \in \mathcal{B}}\{c'_\ell\} \leqslant \min_{\ell \in \mathcal{R}\setminus\mathcal{B}}\{c'_\ell\} \\ b) & |\mathcal{B}| \leqslant \gamma \\ c) & \max_{\ell \in \mathcal{B}}\{c'_\ell\} < gap \end{array}\right\} \tag{4.30}$$

By appropriately setting parameters γ and gap, procedure GENR can be used to generate the route subsets required by the bounding procedures described in this chapter as follows:

- the initial route subset $\overline{\mathcal{R}}$ in procedure H^2:

  define $\mathbf{u}' = \mathbf{u}^1$, $\mathbf{v}' = \mathbf{0}$ and set $gap = z(UB) - LH1$ and $\gamma = \Delta^{min}$;

- the route subset $\mathcal{N}$ at Step 2 of H^2:

  define $\mathbf{u}' = \overline{\mathbf{u}}$, $\mathbf{v}' = \mathbf{0}$ and set $gap = 0$ and $\gamma = \Delta^a$;

- the initial route subset $\overline{\mathcal{R}}$ in procedure H^3:

  define $\mathbf{u}' = \mathbf{u}^2$, $\mathbf{v}' = \mathbf{0}$ and set $gap = z(UB) - LH2$ and $\gamma = \Delta^{min}$;

- the route subset $\mathcal{N}$ in procedure H^3:

  define $\mathbf{u}' = \overline{\mathbf{u}}$, $\mathbf{v}' = \overline{\mathbf{v}}$ and set $gap = 0$ and $\gamma = \Delta^a$;

- the route set $\hat{\mathcal{R}}$ required by the exact method of Section 4.5.1:

  define $\mathbf{u}' = \hat{\mathbf{u}}$, $\mathbf{v}' = \hat{\mathbf{v}}$, set $gap = z(UB) - \hat{z}(LSF)$ and $\Delta = \Delta^{max}$.

Procedure GENR is a dynamic programming procedure that is analogous to Dijkstra's algorithm on an expanded state-space graph dynamically generated as follows. Associate with each arc $(i,j) \in A$ a modified arc cost d'_{ij} defined as:

$$d'_{ij} = \begin{cases} d_{ij} - u'_j & \text{if } j \in P \cup \{0\}, \\ d_{ij} & \text{otherwise,} \end{cases} \qquad \forall (i,j) \in A. \tag{4.31}$$

The reduced cost c'_ℓ of any route $\ell \in \mathcal{R}$ with respect to the dual vectors $\mathbf{u}'$ and $\mathbf{v}'$ can be computed as $c'_\ell = \sum_{(i,j)\in A(R_\ell)} d'_{ij} - \sum_{C\in\mathcal{C}'_\ell} v'_C$, where $A(R_\ell)$ denotes the set of arcs traversed by the route ℓ. Procedure GENR dynamically generates a state-space graph where each vertex corresponds to a feasible *forward path*.

4.6.1 Forward and backward paths

A forward path $L = (i_0, i_1, i_2, \ldots, i_h)$, where $i_0 = 0$, is defined as a directed path in G starting from the depot, visiting a subset of vertices $V(L) \subseteq V$, and ending at vertex $e(L) = i_h$. With each vertex $i_k \in V(L)$ visited by a forward path L is associated a load $\delta^L_{i_k}$ and an arrival time $\tau^L_{i_k}$ which are computed as follows. Let $\tau^L_{i_0} = a_0$ and $\delta^L_{i_0} = 0$ and set:

$$\left.\begin{aligned} \tau^L_{i_k} &= \max\left[a_{i_k}, \tau^L_{i_{k-1}} + t_{i_{k-1}i_k}\right], \\ \delta^L_{i_k} &= \sum_{r<k} q_{i_r}, \end{aligned}\right\} \qquad k = 1, \ldots, h. \tag{4.32}$$

In the following we denote by $\tau(L) = \tau^L_{i_h}$ and $q(L) = \delta^L_{i_h}$ the arrival time and the load at the last vertex $i_h = e(L)$ of a forward path L, respectively.

Similarly, a *backward path* $\overline{L} = (i_h, i_{h-1}, i_{h-2}, \ldots, i_0)$, where $i_0 = 0$, is defined as a directed path in G that starts from an initial vertex $e(\overline{L}) = i_h \in G$, visits the subset of vertices $V(\overline{L})$, and ends at the depot. A backward path $\overline{L}$ is similar to a forward path but it is computed "bottom up", i.e., starting from the last vertex i_0 (the depot) and traversing the arcs in reverse direction. With each vertex $i_k \in V(\overline{L})$ of a backward path $\overline{L}$ are associated an arrival time $\bar{\tau}^{\overline{L}}_{i_k}$ and a load $\bar{\delta}^{\overline{L}}_{i_k}$ that are defined as follows. Let $\bar{\tau}^{\overline{L}}_{i_0} = b_0$ and $\bar{\delta}^{\overline{L}}_{i_0} = 0$ and set:

$$\left.\begin{aligned} \bar{\tau}^{\overline{L}}_{i_k} &= \min\left[b_{i_k}, \bar{\tau}^{\overline{L}}_{i_{k-1}} - t_{i_k i_{k-1}}\right], \\ \bar{\delta}^{\overline{L}}_{i_k} &= \sum_{r<k} -q_{i_r}, \end{aligned}\right\} \qquad k = 1, \ldots, h. \tag{4.33}$$

In the following $\tau(\overline{L}) = \bar{\tau}^{\overline{L}}_{i_h}$ and $q(\overline{L}) = \bar{\delta}^{\overline{L}}_{i_h}$ denote, respectively, the arrival time and the load at the initial vertex $i_h = e(\overline{L})$ of a backward path $\overline{L}$.

The reduced cost of any (forward or backward) path L with respect to solution $(\mathbf{u}', \mathbf{v}')$ is defined as $c'_\ell = \sum_{(i,j)\in A(L)} d'_{ij} - \sum_{C\in\mathcal{C}'_L} v'_C$, where $A(L)$ is the set of arcs traversed by L, $\mathcal{C}'_L = \{C \in \mathcal{C}' : C \cap V(L) \geqslant 2\}$, and the modified arc costs $\{d'_{ij}\}$ are computed using expression (4.31).

Feasible paths

Let $P(L) = V(L) \cap P$ and $D(L) = V(L) \cap D$ be the set of pickup vertices and delivery vertices visited by a path L, respectively. For any path L and for each vertex $i \in V(L)$, we denote by $\Gamma(L, i)$ and $\Gamma^{-1}(L, i)$ the sets of predecessor and successor vertices of vertex i in L, respectively.

A path L satisfies *forward precedence constraints* if for each delivery vertex $i \in D(L)$ the corresponding pickup $i - n$ is visited before i, that is, $i - n \in \Gamma^{-1}(L, i)$, $\forall i \in D(L)$. Similarly, a path L satisfies *backward precedence constraints* if for each pickup vertex $i \in P(L)$ the corresponding delivery $n + i$ is visited after i, that is, $n + i \in \Gamma(L, i)$, $\forall i \in P(L)$.

Given a path L, let the sets $\widetilde{P}(L)$ and $\widetilde{D}(L)$ be the sets of requests visited by L for which backward precedence constraints and forward precedence constraints are not satisfied, that is, $\widetilde{P}(L) = \{i \in P : i \in P(L) \text{ and } n + i \notin \Gamma(L, i)\}$ and $\widetilde{D}(L) = \{i \in P : n + i \in D(L) \text{ and } i \notin \Gamma^{-1}(L, i)\}$.

A forward path L is *feasible* if it is simple and satisfies time window constraints (i.e., $a_i \leqslant \tau_i^L \leqslant b_i$, $\forall i \in V(L)$), capacity constraints (i.e., $0 \leqslant \delta_i^L \leqslant Q$, $\forall i \in V(L)$), and forward precedence constraints (i.e., $\widetilde{D}(L) = \emptyset$).

Similarly, a backward path $\overline{L}$ is *feasible* if it simple and satisfies time window constraints (i.e., $a_i \leqslant \bar{\tau}_i^L \leqslant b_i$, $\forall i \in V(\overline{L})$), capacity constraints (i.e., $0 \leqslant \bar{\delta}_i^L \leqslant Q$, $\forall i \in V(\overline{L})$), and backward precedence constraints (i.e., $\widetilde{P}(\overline{L}) = \emptyset$).

4.6.2 Description of procedure GENR

Procedure GENR is based on the following observations.

O1 : any route $\ell \in \mathcal{R}_i$ passing through vertex i is the combination of a feasible forward path L and a feasible backward path $\overline{L}$ such that:

(a) $e(L) = e(\overline{L}) = i$,

(b) $\tau(L) \leqslant \tau(\overline{L})$,

(c) $V(L) \cap V(\overline{L}) = \{0, i\}$,

(d) $\widetilde{D}(\overline{L}) = \widetilde{P}(L')$, where L' is obtained from L removing the last vertex $e(L)$;

O2 : any feasible forward path L ending at a vertex $e(L) = i \in D$ and such that $\widetilde{P}(L) = \emptyset$ and $\tau_i + t_{i0} \leqslant b_0$ can be extended to a feasible route $R_\ell = L \cup \{(i, 0)\}$ having reduced cost $c'(\ell) = c'(L) + d'_{i,0}$;

O3 : let $LB(L)$ be a lower bound on the reduced cost of any route that contains a forward path L. Any forward path L such that $LB(L) > gap$ cannot be part of a route in the set $\mathcal{B}$ that satisfies conditions (4.30).

Let $\mathcal{T}$ be a set of *temporary* feasible forward paths that is initialized by setting $\mathcal{T} = \{L_0\}$, where L_0 represents the initial empty path such that $e(L_0) = 0$, $\tau(L_0) = a_0$ and $q(L_0) = 0$. The route set $\mathcal{B}$ is initialized by setting $\mathcal{B} = \emptyset$.

At each iteration of algorithm GENR the forward path $L^* \in \mathcal{T}$ having the smallest lower bound value (i.e., such that $LB(L^*) = \min\{LB(L) : L \in \mathcal{T}\}$) is extracted from the set $\mathcal{T}$. If $e(L^*) = 0$, then path L^* represents a feasible route and it is inserted in the set $\mathcal{B}$; otherwise, it is expanded. The expansion of a forward path L creates $2n$ new paths that are derived by extending L with arc $(e(L), j)$, $\forall j \in V \setminus \{e(L)\}$. All such paths that are not feasible are discarded.

Each feasible path L' that is obtained by expanding L^* and such that $LB(L') < gap$ is then added to the set $\mathcal{T}$. The reduced cost $c'(L')$ of each path L' is computed by setting:

$$c'(L') = c'(L^*) + d'_{e(L^*)e(L')} + \sum_{C \in \mathcal{C}'_{L'} \setminus \mathcal{C}'_{L^*}} v'_C. \tag{4.34}$$

Notice that the set $\mathcal{C}'_{L'} \setminus \mathcal{C}'_{L^*}$ corresponds to the set of request triplets having exactly two vertices in common with L', one of which is $e(L')$. Let $\mathcal{C}'_i \subseteq \mathcal{C}'$ be the subset of triplets involving a request $i \in P$. Then, the set $\mathcal{C}'_{L'} \setminus \mathcal{C}'_{L^*}$ can be computed in time $O(|\mathcal{C}'_{e(L')}|)$ by taking all the triplets in the set $\mathcal{C}'_{e(L')}$.

Procedure GENR terminates when either $\mathcal{T} = \emptyset$ or $|\mathcal{B}| = \gamma$. Since the size of the set $\mathcal{T}$ is exponential, we impose that the size of the set $\mathcal{T}$ cannot exceed an a-priori defined limit NSTATB. If $|\mathcal{T}|$ becomes greater than NSTATB, procedure GENR terminates prematurely.

4.6.3 Dominance rules

A significant speed-up in procedure GENR can be obtained by removing *dominated* paths from the set $\mathcal{T}$. A dominated path is either a path that cannot lead to a feasible route or a path such that any route containing it cannot be part of any optimal solution. In general, detecting dominated paths can be very time consuming and the computational trade-off between the speed-up resulting from

the reduced number of paths and the effort required to identify dominated paths must be carefully considered.

Let $d'(L) = \sum_{(i,j)\in L} d'_{ij}$ be the cost of path L with respect to the modified costs $\{d'_{ij}\}$. In our computational experiments we use the following dominance rules proposed by Dumas et al. [54] and Ropke and Cordeau [106].

Dominance rule 1: a forward path L_1 dominates a forward path L_2 if $e(L_1) = e(L_2)$, $V(L_1) = V(L_2)$, $\tau(L_1) \leqslant \tau(L_2)$ and $d'(L_1) \leqslant d'(L_2)$;

Dominance rule 2: a forward path L cannot lead to any feasible route if $\tau_{e(L)} + sht(e(L), j) > b_j$ for any delivery vertex j such that $j{-}n \in \widetilde{P}(L)$, where $sht(e(L), j)$ is the shortest traveling time between vertices $e(L)$ and j.

The following dominance rule 3 is used by GENR only in generating the sets $\mathcal{N}$ in H^2 and H^3.

Dominance rule 3: Let L_1 and L_2 be two forward paths and let ℓ_1 and ℓ_2 be the routes of minimum reduced cost c'_{ℓ_1} and c'_{ℓ_2} containing L_1 and L_2, respectively. If arc costs $\{d'_{ij}\}$ satisfy the condition $d'_{ij} + d'_{jk} \geqslant d'_{ik}$, $\forall i, k \in V$, $\forall j \in D$, and $e(L_1) = e(L_2)$, $\widetilde{P}(L_1) \subseteq \widetilde{P}(L_2)$, $P(L_1) \subseteq P(L_2)$, $\tau(L_1) \leqslant \tau(L_2)$ and $d'(L_1) \leqslant d'(L_2)$, then L_1 dominates L_2, because $c'_{\ell_1} \leqslant c'_{\ell_2}$.

Ropke and Cordeau [106] prove dominance rule 3 in case $\mathcal{C}'_{L_1} = \mathcal{C}'_{L_2} = \emptyset$, that is, when $d'(L) = c'(L)$ for any feasible forward path L. In our case this is not true because of clique inequalities. However, extending the proof to our case is straightforward. The validity of dominance rule 3 directly follows from the following proposition.

Proposition 5 *Let L_1 and L_2 be two feasible forward paths satisfying the conditions stated by dominance rule 3 and let ℓ_2 be the feasible route of minimum reduced cost c'_{ℓ_2} that contains L_2. Then, there exist a route ℓ_1 of cost c'_{ℓ_1} that contains L_1 and such that $c'_{\ell_1} \leqslant c'_{\ell_2}$.*

Proof: Let $\overline{L}_2$ be the feasible backward path that is obtained by removing the forward path L_2 from route ℓ_2 (that is, $\overline{L}_2$ is such that when appended after L_2 gives route ℓ_2). Consider a backward path $\overline{L}_1$ that is obtained from $\overline{L}_2$ by removing all delivery vertices whose corresponding pickup is in $\widetilde{P}(L_2) \setminus \widetilde{P}(L_1)$, and let ℓ_1 be the route obtained by appending $\overline{L}_1$ after L_1. It is quite obvious that ℓ_1 is a feasible route as travel times satisfy the triangle inequality and ℓ_1 services the same requests of ℓ_2 except those in $\widetilde{P}(L_2) \setminus \widetilde{P}(L_1)$. Notice that, since $P(L_1) \subseteq P(L_2)$ and $P(\overline{L}_1) = P(\overline{L}_2)$, we have $S_{\ell_1} \subseteq S_{\ell_2}$ and therefore $\mathcal{C}'_{\ell_1} \subseteq \mathcal{C}'_{\ell_2}$. Using the definition of costs $\{d'_{ij}\}$ the reduced cost c'_{ℓ_1} of route ℓ_1 can be written as:

$$c'_{\ell_1} = d'(L_1) + d'(\overline{L}_1) - \sum_{C \in \mathcal{C}'_{\ell_1}} \nu'_C \tag{4.35}$$

By hypotheses $d'(L_1) \leqslant d'(L_2)$ and since $d'_{ij} + d'_{jk} \geqslant d'_{ik}$, $\forall i, k \in V$, $\forall j \in D$, we also have $d'(\overline{L}_1) \leqslant d'(\overline{L}_2)$. Moreover, as $\mathcal{C}'_{\ell_1} \subseteq \mathcal{C}'_{\ell_2}$ and $\nu'_C \leqslant 0$, $\forall C \in \mathcal{C}'$, we have $\sum_{C \in \mathcal{C}'_{\ell_1}} \nu'_C \geqslant \sum_{C \in \mathcal{C}'_{\ell_2}} \nu'_C$. Therefore, from expression (4.35) we get:

$$c'_{\ell_1} \leqslant d'(L_2) + d'(\overline{L}_2) - \sum_{C \in \mathcal{C}'_{\ell_2}} \nu'_C = c'_{\ell_2} \tag{4.36}$$

□

Ropke and Cordeau [106] also provide a generalization of dominance rule 3 that is obtained by replacing the set $P(L)$ with a superset $U(L) \supseteq P(L)$ that contains, in addition to $P(L)$, all the pickup vertices that cannot be visited by any expansion of path L within their time window. Dominance rule 3 is a generalization of rule 1 and is therefore stronger. However, given a path L_2, finding all the paths $L_1 \in \mathcal{T}$ satisfying the conditions $\widetilde{P}(L_1) \subseteq \widetilde{P}(L_2)$ and $P(L_1) \subseteq P(L_2)$ can be very time consuming. Within GENR we use dominance 3 to test if a path L_2 can be fathomed only when L_2 is generated and we only look for dominating paths L_1 that are sub-paths of L_2.

Notice that proposition 5 ensures that using dominance rule 3 does not prevent GENR to generate the least reduced cost route of the set $\mathcal{B}$ defined by expressions (4.30). However, using this rule does not guarantee that all the routes

satisfying conditions (4.30) are generated. Therefore, dominance rule 3 cannot be used when generating the final route set $\widehat{\mathcal{R}}$ in the exact method. Moreover, using this rule could slow down the convergence of the column generation procedure by preventing some optimal routes to be generated at earlier iterations. Finally, since travel times must satisfy the condition $d'_{ij}+d'_{jk} \geqslant d'_{ik}, \forall i, k \in V, \forall j \in D$, dominance rule 3 cannot be used within a branch-and-bound scheme that branches on arcs. In our algorithm we only use dominance rule 3 when computing the lower bounds LH2 and LH3 on PDPTW-o1 problems and when computing lower bound LB_0 on PDPTW-o2 problems.

4.6.4 Computing Lower Bound $LB(L)$

In this section we describe two methods for computing the lower bound $LB(L)$ associated with a forward path L. These methods are based on two different relaxations of the backward paths that are combined with L to derive not necessarily feasible routes. The first relaxation is used for those PDPTW instances that are loosely capacity constrained, that is, when the ratio Q/q_{min} is large, where $q_{min} = \min_{i \in P}\{q_i\}$. The second relaxation dominates the first one but it is time consuming when the ratio Q/q_{min} is large. In our computational experiments we found computationally convenient to use the second relaxation only when $Q/q_{min} \leqslant 4$.

Relaxation based on (t, i)-paths

The first method is based on a relaxation of the backward paths called *backward* (t, i)*-path*. A backward (t, i)-path is a not-necessarily feasible backward path in G that starts from vertex $i \in V$ at time $a_i \leqslant t \leqslant b_i$, ends at the depot 0, and satisfies time window constraints.

Let $\bar{f}(t, i)$ be the cost of the least cost backward (t, i)-path starting from vertex i at time t with respect to the modified arc costs (4.31), and let $\bar{\pi}(t, i)$ be the vertex just after i. Moreover, let $\bar{g}(t, i)$ be the cost of the least cost backward (t, i)-path

starting from vertex i at time t and such that the vertex after i is different from $\bar{\pi}(t,i)$.

The functions $\bar{f}(t,i)$ and $\bar{g}(t,i)$ can be computed in pseudo-polynomial time (i.e., in time $O(T\,n^2)$, where $T = b_0 - a_0$) using a dynamic programming procedure that is an extension of a method proposed in Christofides et al. [33] for the CVRP. This method also allows to impose the restriction that a backward (t,i)-path cannot contain loops of two consecutive vertices. In computing the functions $\bar{f}(t,i)$ and $\bar{g}(t,i)$ we ignore the dual variables of inequalities (4.12). The dynamic programming recursion to compute functions $\bar{f}(t,i)$ and $\bar{g}(t,i)$ is a follows.

$$\bar{f}(t,i) = \min_{j \in V} \begin{cases} \bar{f}(\bar{\tau}(ijt), j) + d'_{ij}, & \text{if } i \neq \bar{\pi}(\bar{\tau}(ijt), j) \\ \bar{g}(\bar{\tau}(ijt), i) + d'_{ij}, & \text{otherwise} \end{cases}$$

where $\bar{\tau}(ijt) = \max\{a_j, t + t_{ij}\}$, and the functions $\bar{f}(t,i)$ and $\bar{g}(t,i)$ are initialized for each vertex $i \in V$ by setting:

$$\bar{f}(t,i) = \begin{cases} d'_{i0}, & t = a_i, \ldots, b_i \\ \infty, & \text{otherwise} \end{cases}$$

$$\bar{g}(t,i) = \infty, \quad t = a_0, \ldots, b_0$$

Notice that since backward (t,i)-paths are a relaxation of feasible backward paths and the duals of inequalities (4.12) are non-positive, the value $\bar{f}(t,i)$ represents a lower bound on the reduced cost of any feasible backward path starting in i at a time greater than or equal to t. Therefore a lower bound $LB_1(L)$ on the reduced cost of any route containing a forward path L can be computed as follows.

$$LB_1(L) = c'(L) + \min_{\tau(L) \leqslant t \leqslant b_{e(L)}} \begin{cases} \bar{f}(t, e(L)) & \text{if } \bar{\pi}(t,i) \notin V(L), \\ \bar{g}(t, e(L)) & \text{otherwise.} \end{cases} \tag{4.37}$$

The computation of $LB_1(L)$ within procedure GENR by means of expression (4.37) can be time consuming since the value $LB_1(L)$ must be computed each time a new path L generated. A better method for computing $LB_1(L)$ consists of avoiding the

minimization required in the right-hand-side of expression (4.37) as follows. Let $\overline{F}(t,i)$ be the cost of the least cost backward (t,i)-path starting from i at a time greater than or equal to t and let $\overline{\chi}(t,i)$ be the vertex right after i in such path. Similarly, let $\overline{G}(t,i)$ be the cost of the least cost (t,i)-path starting from i at a time greater than or equal to t and such that the vertex after i is not equal to $\overline{\chi}(q,i)$. Functions $\overline{F}(t,i)$, $\overline{\chi}(t,i)$ and $\overline{G}(t,i)$ can be computed before starting GENR using functions $\bar{f}(t,i)$, $\bar{g}(t,i)$ and $\bar{\pi}(t,i)$ as follows:

$$\overline{F}(t,i) = \min_{t \leqslant t' \leqslant b_i} \{\bar{f}(t',i)\}, \quad \forall i \in V,\ a_i \leqslant t \leqslant b_i, \tag{4.38}$$

and $\overline{\chi}(t,i) = \bar{\pi}(t^*,i)$, where t^* is the value of t giving the minimum in expression (4.38).

$$\overline{G}(t,i) = \min_{t \leqslant t' \leqslant b_i} \begin{cases} \bar{f}(t',i), & \text{if } \bar{\pi}(t',i) \neq \overline{\chi}(t,i), \\ \bar{g}(t',i), & \text{otherwise,} \end{cases} \quad \forall i \in V,\ a_i \leqslant t \leqslant b_i. \tag{4.39}$$

Using expressions (4.38) and (4.39) we have:

$$LB_1(L) = c'(L) + \begin{cases} \bar{F}(t, e(L)), & \text{if } \overline{\chi}(t,i) \notin V(L), \\ \bar{G}(t, e(L)), & \text{otherwise.} \end{cases} \tag{4.40}$$

It is straightforward to extend the above recursion to compute forward (t,i)-paths and the corresponding functions $f(t,i)$ and $g(t,i)$.

Relaxation based on $(\widetilde{D},t,i)$-paths

$(\widetilde{D},t,i)$-paths are a stronger relaxation than (t,i)-paths that were introduced by Dumas et al. [54]. This relaxation corresponds to a Shortest Path Problem with Time Windows, Capacity, and Pickup and Delivery which is also used in Ropke and Cordeau [106] and in Dumas et al. [54] as a pricing problem to generate non elementary routes.

$(\widetilde{D},t,i)$-paths are obtained from feasible backward paths by relaxing the requirement that the path must be simple. More precisely a backward $(\widetilde{D},t,i)$-path is a not-necessarily simple feasible backward path $\overline{L}$ that starts from i at

time t and such that $\widetilde{D}(\overline{L}) = \widetilde{D}$ (i.e., $\overline{L}$ visits the deliveries of requests $\widetilde{D}$ without having visited before their pickups). Moreover a backward $(\widetilde{D}, t, i)$-path must satisfy the condition that after visiting a delivery vertex the same delivery is not visited again before the corresponding pickup. This condition is trivially satisfied by any elementary backward path and, in general, forbids loops of two consecutive vertices. In Figure 4.1 it is shown a backward $(\widetilde{D}, t, i)$-path $\overline{L} = (1, n+3, n+2, n+1, 3, n+3, 0)$ with $\widetilde{D} = \{2, 3\}$ and $i = 1$.

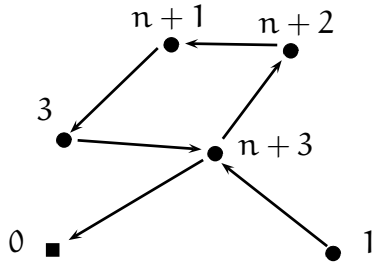

Figure 4.1: Example of $(\widetilde{D}, t, i)$-path with $\widetilde{D} = \{2, 3\}$

Consider any feasible forward path L and let L' be the sub-path which is obtained from L removing the last vertex $e(L)$. Combining L with any $(\widetilde{D}, t, i)$-path $\overline{L}$ such that $\widetilde{D} = \widetilde{P}(L')$, $i = e(L)$ and $t \geqslant \tau(L)$ gives a not necessarily simple circuit in G satisfying pairing, precedence, capacity and time window constraints. Let $f(\widetilde{D}, t, i)$ be the cost of the least cost $(\widetilde{D}, t, i)$-path, with respect to the modified arc costs (4.31), starting from i at time t. Then, a lower bound $LB_2(L)$ on the reduced cost of any route containing L can be computed as follows.

$$LB_2(L) = c'(L) + \min_{\substack{\tau(L) \leqslant t \leqslant b_{e(L)} \\ \widetilde{D} = \widetilde{P}(L')}} \{f(\widetilde{D}, t, e(L))\}. \tag{4.41}$$

The functions $f(\widetilde{D}, t, i)$ are computed using a backward dynamic programming procedure where each vertex of the state-space graph corresponds to a backward $(\widetilde{D}, t, i)$-path $\overline{L}$. In computing the reduced cost $c'(\overline{L})$ we ignore the dual contributions of the clique inequalities (4.12), that is, the reduced cost $c'(\overline{L})$ is defined as $c'(\overline{L}) = \sum_{(i,j) \in A(\overline{L})} d'_{ij}$. The recursion is similar to the one described in Section 4.6.2

except that it is backward and the set of vertices $V(\overline{L})$ associated with a path $\overline{L}$ is replaced with the set $\widetilde{D}(\overline{L})$. A step-by-step description of the dynamic programming procedure used for computing functions $f(\widetilde{D}, t, i)$, called GENB, is given in Section 4.6.5.

Notice that, since (t, i)-paths are a relaxation of forward paths, a lower bound $LB(\overline{L})$ on the reduced cost of the least cost circuit containing a $(\widetilde{D}, t, i)$-path $\overline{L}$ can be computed using functions $f(t, i)$ as follows:

$$LB(\overline{L}) = c'(\overline{L}) + \min_{a_{e(\overline{L})} \leqslant t \leqslant \tau(\overline{L})} \{f(t, e(\overline{L}))\} \tag{4.42}$$

Therefore in computing functions $f(\widetilde{D}, t, i)$, any $(\widetilde{D}, t, i)$-path $\overline{L}$ such that $LB(\overline{L}) > gap$ can be fathomed as, for any feasible forward path L, $LB(\overline{L}) > gap$ implies $c'(L) + c'(\overline{L}) > gap$.

Lower bound LB_2 is stronger than LB_1 but computing functions $f(\widetilde{D}, t, i)$ can be time consuming as, for any t, i and $\widetilde{D}$, the number of $(\widetilde{D}, t, i)$-paths grows exponentially with $|\widetilde{D}|$. However, the definition of $(\widetilde{D}, t, i)$-path implies that an upper bound on the value $|\widetilde{D}|$ can be computed as Q/q_{min}. Therefore, the ratio Q/q_{min} can be used as an estimate of the computing time required to compute functions $f(\widetilde{D}, t, i)$.

4.6.5 Dynamic programming algorithms GENR and GENB

In this Section we give a step-by-step description of the dynamic programming algorithms GENR and GENB that are used to compute feasible routes and backward $(\widetilde{D}, t, i)$-paths. Both algorithms use a set $\mathcal{T}$ of temporary feasible paths. Procedure GENB uses a straightforward adaptation of dominance rules 1 and 2 described in Section 4.6.3. For the sake of completeness we report here these rules.

Dominance rule 1′: a backward $(\widetilde{D}, t, i)$-path $\overline{L}_1$ dominates a backward $(\widetilde{D}, t, i)$-path $\overline{L}_2$ if $e(\overline{L}_1) = e(\overline{L}_2)$, $\widetilde{D}(\overline{L}_1) = \widetilde{D}(\overline{L}_2)$, $\tau(\overline{L}_1) \geqslant \tau(\overline{L}_2)$ and $d'(\overline{L}_1) \leqslant d'(\overline{L}_2)$;

Dominance rule $2'$: Let $sht(i,j)$ be the shortest traveling time between vertices i and j. A backward $(\widetilde{D}, t, i)$-path $\overline{L}$ can be fathomed if $\tau_{e(\overline{L})} - sht(j, e(\overline{L})) < a_j$ for any pickup vertex $j \in \widetilde{D}(\overline{L})$.

Step-by-step description of GENR

Step 1 Let $L_0 = (0)$ be the empty path containing only the depot. Set $\mathcal{T} = \{L_0\}$, $c'(L_0) = 0$, $e(L_0) = 0$, $\tau(L_0) = a_0$ and $q(L_0) = 0$. Compute lower bound $LB(L_0)$ as described in Section 4.6.4. Initialize $\mathcal{B} = \emptyset$.

Step 2 If $\mathcal{T} = \emptyset$, then Stop.

Step 3 Let $L^* \in \mathcal{T}$ be such that $LB(L^*) = \min[LB(L) : L \in \mathcal{T}]$. Update $\mathcal{T} = \mathcal{T} \setminus \{L^*\}$.

Step 4 If $e(L^*) = 0$ and $L^* \neq L_0$ then:

- Update $\mathcal{B} = \mathcal{B} \cup \{L^*\}$. If $|\mathcal{B}| = \gamma$ then stop, otherwise return to step 2.

Step 5 Let $i = e(L^*)$. For every path L_j obtained by appending arc (i,j) at the end of L^*, $\forall j \in V \setminus V(L^*)$, repeat the following Step 6.

Step 6 Compute $\tau(L_j) = \max[a_j, \tau(L^*) + t_{ij}]$ and $q(L_j) = q(L^*) + q_j$. Compute the cost $c'(L_j)$ using expression (4.34) and the lower bound $LB(L_j)$ as described in Section 4.6.4. Test if the expanded path L_j must be rejected because of one of the following tests:

A) *Feasibility test*. If L_j is not a feasible forward path, then reject it.

B) *Lower bound test*. If $LB(L_j) \geqslant gap$ then reject path L_j.

C) *Dominance test*. Check if L_j is dominated according to dominance rules 1, 2 and 3, described in section 4.6.3.

If L_j is not rejected by tests A, B or C, then remove from $\mathcal{T}$ any path L that is dominated by L_j according to dominance rule 1. (i.e., such that

$e(L) = e(L_j)$, $V(L) = V(L_j)$, $d'(L) \geqslant d'(L_j)$ and $\tau(L) \geqslant \tau(L_j)$).
If $|\mathcal{T}| < NSTATB$ then insert L_j in $\mathcal{T}$, otherwise Stop.

Step 7 Return to Step 2.

Step-by-step description of GENB

Step 1 Let $\overline{L}_0 = (0)$ be the empty path containing only the depot. Set $\mathcal{T} = \{\overline{L}_0\}$, $c'(\overline{L}_0) = 0$, $e(\overline{L}_0) = 0$ $\tau(\overline{L}_0) = b_0$ and $q(\overline{L}_0) = 0$. Compute lower bound $LB(\overline{L}_0)$ using expression (4.42).

Step 2 If $\mathcal{T} = \emptyset$, then Stop.

Step 3 Let $\overline{L}^* \in \mathcal{T}$ be such that $LB(\overline{L}^*) = \min[LB(\overline{L}) : \overline{L} \in \mathcal{T}]$. Update $\mathcal{T} = \mathcal{T} \setminus \{\overline{L}^*\}$ and set $f(\widetilde{D}(\overline{L}^*), \tau(\overline{L}^*), e(\overline{L}^*)) = c'(\overline{L}^*)$.

Step 4 Let $i = e(\overline{L}^*)$. For every path $\overline{L}_j$ obtained by appending arc (j, i) at the beginning of $\overline{L}^*$, $\forall j \in D \cup \widetilde{D}(\overline{L}^*)$, repeat the following Step 5.

Step 5 Compute $\tau(\overline{L}_j) = \min[b_j, \tau(\overline{L}^*) - t_{ji}]$ and $q(\overline{L}_j) = q(\overline{L}^*) - q_j$. Compute the cost $c'(\overline{L}_j) = c'(\overline{L}^*) + d'_{ji}$ and the lower bound $LB(\overline{L}_j)$ using expression (4.42). Compute the set $\widetilde{D}(\overline{L}_j)$ as follows:

$$\widetilde{D}(\overline{L}_j) = \begin{cases} \widetilde{D}(\overline{L}^*) \setminus \{j\} & \text{if } j \in \widetilde{D}(\overline{L}^*), \\ \widetilde{D}(\overline{L}^*) \cup \{j - n\} & \text{if } j \in D. \end{cases} \tag{4.43}$$

Test if the expanded path $\overline{L}_j$ must be rejected because of one of the following tests:

A) *Feasibility test* . If path $\overline{L}_j$ violates time window, capacity or backward precedence constraints then reject path $\overline{L}_j$. If $j \in D$ and $j - n \in \widetilde{D}(\overline{L}^*)$ then reject path $\overline{L}_j$.

B) *Lower bound test*. If $LB(\overline{L}_j) \geqslant gap$ then reject path $\overline{L}_j$.

C) *Dominance test*. Check if $\overline{L}_j$ is dominated according to dominance rules $1'$ and $2'$.

If $\overline{L}_j$ is not rejected by tests A, B or C, then remove from $\mathcal{T}$ any path $\overline{L}$ that is dominated by $\overline{L}_j$ according to dominance rule $1'$. (i.e., such that $e(\overline{L}) = e(\overline{L}_j)$, $\widetilde{D}(\overline{L}) = \widetilde{D}(\overline{L}_j)$, $d'(\overline{L}) \geqslant d'(\overline{L}_j)$ and $\tau(\overline{L}) \leqslant \tau(\overline{L}_j)$).
If $|\mathcal{T}| < NSTATB$ then insert $\overline{L}_j$ in $\mathcal{T}$, otherwise Stop.

Step 6 Return to Step 2.

4.7. Computational Results

All the algorithms described in this chapter were coded in Fortran 77 and compiled with the Intel Fortran 10.1 compiler. CPLEX 11.0 was used as the LP solver in procedure H^3 and as the integer programming solver in the exact method. All the experiments were performed on an Intel Xeon E5310 Workstation clocked at 1.6 GHz with 8 Gb RAM running Windows Server 2003 Enterprise x64 Edition.

In our computational experiments we considered the following two sets of PDPTW instances from the literature that were also considered by Ropke and Cordeau [106].

a) *Class 1 instances*. Class 1 instances were introduced in Ropke and Cordeau [106] and correspond to randomly generated PDPTW-o2 problems with up to 75 requests where the primary objective consists of minimizing the number of vehicles used to service the requests. We refer the reader to Ropke and Cordeau [106] for a detailed description of the characteristics of these instances. Class 1 instances are further partitioned into 4 subclasses with respect to time windows width and vehicle capacity, called AA, BB, CC and DD. All these instances are publicly available at `http://www.diku.dk/~sropke/`.

b) *Class 2 instances*. The second class of instances considered in this chapter corresponds to a subset of the instances introduced by Li and Lim [83]. These

instances are PDPTW-o1 problems, derived from the Solomon VRPTW data set [see 115], involving approximately 100 requests. Class 2 instances are divided into three subclasses, LR, LC and LRC (identified by the first three letters in each instance name) with respect to the customer distribution. The LC instances have a clustered distribution of customers, LR instances have a random distribution of customers and LRC instances are a mix of clustered and randomly distributed customers.

We also report computational results on 6 bigger instances involving approximately 500 requests and tight time windows. Class 2 instances are publicly available at `http://www.top.sintef.no/`.

For all the instances we compute the travel cost matrix using real-valued Euclidean distances, that is, each entry $\{d_{ij}\}$, (corresponding to the travel cost associated with arc (i,j)), is computed by setting $d_{ij} = \sqrt{x_i^2 + y_i^2}$, where x_i and y_i represent the planar coordinates of vertex $i \in V$. All travel times are set equal to the corresponding travel cost rounded down to the 3^{rd} decimal digit. Several preprocessing rules for tightening time windows and remove arcs that cannot be in a feasible solution of the PDPTW have been described by Dumas et al. [54], Cordeau [36] and Desrochers et al. [51]. All these rules have been implemented and used to obtain the computational results reported in this section.

Based on the results of several preliminary experiments to identify good parameter settings for our method, we decided to use the following:

- in procedure H^1: $Maxt1 = 300$, $\epsilon = 1.5$;
- in procedure H^2: $Maxt2 = 30$, $\epsilon = 0.3$, $Maxt3 = 100$, $\Delta^{min} = 5,000$ and $\Delta^a = 400$;
- in procedure H^3: $\Delta^a = 500$ for Class 1 instances, $\Delta^a = 200$ for Class 2 instances, $\Delta^{min} = 20,000$ and $\theta = 20$;
- in order to avoid out-of-memory errors, we impose that the size of the final route set $\hat{\Re}$ cannot contain more than $\Delta^{max} = 10^5$ routes, and in procedure

GENR we set NSTATB $= 50 \times 10^6$.

In our preliminary computational experiments we found that on most Class 1 instances adding inequalities (4.12) permitted only a slight increase of the lower bound quality at the root node, but resulted in a significant increase in the computing time spent by procedures H^3 and GENR. Therefore, we decided to use inequalities (4.12) only on Class 2 instances.

Tables 4.2 – 4.5 report the results obtained by the new exact method and by the branch-and-cut-and-price algorithm of Ropke and Cordeau [106] on Class 1 instances. Tables 4.6 – 4.9 report the results obtained on Class 2 instances. A time limit of $36,000$ seconds is imposed to the exact method described in this paper, whereas a time limit of $7,200$ seconds was used for the branch-and-cut-and-price of Ropke and Cordeau. Tables 4.2 – 4.9 report the following columns:

$z(UB)$: best upper bound value available in the literature;

z^*: cost of the best solution found by our exact algorithm; values in bold indicate an improvement of the best known upper bound in the literature;

$\%LB_{RC}$: percentage ratio of lower bound LB_{RC} obtained by Ropke and Cordeau at the root node, computed as $100.0\ LB_{RC}/z^*$;

t_{RC}: total computing time in seconds spent by Ropke and Cordeau;

t_{TOT} total computing time in seconds spent by the exact method;

Nodes: total number of nodes of the exact algorithms;

%LHx: percentage ratio of lower bound LHx at the root node achieved by procedure H^x on PDPTW-o1 instances;

$\%LB_0$: percentage ratio of the best lower bound LB_0 at the root node, achieved on PDPTW-o2 instances;

m_{UB}, m_{LB}: upper and lower bound on the number of vehicles in PDPTW-o2 instances;

t_{LB}: total computing time in seconds to compute the lower bound at the root node, i.e. the sequence LH1, LH2 and LH3 for PDPTW-o1 instances and LB_0 for PDPTW-o2 instances (see Section 4.5.3);

$|\hat{\Re}|$: total number of routes in the final route subset;

%Opt: percentage ratio of the best lower bound achieved by the exact method, a value of 100.000 indicates that the algorithm terminates with an optimal solution.

An entry tl under columns t_{RC} and t_{TOT} denotes that the algorithm terminates after reaching the imposed time limit. Moreover the line after each subclass reports the average values of $\%LB_{RC}$, $\%LB_0$, %LH3, t_{LB}, t_{RC} and t_{TOT} computed over all subclass instances solved within the imposed time limit by both algorithms.

Finally, Tables 4.10 and 4.11 report a summary of the computational results presented in Tables 4.2 – 4.9 divided by class. In each table we report for each subclass:

- the number of instances (#prob.);
- the average percentage ratio of the best lower bound at the root node computed over all instances solved within the imposed time limit by both algorithms (%LB);
- the number of instances solved to optimality (#Opt);
- the average computing time in second over all instances solved within the imposed time limit by both algorithms (Time).

The results reported in Tables 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8 and 4.9 for the branch-and-cut-and-price algorithm of Ropke and Cordeau are, on each instance, the best achieved by the algorithm using the pricing procedures SP1 and SP2. All the computing times reported for the algorithm of Ropke and Cordeau are relative

to an AMD Opteron 250 computer (2.4 GHz) running Linux. In order to allow a fair comparison between the running times of our exact method and the branch-and-cut-and-price algorithm of Ropke and Cordeau we used the CPU2000 benchmarks, reported by the Standard Performance Evaluation Corporation (SPEC 2005), to compare the speed of the machines used to run the two algorithms.

These benchmark are publicly available at `http://www.spec.org/cpu/results/`. Table 4.1 reports the integer and floating point benchmark scores, together with a normalized speed value, for the CPUs used to run the algorithms compared in this section. According to Table 4.1 we estimate that the exact method described in this chapter is run on a computer which is approximately 5-10% faster than that of Ropke and Cordeau.

Table 4.1: Comparison of computer speed

Author(s)	CPU	SPECint	SPECfp	Norm [(a)]
This thesis	Intel Xeon E5310 1.6 GHz	1,680	1,619	1.00
Ropke and Cordeau (2009)	AMD Opteron 250 2.4 Ghz	1,417	1,700	0.94

[(a)]Normalized CPU speed with respect to the average SPECint and SPECfp

The results on Class 1 instances reported in Tables 4.2, 4.3, 4.4 and 4.5 indicate that both the new lower bounds and the new exact method (see Section 4.5) proposed in this thesis are superior to the lower bound LB_{RC} and the branch-and-cut-and-price of Ropke and Cordeau [106].

Table 4.10 shows that the new exact algorithm is on average 9 times faster on PDPTW-o2 instances and solves 9 problems previously unsolved. Notice that our method can solve all but one instances of Class 1 and produces for the unsolved instance a lower bound that is close to the upper bound. Moreover, it finds for problem DD60 an optimal solution with 3 vehicles when the previous best known

upper bound involved 4 vehicles.

Tables 4.6 – 4.9 show that the proposed exact method outperforms the branch-and-cut-and-price algorithm of Ropke and Cordeau on Class 2 instances being on average 8 times faster. Moreover, it can solve for the first time 6 problems of Class 2 previously unsolved.

4.8. Summary

In this chapter we studied the pickup and delivery problem with time windows (PDPTW), where a given fleet of vehicles of identical capacity located at a central depot must service a set of customers requests at minimum cost, while satisfying time window, capacity, pairing and precedence constraints. We considered two variants of the problem, called PDPTW-o1 and PDPTW-o2, which differ for the objective function to be minimized.

In the PDPTW-o2 it is required to minimize first the number of vehicles and second the sum of route costs. In the PDPTW-o1, instead, it is required to minimize the route costs only. We developed an exact algorithm for the PDPTW-o1 that is based on a set partitioning formulation involving an exponential number of variables. Three bounding procedures are used to compute different dual solutions whose cost provides a valid lower bound on the problem. The exact method attempts to generate a reduced problem containing only variables having reduced cost smaller than the gap between a known upper bound and the best lower bound achieved by the algorithm. If the resulting problem is of moderate size it is solved by an integer programming solver; otherwise, a branch-and-cut-and-price algorithm is used to find an optimal solution. Finally, we described an exact algorithm for the PDPTW-o2 that decomposes the original PDPTW-o2 instance into a number of PDPTW-o1 instances each involving a different number of vehicles. The computational results on test problems from the literature show that the proposed algorithms outperform a recent branch-and-cut-and-price algorithm from the literature on both PDPTW-o1 and PDPTW-o2 problems. The

proposed algorithms are on average faster on the instances under consideration and can solve for the first time 15 instances previously unsolved.

Table 4.2: Class 1 instances: subclass AA

Class 1 instances			Ropke and Cordeau			Our Method							
						Lower bounds				Exact algorithm			
Name	z_{UB}	z^*	$\%LB_{RC}$	Nodes	t_{RC}	m_{LB}	m_{UB}	$\%LB_0$	t_{LB}	$\lvert\widehat{\mathcal{R}}\rvert$	Nodes	%Opt	t_{TOT}
AA30	31,119.1	31,119.1	84.126	3	6.5	3	3	99.998	3.2	278	–	100.000	3.2
AA35	31,299.8	31,299.8	84.447	5	15.9	3	3	99.961	5.4	Δ^{max}	38	100.000	24.2
AA40	31,515.9	31,515.9	100.000	1	13.4	3	3	99.970	6.6	Δ^{max}	18	100.000	26.1
AA45	31,759.8	31,759.8	100.000	1	15.9	3	3	100.000	10.2	–	–	100.000	10.2
AA50	41,775.0	41,775.0	81.648	7	72.0	4	4	99.960	9.6	Δ^{max}	14	100.000	97.6
AA55	41,907.8	41,907.8	88.272	3	65.6	4	4	99.997	9.6	1060	–	100.000	9.7
AA60	42,140.7	42,140.7	92.291	5	235.7	4	4	99.971	24.0	Δ^{max}	24	100.000	91.1
AA65	42,250.2	42,250.2	94.724	9	349.5	4	4	99.975	27.5	Δ^{max}	10	100.000	72.5
AA70	42,452.3	42,452.3	97.196	75	2,477.5	4	4	99.947	23.0	Δ^{max}	200	100.000	438.2
AA75	52,472.7	**52,461.6**	82.358	188	*tl*	5	5	99.981	357.2	Δ^{max}	108	100.000	6,442.4
Average			91.412		361.3			99.975	13.2				85.8

Table 4.3: Class 1 instances: subclass BB

Class 1 instances			Ropke and Cordeau			Our Method								
						Lower bounds				Exact algorithm				
Name	z_{UB}	z^*	$\%LB_{RC}$	Nodes	t_{RC}	m_{LB}	m_{UB}	$\%LB_0$	t_{LB}	$\lvert\widehat{\mathcal{R}}\rvert$	Nodes	%Opt	t_{TOT}	
BB30	31,086.3	31,086.3	73.257	3	6.0	3	3	100.000	2.5	–	–	100.000	2.5	
BB35	31,281.2	31,281.2	86.564	5	18.0	3	3	99.993	5.0	881	–	100.000	5.1	
BB40	31,493.4	31,493.4	96.224	3	18.9	3	3	99.996	3.9	9702	–	100.000	4.8	
BB45	41,555.1	41,555.1	78.823	9	46.3	4	4	99.994	6.9	2991	–	100.000	7.2	
BB50	41,701.0	41,701.0	86.160	11	159.9	4	4	99.994	6.3	7569	–	100.000	9.6	
BB55	41,885.7	41,885.7	94.976	3	55.8	4	4	100.000	8.2	–	–	100.000	8.2	
BB60	62,420.1	62,420.1	87.994	27	181.7	6	6	99.995	9.9	Δ^{max}	14	100.000	17.6	
BB65	62,639.1	62,639.1	89.259	25	294.8	6	6	99.992	14.1	Δ^{max}	40	100.000	44.0	
BB70	62,951.0	62,951.0	97.755	139	1,839.4	6	6	99.939	13.4	Δ^{max}	108	100.000	418.1	
BB75	63,127.5	63,127.5	98.582	265	*tl*	6	6	99.822	17.7	Δ^{max}	29	99.892	1,055.3	a
Average			87.890		291.2			99.989	7.8				57.4	

a: branch-and-cut-and-price terminates prematurely due to memory overflow in GENR

Table 4.4: Class 1 instances: subclass CC

Class 1 instances			Ropke and Cordeau			Our Method							
						Lower bounds				Exact algorithm			
Name	z_{UB}	z^*	$\%LB_{RC}$	Nodes	t_{RC}	m_{LB}	m_{UB}	$\%LB_0$	t_{LB}	$\lvert\widehat{\mathcal{R}}\rvert$	Nodes	%Opt	t_{TOT}
CC30	31,087.7	31,087.7	73.396	5	11.3	3	3	99.993	3.8	533	–	100.000	3.9
CC35	31,230.6	31,230.6	77.673	17	59.1	3	3	99.983	5.0	3289	–	100.000	5.8
CC40	31,358.5	31,358.5	80.660	29	254.8	3	3	99.976	8.2	Δ^{max}	40	100.000	26.4
CC45	31,509.1	31,509.1	91.844	11	175.9	3	3	99.983	11.6	Δ^{max}	12	100.000	17.0
CC50	41,685.3	41,685.3	81.703	137	1,962.3	4	4	99.981	13.7	Δ^{max}	56	100.000	38.9
CC55	41,836.3	41,836.3	87.083	117	2,729.2	4	4	99.970	16.7	Δ^{max}	288	100.000	185.2
CC60	42,015.5	**42,009.3**	90.075	162	*tl*	4	4	99.943	17.8	Δ^{max}	2294	100.000	2,128.2
CC65	42,172.1	**42,164.0**	93.635	85	*tl*	4	4	99.929	20.8	Δ^{max}	4884	100.000	7,111.2
CC70	52,201.9	**52,201.7**	80.696	44	*tl*	4	5	81.137	78.7	Δ^{max}	899	100.000	5,565.7
CC75	52,375.6	**52,359.0**	83.204	47	*tl*	5	5	99.978	52.1	Δ^{max}	148	100.000	259.6
Average			82.060		865.4			99.981	9.9				46.2

Table 4.5: Class 1 instances: subclass DD

Class 1 instances			Ropke and Cordeau			Our Method							
						Lower bounds				Exact algorithm			
Name	z_{UB}	z^*	$\%LB_{RC}$	Nodes	t_{RC}	m_{LB}	m_{UB}	$\%LB_0$	t_{LB}	$\lvert\widehat{\mathcal{R}}\rvert$	Nodes	%Opt	t_{TOT}
DD30	21,133.3	21,133.3	100.000	1	25.2	2	2	99.956	5.4	10929	–	100.000	7.1
DD35	31,210.9	31,210.9	69.271	143	765.1	3	3	99.982	5.1	2219	–	100.000	5.4
DD40	31,352.2	31,352.2	73.847	15	160.8	3	3	99.983	7.5	9837	–	100.000	13.7
DD45	31,483.9	31,483.9	79.026	31	525.6	3	3	99.972	12.0	Δ^{max}	70	100.000	62.9
DD50	31,600.9	31,600.9	84.155	77	1,976.0	3	3	99.958	19.5	Δ^{max}	92	100.000	96.7
DD55	31,743.3	31,743.3	90.843	25	1,178.5	3	3	99.971	18.0	Δ^{max}	14	100.000	36.5
DD60	41,869.4	**32,069.2**	98.121	88	*tl*	3	4	99.685	24.1	Δ^{max}	2158	100.000	13,048.3
DD65	42,125.7	**42,107.3**	83.866	59	*tl*	4	4	99.935	23.6	Δ^{max}	10152	100.000	25,929.1
DD70	42,220.3	**42,214.2**	86.915	55	*tl*	4	4	99.943	75.7	Δ^{max}	7542	100.000	20,737.5
DD75	42,396.8	**42,359.9**	91.507	42	*tl*	4	4	99.937	113.7	Δ^{max}	4510	100.000	34,718.6
Average			82.857		771.9			99.970	11.3				37.1

Table 4.6: Class 2 instances: subclass LC1

Class 2 instances			Ropke and Cordeau			Our method							
						Lower bounds					Exact method		
Name	z_{UB}	z^*	$\%LB_{RC}$	Nodes	t_{RC}	$\%LH_1$	$\%LH_2$	$\%LH_3$	t_{LB}	$\lvert\widehat{\mathcal{R}}\rvert$	%Opt	t_{TOT}	
lc1_2_1	2,704.6	2,704.6	100.000	1	4.9	99.844	100.000	100.000	3.3	–	100.000	3.3	
lc1_2_2	2,764.6	2,764.6	100.000	1	27.6	97.241	100.000	100.000	21.5	–	100.000	21.5	
lc1_2_3	2,772.2	2,772.2	100.000	241	240.6	92.682	100.000	100.000	114.9	–	100.000	114.9	
lc1_2_4	2,661.4	2,661.4	–	–	–	90.015	–	–	454.2	–	90.015	454.2	α
lc1_2_5	2,702.0	2,702.0	100.000	1	6.3	99.844	100.000	100.000	4.8	–	100.000	4.8	
lc1_2_6	2,701.0	2,701.0	100.000	1	9.8	99.844	100.000	100.000	7.4	–	100.000	7.4	
lc1_2_7	2,701.0	2,701.0	100.000	1	10.9	98.774	100.000	100.000	7.7	–	100.000	7.7	
lc1_2_8	2,689.8	2,689.8	100.000	1	31.5	97.248	99.961	100.000	16.0	–	100.000	16.0	
lc1_2_9	2,724.2	2,724.2	99.684	91	6,628.6	92.007	99.629	100.000	55.3	–	100.000	55.3	
lc1_2_10	2,741.6	2,741.6	99.756	9	–	90.278	99.729	100.000	137.1	–	100.000	137.1	
Average			99.961		870.0			100.000	28.8			28.8	

α: H^2 terminates prematurely due to memory overflow in GENR

Table 4.7: Class 2 instances: subclass LR1

Class 2 instances			Ropke and Cordeau			Our method							
						Lower bounds					Exact method		
Name	z_{UB}	z^*	$\%LB_{RC}$	Nodes	t_{RC}	$\%LH_1$	$\%LH_2$	$\%LH_3$	t_{LB}	$\lvert\widehat{\mathcal{R}}\rvert$	%Opt	t_{TOT}	
lr1_2_1	4,819.1	4,819.1	100.000	1	5.1	98.046	100.000	100.000	1.6	–	100.000	1.6	
lr1_2_2	4,093.1	4,093.1	100.000	1	84.0	96.912	100.000	100.000	20.6	–	100.000	20.6	
lr1_2_3	3,486.9	**3,486.8**	99.923	2	–	89.316	99.898	100.000	3,690.8	–	100.000	3,690.8	
lr1_2_4	2,830.7	2,830.7	–	–	–	82.727	–	–	1,809.6	–	82.727	1,809.6	α
lr1_2_5	4,221.6	4,221.6	100.000	1	11.7	95.700	99.977	100.000	2.6	–	100.000	2.6	
lr1_2_6	3,763.0	3,763.0	100.000	1	1,041.6	91.270	100.000	100.000	180.9	–	100.000	180.9	
lr1_2_7	3,112.9	3,112.9	–	–	–	88.719	–	–	1,320.4	–	88.719	1,320.4	α
lr1_2_8	2,645.5	2,645.5	–	–	–	81.299	–	–	566.9	–	81.299	566.9	α
lr1_2_9	3,953.5	3,953.5	99.820	25	418.5	90.580	99.342	100.000	15.4	–	100.000	15.4	
lr1_2_10	3,389.2	**3,386.3**	99.702	4	–	84.909	99.689	100.000	1,376.7	–	100.000	1,376.7	
Average			99.964		312.2			100.000	44.2			44.2	

α: H^2 terminates prematurely due to memory overflow in GENR

Table 4.8: Class 2 instances: subclass LRC1

Class 2 instances			Ropke and Cordeau			Our method							
						Lower bounds				Exact method			
Name	z_{UB}	z^*	$\%LB_{RC}$	Nodes	t_{RC}	$\%LH_1$	$\%LH_2$	$\%LH_3$	t_{LB}	$\lvert\widehat{\mathcal{R}}\rvert$	%Opt	t_{TOT}	
lrc1_2_1	3,606.1	3,606.1	100.000	1	12.6	95.081	99.919	100.000	3.1	–	100.000	3.1	
lrc1_2_2	3,292.4	3,292.4	99.830	3	1,053.3	90.830	99.143	100.000	322.3	–	100.000	322.3	
lrc1_2_3	3,079.5	3,079.5	–	–	–	81.112	–	–	304.3	–	81.112	304.3	a
lrc1_2_4	2,525.8	2,525.8	–	–	–	78.426	–	–	188.1	–	78.426	188.2	a
lrc1_2_5	3,715.8	3,715.8	99.868	17	517.8	83.254	99.838	100.000	42.1	–	100.000	42.1	
lrc1_2_6	3,360.9	3,360.9	100.000	1	27.7	89.202	100.000	100.000	7.0	–	100.000	7.0	
lrc1_2_7	3,317.7	3,317.7	99.412	81	–	84.235	99.298	99.941	393.6	1396	100.000	408.2	
lrc1_2_8	3,086.7	**3,086.5**	98.033	2	–	83.390	97.972	99.733	1,444.1	4697	100.000	1,562.7	
lrc1_2_9	3,058.6	**3,053.8**	98.091	2	–	82.816	97.661	99.360	1,194.5	45537	100.000	1,757.2	
lrc1_2_10	2,837.5	2,837.5	–	–	–	82.308	–	–	217.4	–	82.308	217.4	a
Average			99.925		402.9			100.000	93.6			93.6	

a: H^2 terminates prematurely due to memory overflow in GENR

Table 4.9: Class 2 instances: subclass L500 (500 requests and tight time windows)

Class 2 instances			Ropke and Cordeau			Our method								
						Lower bounds				Exact method				
Name	z_{UB}	z^*	$\%LB_{RC}$	Nodes	t_{RC}	$\%LH_1$	$\%LH_2$	$\%LH_3$	t_{LB}	$\lvert\widehat{\mathcal{R}}\rvert$	Nodes	%Opt	t_{TOT}	
lc1101	42,488.7	42,488.7	100.000	1	704.8	99.576	100.000	100.000	79.5	–	–	100.000	79.5	
lc1105	42,477.4	42,477.4	100.000	1	762.7	98.997	100.000	100.000	118.7	–	–	100.000	118.7	
lr1101	56,744.9	56,744.9	99.993	3	1,682.3	93.824	99.988	100.000	233.1	–	–	100.000	233.1	
lr1105	52,901.3	52,901.3	99.055	8	–	83.639	98.875	99.308	3,532.3	–	1	99.308	4,068.8	b
lrc1101	48,666.5	48,666.5	99.039	9	–	83.466	98.563	99.448	930.0	–	1	99.448	2,533.3	b
lrc1105	49,287.1	49,287.1	–	–	–	77.464	–	–	1,650.3	–	–	77.464	1,650.3	a
Average			99.998		1,049.9			100.000	143.8				143.8	

a: H^2 terminates prematurely due to memory overflow in GENR

b: branch-and-cut-and-price terminates prematurely due to memory overflow in GENR

Table 4.10: Summary results of Class 1 instances

		Ropke and Cordeau			**Our method**		
Subclass	# prob.	%LB	# Opt	Time	%LB	# Opt	Time
AA	10	91.412	9	361.3	99.975	10	85.8
BB	10	87.890	9	291.2	99.989	9	57.4
CC	10	82.060	6	865.4	99.981	10	46.2
DD	10	82.857	6	771.9	99.970	10	37.1
	40	86.055	30	572.5	99.979	39	56.6

Table 4.11: Summary results of Class 2 instances

		Ropke and Cordeau			**Our method**		
Subclass	# prob.	%LB	# Opt	Time	%LB	# Opt	Time
LC1	10	99.961	8	870.0	100.000	9	28.8
LR1	10	99.964	5	312.2	100.000	7	44.2
LRC1	10	99.925	4	402.9	100.000	7	93.6
LL500	6	99.998	3	1,049.9	100.000	3	143.8
	36	99.962	20	658.7	100.000	26	77.6

Chapter 5

Conclusions

We studied three combinatorial optimization problems belonging to the classes of Network Design and Vehicle Routing Problems: the Non-Bifurcated Capacitated Network Design Problem (NBP), the Period Vehicle Routing Problem (PVRP), and the Pickup and Delivery Problem with Time Windows (PDPTW). We presented both new heuristic and exact algorithms for solving such problems and we tested their effectiveness on different classes of instances from the literature.

For the NBP we proposed an exact algorithm based on solving a mathematical formulation of the problem that is strengthened by adding only a subset of two well-known classes of valid inequalities, and four new heuristics. The computational results showed that the general purpose integer programming solver CPLEX when solving this strengthened formulation is competitive with a recent branch-and-cut algorithm from the literature. Nevertheless, both algorithms were unable to solve instances of small size, whereas the new heuristic algorithms, though on average outperformed on small instances, obtained the best results on the bigger instances. It is interesting to notice that on bigger instances the best results were obtained by algorithm F&B that represents, in contrast to the other proposed heuristics, a more general framework for solving a wider class of combinatorial problems.

For the Period Routing Problem, for which neither lower bounds nor exact algorithms have been proposed in the literature so far, we proposed an exact algo-

rithm based on different relaxations of the mathematical formulation of the problem. We tested our algorithm on a set of instances from the literature involving up to 153 customers for which only heuristic solutions were available. The algorithm was able to obtain for the first time the optimal solution on several test instances and to improve some of the previous best known upper bounds. Moreover, the algorithm was able to achieve tight lower bounds, on average within one percent of optimality, that permitted to estimate the quality of the best known upper bound for those instances which are still unsolved. We also considered a variant of this problem, called Tactical Planning VRP, that arises in food and beverage distribution systems and in field force planning such as maintenance or service logistics activities undertaken by utility companies. This problem represents a more tactical problem than the classical Period Vehicle Routing where customers specify a visit day over the planning period and a dissatisfaction service cost is incurred if they are visited at a later day than required. We generated a set of Tactical Planning VRP instances and we compared the best solutions found by the exact algorithm with and without service costs. The results showed that taking into account service costs permitted solutions of better quality with respect to the customer service level at the expense of only a slight increase in the routing costs. Moreover, the exact algorithm was able to solve to optimality 28 out of the 40 Tactical Planning instances and all instances involving up to 100 customers could be solved to optimality.

For the Pickup and Delivery Problem with Time Windows we considered two variants of the problem with respect to the objective function to minimize. One such variant, called PDPTW-o2, has fixed costs associated with each vehicle and requires to minimize the sum of fixed costs and route costs, whereas the other is called PDPTW-o1 and requires to minimize the route costs only. We presented an exact algorithm for the PDPTW-o1 based on three different bounding procedures to compute a final lower bound on the problem and a branch-and-cut-and-price algorithm where the pricing subproblem is solved by a dynamic programming procedure that generates feasible routes. We also described a different exact al-

gorithm for the second variant that decomposes the original PDPTW-o2 instance into a number of PDPTW-o1 instances, each involving a different number of vehicles. The results obtained on two classes of problems from the literature showed that the new algorithms outperformed the best algorithm presented in the literature so far. The proposed strategy for solving PDPTW-o2 problems turned out to be superior to the standard approach adopted by this latter algorithm that consists in adding the fixed costs to the outgoing arcs from the depot. In particular, the new algorithm ran on average 9 times faster on PDPTW-o2 instances, achieved tighter lower bounds, and solved to optimality 9 PDPTW-o2 problems that were previously unsolved. The new algorithm also obtained better results on PDPTW-o1 instances being on average 8 times faster than the best algorithm from the literature and solving for the first time 6 PDPTW-o1 problems previously unsolved.

Bibliography

[1] Y. K. Agarwal. Design of capacitated multicommodity networks with multiple facilities. *Operations Research*, 50:333–344, 2002.

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

[3] S. Anily and J. Bramel. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. *Naval Research Logistics*, 46:654–670, 1999.

[4] A. Armacost, C. Barnhart, and K. Ware. Composite variable formulations for express shipment service network design. *Transportation Science*, 36:1–20, 2002.

[5] N. Ascheuer, M. Fischetti, and M. Grötschel. A polyhedral study of the asymmetric traveling salesman problem with time windows. *Networks*, 36: 69–79, 2000.

[6] A. Atamtürk and D. Rajan. On splittable and unsplittable flow capacitated network design arc-set polyhedra. *Mathematical Programming*, 92:315–333, 2002.

[7] P. Augerat, J. Belenguer, E. Benavent, A. Corberán, and D. Naddef. Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, 106:546–557, 1998.

[8] P. Avella, S. Mattia, and A. Sassano. Metric inequalities and the network loading problem. *Discrete Optimization*, 4:103 – 114, 2007.

[9] A. Balakrishnan, T. L. Magnanti, and R. T. Wong. A dual-ascent procedure for large scale uncapacitated network design. *Operations Research*, 37:716–740, 1989.

[10] A. Balakrishnan, T. Magnanti, and P. Mirchandani. "network design". In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 311–334. New York: Wiley, 1997.

[11] N. Balakrishnan. Simple heuristics for the vehicle routeing problem with soft time windows. *The Journal of the Operational Research Society*, 44:279–287, 1993.

[12] E. Balas and M. W. Padberg. Set partitioning: A survey. *SIAM Review*, 18: 710–760, 1976.

[13] R. Baldacci and A. Mingozzi. A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming Ser. A*, 2008. URL `http://dx.doi.org/10.1007/s10107-008-0218-9`.

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

[15] R. Baldacci, L. D. Bodin, and A. Mingozzi. The multiple disposal facilities and multiple inventory locations rollon-rolloff vehicle routing problem. *Computers & Operations Research*, 33:2667–2702, 2006.

[16] R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming Ser. A*, 115(2):351–385, 2008.

[17] F. Barahona. Network design using cut inequalities. *SIAM Journal on Optimization*, 6:823–837, 1996.

[18] F. Barahona. On the k-cut problem. *Operations Research Letters*, 26:99–105, 2000.

[19] C. Barnhart, C. Hane, and P. Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48:318–326, 2000.

[20] J. Beasley and C. Tan. A heuristic algorithm for the period routing problem. *Omega*, 12:497–504, 1984.

[21] G. Berbeglia, J. F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *Top*, 15(1):1–31, July 2007.

[22] D. Berger, B. Gendron, J.-Y. Potvin, S. Raghavan, and P. Soriano. Tabu search for a network loading problem with multiple facilities. *Journal of Heuristics*, 6:253–267, 2000.

[23] D. Bienstock and O. Günlük. Capacitated network design – polyhedral structure and computation. *INFORMS J. on Computing*, 8:243–259, 1996.

[24] D. Bienstock, S. Chopra, O. Günlük, and C.-Y. Tsai. Minimum cost capacity installation for multicommodity network flows. *Matemathical Programming*, 81:177–199, 1998.

[25] M. Bossert and T. L. Magnanti. Pup matching: Model formulations and solution approaches. Working Paper OR 366-03, Operations Research Center, Massachusset Institute of Technology, 2003.

[26] N. Bostel, P. Dejax, P. Guez, and F. Tricoire. Multiperiod planning and routing on a rolling horizon for field force optimization logistics. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43. Springer Verlag, 2008.

[27] B. Brockmüller, O. Günlük, and L. Wolsey. Designing private line networks - polyhedral analysis and computation. Discussion Paper 9647, Center for Operations Research and Econometrics, Universite Catolique de Louvain, 1996.

[28] B. Brockmüller, O. Günlük, and L. Wolsey. Designing private line networks - polyhedral analysis and computation. Discussion Paper 9647 revised, Center for Operations Research and Econometrics, Universite Catolique de Louvain, 1998.

[29] M. Carter, J. M. Farolden, G. Laporte, and J. Xu. Solving an integrated logistics problem arising in grocery distribution. *INFOR*, 34:290–306, 1996.

[30] I. M. Chao, B. L. Golden, and E. Wasil. An improved heuristic for the period vehicle-routing problem. *Networks*, 26(1):25–44, 1995.

[31] N. Christofides and J. Beasley. The period routing problem. *Networks*, 14: 237–256, 1984.

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

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

[34] P. C. Chu and J. E. Beasley. Constraint handling in genetic algorithms: the set partitioning problem. *Journal of Heuristics*, 4:323–357, 1998.

[35] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[36] J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[37] J.-F. Cordeau and G. Laporte. The dial-a-ride problem: Models and algorithms. *Annals of Operations Research*, 152:29–46, 2007.

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

[39] J.-F. Cordeau, G. Laporte, J.-Y. Potvin, and M. Savelsbergh. Transportation on demand. In C. Barnhart and G. Laporte, editors, *Transportation*, Handbooks in Operations Research and Management Science, pages 429–466. Elsevier, Amsterdam, 2007.

[40] J.-F. Cordeau, G. Laporte, and S. Ropke. Recent models and algorithms for one-to-one pickup and delivery problems. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43, pages 327–357. Springer, 2008.

[41] T. M. Corsi. The truckload carrier industry segment. In D. Belman and C. White III, editors, *Trucking in the age of information*. Aldershot: Ashgate, 2005.

[42] A. M. Costa, J.-F. Cordeau, and B. Gendron. Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications*, 2007. doi: 10.1007/s10589-007-9122-0.

[43] CPLEX. *ILOG CPLEX 11.0 callable library*. ILOG, 2008.

[44] T. Crainic. Service network design in freight transportation. *European Journal of OperationaI Research*, 122:272–288, 2000.

[45] T. Crainic, M. Gendreau, and J. Farvolden. A simplex-based tabu search method for capacitated network design. *INFORMS J. on Computing*, 12(3): 223–236, 2000.

[46] T. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112:73 – 99, 2001.

[47] T. G. Crainic. Long haul freight transportation. In R. Hall, editor, *Handbook of Transportation Science 2nd edition*, pages 451–516. Kluwer Academic Publishers, Boston, Massachusetts, 2003.

[48] G. Dahl and M. Stoer. A polyhedral approach to multicommodity survivable network design. *INFORMS J. on Computing*, 10:1–11, 1998.

[49] G. Dantzing and J. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

[50] G. Desaulniers, J. Desrosiers, A. Erdmann, M. Solomon, and F. Soumis. *VRP with Pickup and Delivery*, volume 9, chapter 9, pages 225–242. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.

[51] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342 – 354, 1992.

[52] J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-89-85, Computer Science Department, University of Tennessee, Knoxville, TN, 2007.

[53] S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1:195–207, 1971.

[54] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54(1):7–22, Sept. 1991.

[55] J. A. Ferland and L. Fortin. Vehicle routing with sliding time-windows. *European Journal of OperationaI Research*, 38:213–226, 1989.

[56] P. M. Francis, K. R. Smilowitz, and M. Tzur. The period vehicle routing problem and its extensions. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43. Springer Verlag, 2008.

[57] R. Fukasawa, H. Longo, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, and R. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming Ser. A*, 106:491–511, 2006.

[58] M. Gaudioso and G. Paletta. A heuristic for the periodic vehicle-routing problem. *Transportation Science*, 26(2):86–92, 1992.

[59] B. Gavish and K. Altinkemer. Backbone network design tools with economic tradeoffs and transportation planning: Models and algorithms. *ORSA Journal on Computing*, 2:236–252, 1990.

[60] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40: 1086–1094, 1992.

[61] B. Gendron and T. G. Crainic. Bounding procedures for multicommodity capacitated network design problems. *Publication CRT-96-06, Centre de Reserche sur les Transports, Universit de Montral, Montral, QC, Canada*, 1996.

[62] I. Ghamlouche, T. G. Crainic, and M. Gendreau. Cycle-based neighbourhoods for fixed-charge capacitated multicommodity network design. *Operations Research*, 51:655 – 667, 2003.

[63] I. Ghamlouche, T. Crainic, and M. Gendreau. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations Research*, 131:109–133, 2004.

[64] F. Glover. A template for scatter search and path relinking. In J. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution*, volume 1363 of *Lecture Notes in Computer Science*, pages 13 – 54. Berlin: Springer, 1997.

[65] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic, Boston, 1997.

[66] D. E. Goldberg. Genetic algorithms. In *Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[67] I. Gribkovskaia and G. Laporte. One-to-many-to-one single vehicle pickup and delivery problems. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43, pages 359–377. Springer, 2008.

[68] O. Günlük. A branch-and-cut algorithm for capacitated network design problems. *Matemathical Programming*, 86:17–39, 1999.

[69] S. L. Hakimi. Steiner's problem in a graph and its implications. *Networks*, 1:113–133, 1971.

[70] V. C. Hemmelmayr, K. F. Doerner, and R. F. Hartl. A variable neighborhood search heuristic for periodic routing problems. *European Journal of Operational Research*, 2007. doi: 10.1016/j.ejor.2007.08.048.

[71] H. Hernandez-Perez and J. S. Salazar-Gonzalez. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126–139, Dec. 2004.

[72] K. Hoffman and M. W. Padberg. Solving airline crew scheduling problems by branch and cut. 39:657–682, 1993.

[73] K. Holmberg and D. Yuan. A lagrangean heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48:461–481, 2000.

[74] K. Holmberg, M. Ronnqvist, and D. Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113:544–559, 1999.

[75] M. Iri. On an extension of the max-flow min-cut theorem to multicommodity flows. *J. Oper. Res. Soc. Jpn*, 13:129–135, 1971.

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

[77] R. M. Karp. *Complexity of Computer Computations*, chapter Reducibility Among Combinatorial Problems, pages 85 – 104. Plenum Press, 1972.

[78] N. Kohl, J. Desrosiers, O. B. G. Madsen, M. Solomon, and F. Soumis. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33:101–116, 1999.

[79] I. Kousik, D. Ghosh, and I. Murthy. A heuristic procedure for leasing channels in telecommunications networks. *Journal of the Operational Research Society*, 44:659–672, 1993.

[80] F. Lafontaine and L. M. Valeri. The deregulation of international trucking in the european union: form and effect. *Journal of Regulatory Economics*, 35: 19–44, 2009.

[81] G. Laporte, Y. Nobert, and D. Arpin. Optimal solutions to capacitated multi depot vehicle routing problem. *Congressus Numerantium*, 44:283–292, 1984.

[82] G. Laporte, Y. Nobert, and S. Taillefer. Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation Science*, 22: 161–172, 1988.

[83] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. In *13th IEEE International Conference on Tools with Artificial Intelligence,ICTAI-2001,Dallas,USA.*, 2001.

[84] Q. Lu and M. Dessouky. An exact algorithm for the multiple vehicle pickup and delivery problem. *Transportation Science*, 38(4):503–514, Nov. 2004.

[85] J. Lysgaard. CVRPSEP: A package of separation routines for the capacitated vehicle routing problem. Technical report, Dept. of Mgt. Science and Logistics, Aarhus School of Business, 2003.

[86] J. Lysgaard. Reachability cuts for the vehicle routing problem with time windows. *European Journal of Operational Research*, 175:210 – 223, 2006.

[87] J. Lysgaard, A. N. Letchford, and R. W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming Ser. A*, 100:423–445, 2004.

[88] T. Magnanti and R. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, 18(1):1–55, 1986.

[89] T. Magnanti, P. Mirchandani, and R. Vachani. Modeling and solving the two-facility capacitated network loading problem. *Operations Research*, 43 (1):142–157, 1995.

[90] T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, 18:1–55, 1984.

[91] T. L. Magnanti, P. Mirchandani, and R. Vachani. The convex hull of two core capacitated network design problems. *Mathematical Programming*, 60: 233–250, 1993.

[92] M. Minoux. Network syntesys and optimum network design problems: Models, solution methods and applications. *Networks*, 19:313–360, 1989.

[93] M. Mourgaya and F. Vanderbeck. The periodic vehicle routing problem: classification and heuristic. *Rairo-Operations Research*, 40(2):169–194, 2006.

[94] M. Mourgaya and F. Vanderbeck. Column generation based heuristic for tactical planning in multi-period vehicle routing. *European Journal of Operational Research*, 183(3):1028–1041, 2007.

[95] D. Naddef and G. Rinaldi. Branch-and-cut algorithms for the capacitated vrp. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, pages 53–84. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002.

[96] S. Niskanen and P. R. J. Östergård. Cliquer user's guide. Technical Report 48, Helsinki University of Technology Communications Laboratory, 2003.

[97] European Union. *Transport in figures, statistical pocketbook*. Office for the official publications of the European communities, 1999, 2000.

[98] European Commission. *Panorama of transport: Statistical overview of transport in the European Union*. Office for the official publications of the European communities, Luxembourg, 2003.

[99] K. Onaga and O. Kakusho. On feasibility conditions of multicommodity flows in networks. *IEEE Trans. Circuit Theory CT-18*, 4:425–429, 1971.

[100] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems part i: Transportation between customers and depot. *Journal für Betriebswirtschaft*, 51:21–51, 2008.

[101] S. N. Parragh, K. F. Doerner, and R. F. Hartl. A survey on pickup and delivery problems part ii: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 51:81–117, 2008.

[102] W. Powell and Y. Sheffi. The load-planning problem of motor carriers: problem description and a proposed solution approach. *Transportation Research A: Policy and Practice*, 17:471–480, 1983.

[103] W. B. Powell and Y. Sheffi. Design and implementation of an interactive optimization system for network design in the motor carrier industry. *Operations Research*, 37:12 –29, 1989.

[104] S. Raghavan. A heuristic for the iof routing problem. Working Paper, USWest AdvancedTechnologies Boulder, CO, 1995.

[105] J. T. Richardson, M. R. Palmer, G. Liepins, and M. Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. D. Shaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191 – 197. Morgan Kaufmann, 1989.

[106] S. Ropke and J.-F. Cordeau. Branch-and-cut-and-price for the pickup and delivery problem with time windows. *Transportation Science*, 2009. Forthcoming.

[107] S. Ropke, J. F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49 (4):258–272, July 2007.

[108] K. S. Ruland and E. Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics With Applications*, 33 (12):1–13, June 1997.

[109] R. Russel and D. Gribbin. A multiphase approach to the period routing problem. *Networks*, 21:747–765, 1991.

[110] R. Russel and W. Igo. An assignment routing problem. *Networks*, 9:1–17, 1979.

[111] D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pages 269–280. North-Holland, Amsterdam, 1981.

[112] M. Savelsbergh and M. Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998.

[113] M. W. P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, Feb. 1995.

[114] A. E. Smith and D. M. Tate. Genetic optimization using a penalty-function. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 230 – 236. Morgan Kaufmann, 1993.

[115] M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35:254265, 1987.

[116] H. Tang, E. Miller-Hooks, and R. Tomastik. Scheduling technicians for planned maintenance of geographically distributed equipment. *Transportation Research Part E: Logistics and Transportation Review*, 43(5):591–609, 2007.

[117] S. van Hoesel, A. Koster, R. van de Leensel, and M. Savelsbergh. Polyhedral results for the edge capacity polytope. *Mathematical Programming*, 92:335–358, 2002.

[118] S. van Hoesel, A. Koster, R. van de Leensel, and M. Savelsbergh. Bidirected and unidirected capacity installation in telecommunication networks. *Discrete Applied Mathematics*, 133:103–121, 2004.

[119] T. van Roy. A cross decomposition algorithm for capacitated facility location. *Operations Research*, 34:145–163, 1986.

[120] N. Wieberneit. Service network design for freight transportation: a review. *OR Spectrum*, 30:77–112, 2008.

[121] R. T. Wong. Vehicle routing for small package delivery and pickup services. In B. Golden, S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43, pages 475–485. Springer, 2008.