UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Tecnologie dell'Informazione

Corso di dottorato di ricerca in **informatica**
Ciclo **XVII**

Tesi di dottorato di ricerca

Branch-and-price algorithms for Vehicle Routing Problems

dott. Matteo Salani

Anno accademico 2004/2005

Contents

Abstract

In this thesis I consider some *routing problems* and exact algorithms to solve them based on the *branch-and-price* framework. I consider three variants of the *Vehicle Routing Problem* **(VRP)**: the *Capacitated Vehicle Routing Problem* **(CVRP)**: is the basic and most studied version of VRP; the *Vehicle Routing Problem with Distribution and Collection* (**VRPDC**): is the variation of the CVRP arising when the distribution of goods from a depot to a set of customers and the collection of waste from the customers to the depot must be performed by the same vehicles of limited capacity and where the customers can be visited in any order; the *Capacitated Vehicle Routing Problem with Time Windows* (**CVRPTW**): is the variation of the capacitated VRP arising when customers must be served within a specified period during the day.

These Vehicle Rouring Problems are strongly NP-hard; thus I search for exact solutions in reasonable computing time. Since the computational effort required by branch-and-price based algorithms is affected mainly by the efficient solution of the so called *pricing subproblem*, I devote the main part of the thesis to enlarge the knowledge on pricing algorithms .

In particular I present some new ideas to improve the known dynamic programming algorithms. I experimentally compare different strategies to solve the subproblem and their effect on the solution of the VRPs.

Chapter 1

Introduction

> *Whenever we use a telephone, shop at out neighborhood foodstore or mall, read our mail or fly for business or for pleasure, we are the beneficiaries of some system that has routed messages, goods or people from one place to another. [63]*

In this chapter I provide the motivation for the study of vehicle routing problems as well as the purpose of this thesis.

1.1 Motivation

The distribution of goods and data, the collection of waste, the transportation of people between places and, in general, all transportation systems require the organization and the planning of sequences of operations performed by vehicles, trucks, telecommunication networks and transportation media.

Since 1970 transport activity has more than doubled in the European Union: +185% for the transportation of goods and +145 % for the transportation of people. Road transport is today dominant over other modes of transport, with a market share of 45 % for the transportation of goods and of 87 % for passenger transport. In passenger traffic, it is air transport that has made the most progress since the sector was opened up to competition in the Nineties. This trend has been strengthened recently with the development of lowcost airlines.

The European Union, in the Energy & Transportation report 2000-2004, estimated the transportation costs for that period around EUR 210 billions. Transportation sector contributes for up to 10% to the gross domestic product (GDP) and employs more than 10 millions workers. The prospect of the European Union is that the demand of goods transportation will increase by 70% between now and 2020 in the EU member states and by 95% in the ten new member states and that this will raise the transportation costs by EUR 80 billions per year. Nevertheless transportation systems have also environmental costs, social costs and hidden costs (primarily on life quality). For instance the transportation of goods is responsible for the 28% of gas emissions in 1998 and this share is likely to increase to 50% between now and 2010.

For this reason the sequences of operations performed by transportation systems should be planned in some way to reduce such costs. The sequences, usually called *routes*, can be planned in several ways.

- No planning: requests are satisfied by a FIFO policy;

- Planned by transportation experts: decisions are based on expert's knowledge;

- Planned by computer programs: decisions are taken by *optimization algorithms*;

- Planned by transportation experts with the aid of computer programs: decisions are taken by experts with a quantitative information given by optimization algorithms.

Each approach has its own benefits and its own costs. The absence of planning is the quickest way to compute routes and it is costless; by contrast the quality of the computed routes is poor and the transportation costs are consequently high. The planning based on human knowledge is generally costly, it takes a large amount of time to be carried out, it can be performed only on a small amount of transportation requests and generally the quality of the solution is poor but it can be very adaptive to data changes and it is very flexible to consider several objectives at the same time. The design of routes made by computer algorithms is typically fast and cheap. It provides low cost solutions and it is able to deal with large problems but it is not able to detect errors in the input data and typically it is not devised to take into account several objectives at the same time. The use of a computer program by human experts is useful when several solutions should be evaluated. This type of use of computer programs is commonly known as *Decision Support System* (DSS). It can be implemented when very fast computer programs are available. Using a DSS the quantitative information given by computer programs can be used by human experts who are able to evaluate the computed routes using their own knowledge on the problem.

The last two approach are the challenging ones for researcher who are asked to develop faster and accurate algorithms used in DSSs. At the same time researcher are asked to develop exact algorithms for routing problems which are able to provide a certificate of optimality of the computed solution.

To develop effective algorithms for transportation planning, it is necessary to describe the problem in a formal way: a mathematical model. The model must capture the significant aspects of the transportation problem, representing them in terms of decision variables, constraints and cost functions. The first attempt to define mathematical models of transportation planning problems dates back to the seminal paper of Dantzig and Ramser [31] published more than 40 years ago. It considers the routing problem associated with a gasoline delivery and provides the first formulation of the general *Vehicle Routing Problem* (VRP). In the 60's Clarke and Wright [20] presented the first greedy heuristic approach to the problem. Their algorithm starts from an infeasible solution made of 1-customer trip and iteratively combines routes maximizing the savings.

Since then many progresses have been made both on the side of enriching the models and on the side of improving the algorithms on the vehicle routing problem.

The solution of a vehicle routing problem concerns the service of a set of *customers*, by a set of *vehicles* of a given *capacity* which are located in one or more *depots*. The vehicles are operated by *crews* and they travel along a given *road network*. The solution of a VRP is to determine a set of feasible *routes* whose global *transportation cost* is minimized and such that all customers requirements are satisfied; moreover each route is performed by a single vehicle that starts and ends its travel at its own depot.

Typically the road network is described by a *graph*, whose *arcs* (or *edges*) represent the streets and whose *nodes* represent the depot, the customers and the junctions. Generally with each arc is associated a *cost*, that typically represents the length, and a *travel time*. Moreover other problem-related information can be associated with the arc: allowed traversal periods, allowed vehicle types, time-dependent travel times and so on.

The *fleet* of vehicles that serve the customers is characterized by the *capacity* of each vehicle, that limits the load that the vehicle can carry, the number and characteristics of *commodities*, the *home depot* of each vehicle, the *cost* associated to the use of the vehicle and other information related to the subset of arcs that a vehicle can traverse, the maximum tour length for each vehicle, the loading operations that can be performed.

The information associated with the customers are generally related to the *demand* of goods or waste, possibly of different kind, that must be delivered or collected by the vehicle, the periods of the day (*time windows*) during which the customer can be served, the amount of time required to deliver or collect the goods (*service time*), possibly dependent on the vehicle type, and the set of vehicles allowed for the service operation. Finally the drivers that operate the vehicles have to follow union contracts and rules (breaks, working periods, maximum duration of driving time, driving license restrictions).

Some other constraints can be imposed on the routes. *Precedence constraints* can be imposed on the visiting order of the customers associated with a route (this situation arises in *pickup and delivery problems* and in *backhauling problems*). In order to compute the cost of a route and to check the operational constraints imposed on it the *travel costs* and the *travel times* between customers and depots have to be known. It is not uncommon that the network graph is transformed into a *complete graph* where the cost c_{ij} of arc (i, j) is equal to the *shortest path* connecting i to j and the travel time t_{ij} is equal to the sum of the travel time of the arcs belonging to the shortest path.

Vehicle routing problems can have several and contrasting objectives. Typically the global *transportation cost* is to be minimized. This cost is computed considering the travel costs (or travel times) and the fixed costs related to the use of the vehicles (and perhaps the associated drivers). The number of vehicles needed to serve all customers is another objective that often has to be minimized. This happens when the travel costs are negligible compared to the fixed cost associated with the use of the vehicles. Other objectives that can be required are the minimization of the penalties associated with delays, the maximization of the overall level of service, the balancing of the length of the routes and the balancing of the load of the vehicles.

I refer the reader to the book of Toth and Vigo [75] which surveys classical heuristic approaches (Laporte and Semet [27]), metaheuristics (Gendreau, Laporte and Potvin [4]) and exact approaches based on branch-and-bound (Toth and Vigo [73]), branch-and-cut (Naddef and Rinaldi [9]) and Branch-and-Price (Bramel and Simchi-Levi [36]). Other surveys of VRPs can be found in Fisher [55], Laporte [26] and in Laporte et al. [22]. Several exact algorithms have been proposed in the last decade. Branch-and-Bound based algorithms for direct graphs (see Fischetti et al. [53]) are able to solve up to 300 customers randomly generated instances and up to 70 customers real world instances. Branch-and-Bound algorithms for undirected graphs (see Fisher [54]) solve instances with up to 100 customers. Recent serial and parallel Branch-and-Cut codes (see Ralphs et al. [88], Lysgaard et al. [42] and Blasum et al. [89]) are able to solve difficult 76 and 100 customers instances. The Branch-and-Price implementation of and Hadjicostantinou et al. [14] provide effective lower bounds (at least 96%) for instances with up to 150 customers. Another effective Column Generation algorithm is proposed by Chabrier [1] who obtained 17 new optimal solutions for CVRPTW instances from the Solomon's set [61]. More recent hybridizations of Cut and Column generation algorithms can be found in Fukasawa et al. [79], the authors present a robust framework for the solution of routing problems that solves instances with up to 100 customers.

1.2 Purpose

The purpose of this thesis is to study and implement an algorithmic approach based on the Branch-and-Price framework for the solution of various vehicle routing problems. Such approach should be general enough to be applicable to different variations of the basic VRP and for this purpose I considered the three problems described in the next section: the Capacitated VRP, the VRP with Delivery and Collection and the VRP with Time Windows. Vehicle routing problems have been studied using different strategies and techniques, and it is often difficult to make a comparison between different results obtained on different machines or with slightly different formulations. An attempt to compare different approaches has been made by Toth and Vigo in [75]. The Branch-and-Price algorithms for vehicle routing problems presented in the literature are based on a Set Partitioning or Set Covering reformulation of the VRP, solved via column generation, where the pricing problem is the *Resource Constrained Elementary Shortest Path Problem* (RCESPP). Since the solution of the pricing problem is a central issue to implement effective Branch-and-Price codes, I devote the main part of this thesis to enlarge the knowledge on this problem. The most successful approaches for the RCESPP proposed in the literature are based on dynamic programming (see Irnich et al. [83] for a recent review). With this thesis I propose some innovative algorithmic ideas to improve the dynamic programming algorithms for the solution of the pricing problem: further elaborating on the approach of Feillet et al. [8] I propose exact algorithms for the RCESPP based on bi-directional and bounded dynamic programming. I also propose a new dynamic programming algorithm for the solution of the RCESPP, called *Decremental State Space Relaxation*, which is based on the *State Space Relaxation* idea proposed by Christofides et al. [67].

Finally a goal of this thesis is to investigate advantages and drawbacks of two possible approaches: to solve the pricing problem at optimality or to solve a relaxation, namely the RCSPP, where cycles are allowed. This second technique has been frequently used in the literature (see Cordeau et al. [44]). The trade-off between relaxed pricing, which is faster, and exact pricing, which yields a stronger dual bound, is computationally examined.

1.3 Outline

The chapter 2 is devoted to the definition and formulation of vehicle routing problems with particular attention to the CVRP, the VRPDC and the CVRPTW. Chapter 3 provides the shortcomings for column generation methods and the VRP reformulation. Chapter 4 describes the algorithms used to solve the pricing subproblem arising from the VRP reformulation. Chapter 5 is devoted to the implementation issues and finally chapters 6 to 8 report on computational experiments.

Chapter 2

Vehicle Routing Problems

In this chapter I provide the definitions of the vehicle routing problems that I consider. Next I provide different mathematical formulations for VRP. Moreover in this thesis I consider a less studied VRP problem arising in the reverse logistic environment: the vehicle routing problem with delivery and collection, where the forward flow of goods is combined, and should be optimized, with the backward flow of waste. This crucial problem in the reverse logistics management has several practical motivations because the recycling topic has received more and more attention in the last period.

2.1 The Capacitated Vehicle Routing Problem

The Capacitated Vehicle Routing Problem (**CVRP**) is the basic version of the VRPs, where all customers are delivery customers, the demands are known, all vehicles are identical and they belong to the same central depot. The only imposed constraints are related to the capacity of the vehicles.

The objective is to minimize the total travel cost. Using graph notation the CVRP may be described as follows: let $G = (V, A)$ be a complete graph, where $V = \{0, \dots, n\}$ is the node set in which node 0 represents the depot while nodes $1, \dots, n$ represent the customers.

A given nonnegative cost c_{ij} is associated to each arc $(i, j) \in A$ and it represents the travel cost to reach j starting from i.

If the graph is directed the problem is called asymmetric capacitated VRP (ACVRP) otherwise $c_{ij} = c_{ji}$ and the problem is called symmetric capacitated VRP (SCVRP). The graph G has to be (strongly) connected.

Given a set of nodes $S \subseteq V$, $\delta(S)$ is the set of edges of the graph with only one endpoint in S and $E(S)$ is the set of edges with both endpoints in S.

It is a common assumption that the cost matrix satisfies the *triangle inequality*, that is

$$c_{ik} + c_{kj} \geq c_{ij} \qquad \forall i, j, k \in V \tag{2.1}$$

It is also often assumed that each customer is associated with a point in the plane and the cost c_{ij} is equal to the Euclidean distance between the two points. In this case the distance matrix is symmetric and satisfies the triangle inequality. The resulting problem is called *Euclidean SCVRP*.

A known nonnegative delivery demand d_i is associated with each customer. Given the set $S \subseteq V$, $d(S)$ denotes the total demand of the set, that is $d(S) = \sum_{i \in S} d_i$.

A set of K identical vehicles, each with capacity Q, is available at the depot; Obviously $Q \geq d_i$ for each $i = 1, \ldots, n$. If it happens that $d_i > Q$ for some customer i the arising problem is called *VRP with Split Delivery* where multiple visits to each customer are allowed (see Dror et al. [52] and Archetti et al. [13]). It is assumed that the number of available vehicles K is equal than K_{min}, the minimum number of vehicles needed to serve all customers. The value of K_{min} can be computed solving a *Bin Packing Problem* (BPP) (see Martello and Toth [86]) associated with the CVRP. It requires the computation of the minimum number of bins needed to pack all the given items. The weight of each item is equal to the delivery demand d_i of each customer i, while the capacity of each bin is equal to Q.

The CVRP requires the computation of at most K tours with minimum cost such that:

1. each tour starts and ends at the depot
2. each customer is visited once
3. the sum of the demands of the customers visited in a tour does not exceed the vehicle capacity Q.

In the literature some variants of the CVRP have been considered. If the number of routes is to be minimized it is common to associate a big fixed cost with the use of the vehicles. This can be done if the number K of available vehicles at the depot is greater than the minimum number of vehicles required, K_{min} (see, e.g, Hadjicostantinou et al. [14] and Ralphs et al. [88]).

Another variant arises when the vehicles have different capacities $Q_k, k = 1, \ldots, K$ or when multiple depots are present and each vehicle is associated to its own depot (see Desrochers et al [50], Fisher [55], Ribeiro et al. [6] and Carpaneto et al. [19]).

When the cost c_{ij} represents the travel time from customer i to customer j the objective of the problem is to minimize the total duration of the routes. When a constraint on the maximum duration is imposed a *Distance Constrained* VRP (DVRP) arises. When both capacity and duration constraints are imposed the problem is called Distance Constrained CVRP (DCVRP). Moreover each vehicle may be associated with a different maximum travel time $T_k, k = 1, \ldots, K$ (see Laporte et al. [30], [28]).

The CVRP is known to be NP-hard in the strong sense because the well known *Traveling Salesman Problem* (TSP) arises as a special case, when $\sum_{i \in V} d_i \leq Q$ and $K = 1$ (see Lawler et al. [16]).

2.2 The VRP with Delivery and Collection

The *VRP with Delivery and Collection* (**VRPDC**)arises when with each customer i are associated a delivery demand d_i and a pickup demand p_i representing the transport requests of an homogeneous commodity. Sometimes it happens that only the difference between the delivery demand and the pickup demand is given (that can be negative). For each customer i the origin and the destination of the transportation request are common (that is they coincide with the depot). This is a crucial problem within the reverse logistic process where the forward flow of goods must be optimized with the backward flow of waste and recycled items.

The solution of a VRPDC consist of finding exactly K routes such that:

1. each tour starts and ends at the depot

2. each customer is visited once

3. the load of the vehicle does not exceed its capacity Q.

It is assumed that, at each customer, the delivery operation is performed before the pickup operation. This means, for example, that a vehicle with capacity equal to 10 can visit a node with pickup demand $p_i = 8$ and a delivery demand $d_i = 9$. The VRPDC has been considered by Dell'Amico et al [46] who proposed an exact algorithm based on Branch-and-Price. Dethloff [10] and Bianchessi and Righini [65] proposed respectively some heuristics and meta-heursistics to solve large sized instances. It is commonly assumed that the VRP with Delivery and Collection and the VRP with Simultaneous Pickups and Deliveries (VRPSPD) are the same problem.

When for each customer i two additional nodes are defined O_i and D_i representing respectively the origin of the goods to be delivered at node i and the destination of the goods picked up at node i the *VRP with Pickup and Delivery* (**VRPPD**) arises. (see Savelsbergh et al. [64] and Sol [60])

The solution of a VRPPD consist of finding exactly K routes such that:

1. each tour starts and ends at the depot
2. each customer is visited once in one tour
3. the load of the vehicle does not exceed its capacity Q.
4. for each customer i the origin node O_i is visited in the same route and before customer i
5. for each customer i the destination node D_i is visited in the same route and after customer i

Both VRPPD and VRPDC are NP-hard in the strong sense since they generalize the CVRP problem, arising when $O_i = D_i = 0$ and $p_i = 0$ for each $i \in V$. Moreover the *TSP with Pickup and Delivery* and the *TSP with Distribution and Collection* arise when $K = 1$.

A special version of VRPPD arises when each customer i has only a delivery demand or a pickup demand, and the set of customers can be divided into two parts: the *Linehaul customers* requiring the delivery of a certain amount of goods d_i and the *Backhaul customers* requiring the collection of an amount p_i; the set of linehaul customers has to be served before the set of backhaul customers. This variant is called *VRP with backhauls* (**VRPB**) (see Goetschalckx et al. [57] and Mingozzi et al. [2] and Toth et al. [72]).

2.3 The VRP with Time Windows

In real world applications customers are not available for delivery operations during all the day due, for example, to warehouse opening time periods, for traffic restrictions on the road network within cities or for any other time restriction.

The widely studied version of CVRP with time restrictions is the VRP *with Time Windows* (**CVRPTW**) where each customer i has an associated *time window* $[a_i, b_i]$. Customer i has to be visited within its time window. It is common that waiting until instant a_i at node i is allowed if the vehicle arrives early.

The travel time t_{ij} and the service time s_i at node i are given. The service time represents the time needed for loading or unloading operations at customers location.

A common assumption is that the costs and the travel-times coincide, and the starting time of the vehicles leaving the depot is assumed to be equal to 0.

Similarly to the CVRP the solution of the CVRPTW is a set of K routes of minimum total cost such that

1. each route starts and ends at the depot
2. each customer is visited once
3. the capacity of the vehicle is not exceeded along the route
4. for each customer i the service time starts within the associated time window and it takes s_i.

When the capacity constraint is not tight, that is $\sum_{i \in V} d_i \leq Q$ and $K = 1$, a *TSP with time windows* arises (see Dumas et al. [92]).

Several special cases and variants have been considered in the literature: the multiple TSP with time windows arises from an CVRPTW eliminating the capacity constraint (see Desaulniers et al. [24], [39]); heterogeneous fleet, multiple-depot and multiple time windows have been considered in Solomon and Desrosiers [62]; soft time windows have been taken into account as an extension in Desaulnier et al. [23].

Several other problems arise when pick-up and delivery operations are combined with time windows see, e.g., Thangiah et al. [87], Sigurd et al. [59], Nanry and Barnes [91] and Hong and Liang [7].

CVRPTW is NP-hard in the strong sense because it generalizes the CVRP when $a_i = 0$ and $b_i = \infty$ for each customer i.

For a recent survey on the CVRPTW see Cordeau et al. ([44]).

2.4 Mathematical models

In this section I recall the basic mathematical programming formulations for VRPs. For the purpose of this thesis I will use only the set partitioning model of VRP presented in section 2.4.3 but for the sake of completeness I also report the flow model and the commodity model. Since an exhaustive discussion of mathematical formulations can be found in [74] and [29] here I give only a brief description of the most commonly used models for CVRP.

2.4.1 Vehicle flow model

There are two main type of vehicle flow models: the first is a *two-index vehicle flow model* which uses $O(n^2)$ binary variables x. Each variable x_{ij} takes value 1 if arc (i, j) is used in the solution and 0 otherwise. Constraints are imposed on the incoming and outgoing degree of the flow variables for each customer i and *subtour elimination* constraints are imposed. There are different type of subtour elimination constraints (see, e.g., Toth and Vigo [74]). This flow formulation is used mainly for basic versions of the VRP since there is no way to handle practical issues such as different vehicles, precedence constraints, time windows, etc.

The second vehicle flow model is the *three-index vehicle flow model*; it uses $O(n^2K)$ binary variables x and $O(nK)$ binary variables y. Each variable x_{ijk} takes value 1 if the vehicle k traverse arc (i,j) in the solution, 0 otherwise. Each variable y_{ik} is equal to 1 if vehicle k serves customer i.

This model is useful to represent constraints on the type of vehicle serving each customer or vehicle-related constraints such as different capacities or tour lengths.

Here I report the three-index mathematical model:

$$\text{Min} \sum_{i \in V} \sum_{j \in V} c_{ij} \sum_{k=1}^{K} x_{ijk} \tag{2.2}$$

$$\text{s.t.} \sum_{k=1}^{K} y_{ik} = 1 \qquad \forall\, i \in V \setminus \{0\} \tag{2.3}$$

$$\sum_{k=1}^{K} y_{0k} = K \tag{2.4}$$

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} = y_{ik} \qquad \forall\, i \in V, k = 1, \dots, K \tag{2.5}$$

$$\sum_{i \in V} d_i y_{ik} \leq Q \qquad \forall k = 1, \dots, K \tag{2.6}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \qquad \forall\, S \subseteq V \setminus \{0\}, |S| \geq 2, k = 1, \dots, K \tag{2.7}$$

$$x_{ijk} \in \{0,1\} \qquad \forall\, i,j \in V, k = 1, \dots, K \tag{2.8}$$

$$y_{ik} \in \{0,1\} \qquad \forall\, i \in V, k = 1, \dots, K \tag{2.9}$$

where 2.7 is one of the known formulations for the subtour elimination constraints.

Successful applications of the vehicle flow model can be found in Branch-and-Bound and Branch-and-cut algorithms for the Capacitated Vehicle Routing Problem (see Toth and Vigo [73] and Naddef and Rinaldi [9])

There are many extensions for the three-index flow formulation to handle different issues. I refer the reader to Toth and Vigo [74], [75] for more details on those extensions.

Recent implementations of parallel Branch-and-Cut computer programs are able to solve up to 100 customers instances of VRP in reasonable computing time (see Ralphs et al. [88], Fukasawa et al. [79] and Lysgaard et al. [42]).

2.4.2 Commodity flow model

Commodity flow model is based on *flow variables* y_{ij} and y_{ji} that represent respectively the capacity used and the capacity left on the vehicle traveling from i to j. These formulations need an extended graph $G'(V', A')$ where a copy of the depot, say node $n+1$, has been added. Within the modified graph routes become paths form vertex 0 to vertex $n+1$. The binary variables x, similarly to the two-index vehicle flow formulation, represent the use of the arc.

$$\text{Min} \sum_{(i,j)\in A'} c_{ij}x_{ijk} \tag{2.10}$$

$$\text{s.t.} \sum_{j\in V'} (y_{ji} - y_{ij}) = 2d_i \qquad \forall\, i \in V' \setminus \{0, n+1\} \tag{2.11}$$

$$\sum_{j\in V'\setminus\{0,n+1\}} y_{0j} = \sum_{i\in V'\setminus\{0,n+1\}} d_i \tag{2.12}$$

$$\sum_{j\in V'\setminus\{0,n+1\}} y_{j0} = KQ - \sum_{i\in V'\setminus\{0,n+1\}} d_i \tag{2.13}$$

$$\sum_{j\in V'\setminus\{0,n+1\}} y_{n+1j} = KQ \tag{2.14}$$

$$y_{ji} + y_{ij} = Q \qquad \forall (i,j) \in A' \tag{2.15}$$

$$\sum_{i\in V'} (x_{ij} + x_{ji}) = 2d_i \qquad \forall\, i \in V' \setminus \{0, n+1\} \tag{2.16}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in A' \tag{2.17}$$

$$y_{ij} \geq 0 \qquad \forall\, (i,j) \in A' \tag{2.18}$$

An application of the commodity flow formulation to the VRPDC can be found in Baldacci et al. [78].

2.4.3 Set partitioning model

The *set partitioning* model is based on a large set made of all feasible routes, say P. The solution of the set partitioning problem asks to select from this set K routes of minimum cost such that each customer is served exactly by one route.

$$\text{Minimize} \sum_{r\in P} c_r z_r$$

$$\text{subject to} \sum_{r\in P} a_{ir} z_r = 1 \qquad \forall i \in V \tag{2.19}$$

$$\sum_{r\in P} z_r = K \tag{2.20}$$

$$z_r \in \{0,1\} \qquad \forall r \in P \tag{2.21}$$

where c_r is the cost of route r and a_{ir} is the number of times route r visits customer i. All problem related constraints are considered in the subproblem of constructing the set of feasible routes P. The high flexibility of the set partitioning formulation to handle several constraints (see Lübbecke [15] for a recent review; see Desrosiers et al. [37], Desrocher and Soumis [48] and Ribeiro et al [6] for some applications of Branch-and-Price to VRP) within the same framework is the reason for the extensive research performed in the last 20 years.

In the next chapter I will introduce the methods used to deal with set partitioning model.

Chapter 3

Column generation and Branch-and-Price

In this chapter I recall the basic concepts of column generation and Branch-and-Price to solve the set covering reformulation of VRPs.

3.1 Column Generation

Several linear problems exhibit some kind of structure in the constraint matrix in form of large submatrices of zeros. Within the constraint matrix it is possible to distinguish two type of constraints: the linking constraints and the "subsystem" constraints. The main idea of decomposition methods is to use the linking constraints at a coordinating level and the "subsystem" constraints at a subordinated level using the subproblem structure to devise ad-hoc algorithms for them.

The seminal work of Dantzig and Wolfe [32] originated an effective decomposition method for large linear programs.

Let us consider the following linear program called *master problem* (MP):

$$\begin{aligned} z^* = \min_{p \in P} & \sum c_p x_P \\ \text{s.t.} & \sum_{p \in P} \mathbf{a_p} x_p \geq \mathbf{b} \\ & x_p \geq 0 & \forall p \in P \end{aligned}$$

At each iteration of the simplex method we look for a non-basic variable to price out and enter the basis. This means that given the non negative vector of dual variables λ we want to find:

$$\min_{p \in P} \{\bar{c}_p = c_p - \lambda^{\mathbf{T}} \mathbf{a_p}\}$$

The complexity of this step depends on $|P|$ and is time consuming when $|P|$ is huge. The idea is to work with a small subset of columns $P' \subseteq P$: the linear program derived in such way from MP is called *Restricted Master Problem* (RMP). If the RMP is feasible, let $\mathbf{x}^*$ and λ^* be respectively the primal and dual optimal solutions of the RMP and assume the cost c_p of a generic column p can be computed as a function $c : P \rightarrow \Re$. Then the problem

$$c^* = \min_{p \in P} \{c(p) - \lambda^{T*} a_p\}$$

is an oracle for pricing. If the solution is non-negative then no reduced cost coefficients $\overline{c}_p$ has a negative value and the restricted master problem cannot be improved and x^* is the optimal solution also for the original master problem. Otherwise the column which has the reduced cost equal to c^* is a candidate to enter the basis and it is added to the RMP. The method is repeated until no negative reduced cost columns are found. The method described is known as *Column Generation.* When the set of negative reduced cost solutions is finite then the column generation algorithm converges and is exact. The core part of column generation is, of course, the pricing step. It inherits the difficulty of the non-basic column search problem but it gains the advantage to be structured in a different way with respect to the original problem. Let $\overline{z}$ be the optimal objective value of the RMP over a set of feasible columns P', $\overline{z} = \lambda^{T*} b$. When $W \geq \sum_{p \in P} x_p$ is a valid upper bound of the optimal solution of the master problem also a lower bound can be computed:

$$\overline{z} + W \cdot c^* \leq z^* \leq \overline{z}$$

The presented bound is available when exact pricing is performed but it may be computed also when a lower bound $\underline{c}^*$ on c^* is known. This bound is always weaker then the one obtained by exact pricing and typically it is computed when the computing time for exact pricing is unmanageable.

3.2 Branch-and-Price

When we are dealing with an integer linear program of the form:

$$\begin{aligned} z^* = \min_{p \in P} & \sum c_p x_P \\ \text{s.t.} & \sum_{p \in P} \mathbf{a_p} x_p \geq \mathbf{b} \\ & x_p \in Z^+ \qquad\qquad \forall p \in P \end{aligned}$$

we would like to apply the column generation method described above to find optimal integer solutions; to do so we need to embed the column generation process into a branch-and-bound algorithm.

The first known attempt dates back to the same years of Dantzig and Wolfe when, independently, Gilmore and Gomory ([76]) developed a column generation approach to the cutting stock problem. Several applications of column generation techniques appeared in the last two decades, Desrosiers et al. [37], Desrocher and Soumis [48], Ribeiro et al [6], Vance et al. [77] and Gamache et al. [56]. The above list of articles on column generation methods isn't exaustive (for a recent and complete survey on column generation methods see Lübbecke and Desrosiers [15]) but it is sufficient to conclude that most successful applications of column generation happen in IP problems which can be modeled as set partitioning (or set covering) ones. In most of the above examples columns have a defined structure and ad-hoc algorithms can be modeled to price out new columns. Because the pricing problem often encodes structures like paths, sets or permutations encoding the knowledge on how the columns are to be constructed.

In this context the *Branch-and-Price* algorithm (B&P), which is a generalization of branch-and-bound with LP relaxations, allows the generation of new columns throughout the branch-and-bound process. Moreover the integer linear program reformulation still allow to devise good branching rules compatible with pricing algorithms.

Figure 3.2 represents the scheme of a basic Branch-and-Price algorithm. The search tree is properly initialized with the root node. At each node of the search tree the RLMP is initialized with a subset of feasible columns for the active node. Then the linear relaxation of the restricted master problem is solved by column generation as described above. If the solution is integer then it is a feasible solution for the original master problem and it is compared with the current incumbent solution, in the same manner of standard branch-and-bound. If the LP solution does not satisfy the integrality conditions then branching occurs to cut off the current fractional point. At the end of the search process the best integer solution is the optimal solution for the original integer linear problem.

The lower bound described in the previous section is still valid for column generation applied to integer linear programs. An application of the weaker bound mentioned above can be found in Desrosiers et al [37] and Desrochers et al. [49] where the authors used the solution of the relaxation of the pricing problem (the resource constrained elementary shortest path, RCESPP) to compute a valid lower bound.

A valid branching scheme should cut off the current fractional solution, should produce a balanced search tree and should keep the structure of the problem unchanged.

It should be pointed out that conventional integer programming branching on variables of the RMP is not very effective on B&P algorithms because it destroys

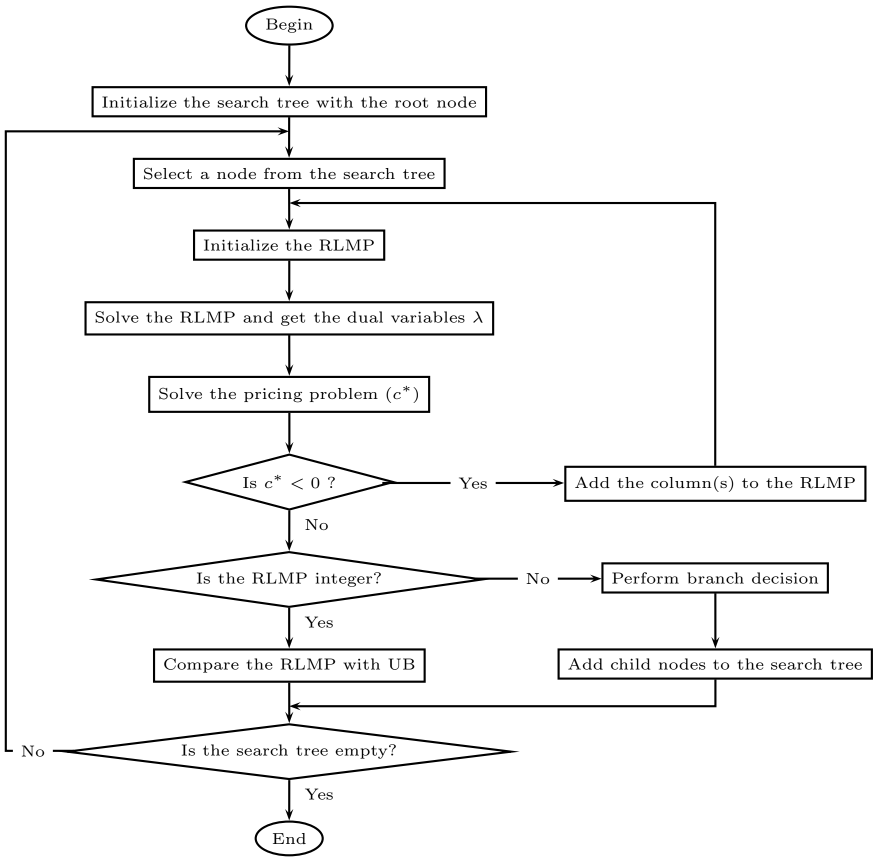

Figure 3.1: The Branch-and-Price algorithm

the structure of the pricing problem. While fixing a variable to 1 does not create problems to be taken into account at the pricing level, fixing a variable to 0 means that this column is excluded from the RMP and it should not be generated anymore by the oracle for the pricing. Unfortunately it is likely to happen because the excluded column will have profitable dual prizes.

Several branching rules have been proposed in the literature to overcome this last issue. Here I report some of the main ideas to perform branching in B&P algorithms.

- One of the most common branching scheme is the one proposed by Ryan and Foster [11] for problems based on set partitioning formulation considering the following proposition: If Y is a $0-1$ matrix and a basic solution to $Yx = 1$ is fractional then there exist two rows r and s such that:

$$0 < \sum_{k: y_{rk}=1,\ y_{sk}=1} x_k < 1$$

  The pair r, s gives the following branching constraints:

$$\sum_{k: y_{rk}=1,\ y_{sk}=1} x_k = 0 \quad \text{and} \quad \sum_{k: y_{rk}=1,\ y_{sk}=1} x_k = 0$$

  the rows r, s have to be covered by different columns on the left branch and by the same column on the right branch. The above conditions can be added to the master problem as constraints or taken into account by removing unfeasible columns from the master problem. Not adding explicitly the branching constraints to the RMP has the advantage of not introducing new dual variables that have to be considered in the pricing problem. On the other hand the insertion and cancellation of columns must be handled at each node of the search tree. When dealing with routing problems it is more easy, at the pricing level, to impose that two customers have to be covered by the same vehicle than imposing that two customers have to be covered by different vehicles.

- Another effective method is branching on original variables. We know that original variables are integer and we need to separate their fractional values when arising in the column generation process. For example Desaulniers et al. [23] and Desrochers et al. [49] used this method in the vehicle routing problem with time windows taking decisions on flow variables and on the starting time of the service at customers. In section 5.1.9 I devise the same methods to get integer values from the RMP.

3.3 VRP reformulation

As explained in chapter 2 vehicle routing problems require to compute a set of tours for a fleet of vehicles that must provide a certain kind of service to a given set of customers. Each vehicle starts from a given depot and goes back to it after visiting a subset of customers. The objective is to minimize the total distance traveled.

The structure of vehicle routing problems suggests to reformulate them as set covering problems and to apply column generation, because a solution is made by a set of subtours, one for each vehicle of the fleet, which can be computed independently provided that they cover the set of customers to be visited. A comprehensive treatment of column generation approaches to vehicle routing problems can be found in Desrosiers et al. [38], [23], Fisher [55] and Bramel et al. [36].

Hereafter I recall the theoretical aspects of decomposition principles and provide the Dantzig-Wolfe decomposition process to obtain a set-partitioning model, like the one presented in section 2.4.3, starting from a vehicle flow model presented in 2.4.1.

The theory is based on the Minkowski's theorem (see Nemhauser and Wolsey [33]). Let $\mathcal{X} = \{\ X \in \mathbb{R}^n_+ |\ AX \leq b\}$ be a nonempty polyhedron defined by a finite set of constraints and lying within a nonnegative orthant of real numbers. A point $x_p \in \mathcal{X}$ is defined to be an extreme point of $\mathcal{X}$ if there do not exist X^1, $X^2 \in \mathcal{X}$, $X^1 \neq X^2$, such that $x_p = 1/2X^1 + 1/2X^2$. If $\mathcal{X}(0) = \{\ X \in \mathbb{R}^n_+ |\ AX \leq 0\} \neq \{0\}$, then $X \in \mathcal{X}(0) =$ is called a ray of $\mathcal{X}$. A point $x_p \in \mathcal{X}$ is defined to be an extreme ray of $\mathcal{X}$ if there do not exist rays X^1, $X^2 \in \mathcal{X}(0)$, $X^1 \neq \lambda X^2$, for any $\lambda > 0$, such that $x_p = 1/2X^1 + 1/2X^2$.

Then a polyhedron $\mathcal{X}$ has a finite number of extreme points and extreme rays and a point $X \in \mathcal{X}$ can be written as a convex combination of extreme points plus a nonnegative combination of extreme rays.

The decomposition scheme is obtained dropping the integrality constraints (2.8) and (2.9). The master problem is given by (2.2)-(2.4). The subproblem is defined by the flow conservation constraints (2.5) and capacity constraints (2.6) from constraints (2.4)and (2.9) we can impose $y_{0k} = 1$ in each subproblem and therefore the subtour elimination constraints (2.7) are handled implicitly.

Let F^k the polyhedron defined by constraints (2.5)-(2.7) associated with vehicle $k \in K$. When we impose the integrality constraints (2.8) and (2.9) we define the convex hull of F^k

Let P be the set of feasible paths. Each path $p \in P$ corresponds to an elementary path which can be described by binary values $x^k_{ijp}, k \in K, (i,j) \in A, p \in P$. Any solution X^k_{ij}, Y^k_i to the master problem can be expressed as a nonnegative convex combination of a finite number of elementary paths:

$$X^k_{ij} = \sum_{p \in P} x^k_{ijp} \Theta^k_p \qquad \forall k \in K, \forall (i,j) \in A \tag{3.1}$$

$$X^k_{i} = \sum_{p \in P} \sum_{j \in V} x^k_{ijp} \Theta^k_p \qquad \forall k \in K, \forall i \in V \tag{3.2}$$

$$\sum_{p \in P} \Theta^k_p = 1 \tag{3.3}$$

$$\Theta^k_p \geq 0 \qquad \forall p \in P \tag{3.4}$$

Hence $x^k_{ijp}, k \in K, (i,j) \in A, p \in P$ represents the set of extreme rays of the subproblem set of constraints F^k.

Making the substitution into (2.2)-(2.4) the master problem takes the following form:

$$\text{minimize} \sum_{p \in P} (\sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x^k_{ijk}) \Theta^k_p \tag{3.5}$$

$$\text{subject to} \sum_{p \in P} (\sum_{j \in V} x^k_{ijp}) \Theta^k_p = 1 \qquad \forall k \in K, \forall i \in V \tag{3.6}$$

$$\sum_{p \in P} (\sum_{j \in V} x^k_{0jp}) \Theta^k_p = 1 \qquad \forall k \in K, \forall i \in V \tag{3.7}$$

$$\Theta^k_p \geq 0 \qquad \forall k \in K, \forall p \in P \tag{3.8}$$

Next define the parameters c_p, a_{ip}, b_p as follows:

$$c^k_p = \sum_{(i,j) \in A} c_{ij} x^k_{ijp} \tag{3.9}$$

$$a^k_{ip} = \sum_{j \in V} x^k_{ijp} \tag{3.10}$$

$$b^k_p = \sum_{j \in V} x^k_{0jp} \tag{3.11}$$

Since $\sum_{j \in V} x^k_{0jp} = y^k_{0p} = 1$ then $b^k_p = 1$ for all $k \in K$

Making the substitution in (3.5)-(3.7) the resulting master problem is then given by:

$$\text{minimize} \sum_{p \in P} \sum_{k \in K} c^k_p \Theta^k_p \tag{3.12}$$

$$\text{subject to} \sum_{p \in P} a^k_{ip} \Theta^k_p = 1 \qquad \forall k \in K, \forall i \in V \tag{3.13}$$

$$\sum_{p \in P} \Theta^k_p = 1 \qquad \forall k \in K \tag{3.14}$$

$$\Theta^k_p \geq 0 \qquad \forall k \in K, \forall p \in P \tag{3.15}$$

If the fleet is composed by identical vehicles the restricted master problem can be surrogated over k. Let $z_p = \sum_{k \in K} \Theta_p^k$, $c_p = \sum_{k \in K} c_p^k$ and $a_{ip} = \sum_{k \in K} a^k ip$ leading to the following formulation:

$$\text{minimize} \sum_{p \in P} c_p z_p$$

$$\text{subject to} \sum_{p \in P} a_{ip} z_p = 1 \qquad \forall i \in V \tag{3.16}$$

$$\sum_{p \in P} z_p = K \tag{3.17}$$

$$z_p \geq 0 \qquad \forall p \in \mathcal{P} \tag{3.18}$$

The above formulation is the linear relaxation of a set partitioning type problem with additional constraint on the total number of vehicles and a set of convex combination constraints.

Let z_p be an optimal solution of the restricted master problem over the subset P' of feasible columns and $\lambda_i, i \in V$ and λ_0 be the dual variables associated with the covering constraints (3.16) and with the constraint (3.17), respectively. A new column with minimum marginal cost is generated solving the following problem:

$$\text{minimize } r_p = c_p - \sum_{i \in V} \lambda_i a_{ip} - \lambda_0 \tag{3.19}$$

$$\tag{3.20}$$

Where the unknown vector a_p is to be determined.

The kind of pricing problem arising in this context is therefore a shortest path problem with some special characteristics: first, it is formulated on a graph with costs on the arcs and prizes on the vertices. This is equivalent to formulate it on a graph with no prizes but with negative cost arcs and possibly negative cost cycles. Therefore the requisite that the path must be elementary does not come for free from cost minimization but it must be explicitly enforced. Second, the pricing problem may be subject to a number of additional restrictions, as mentioned above. These constraints are usually represented as *resource constraints*, since distances, costs, time, capacities can all be interpreted as resources, that are consumed every time a vehicle travels along an arc or visits a customer. Therefore the pricing problem turns out to be a resource constrained elementary shortest path problem (RCESPP).

In the following chapter I formally define the RCESPP and provide algorithms based on dynamic programming for its solution.

Chapter 4

The pricing problem: the RCESPP

In this chapter I consider dynamic programming algorithms for the pricing problem (the resource constrained *elementary* shortest path problem, RCESPP), following the same approach of Feillet et al. [8] and I suggest and evaluate some ideas to improve their performance. Moreover I propose a new algorithm based on state space relaxation obtained from the improved one.

The elementary shortest path problem (without resource constraints) has been studied in many textbooks: see for instance the classical reference by Ahuja et al. [81]. The shortest path problem with resource constraints has been addressed with methods based on the Lagrangean relaxation of the resource constraints, like those of Handler and Zang [34] and Beasley and Christofides [43]; however these methods are effective when the Lagrangean subproblem is a polynomially solvable shortest path problem, that is when arc costs are non-negative. If the underlying graph may have negative cost cycles, the resource constrained elementary shortest path problem (RCESPP) is strongly NP-hard [51].

It is possible to address the pricing problem by optimizing its relaxation, obtained by dropping the constraint that the path must be elementary. Solving a resource constrained shortest path problem (RCSPP) requires less computing time but yields less tight lower bounds, since columns may include cycles. The two different approaches have been followed for instance by Feillet et al. [8] and Desrochers et al. [49] to solve the vehicle routing problem with time windows (CVRPTW) through column generation. The trade-off between the advantages and the drawbacks of these two design choices depends on the kind of side-constraints of the vehicle routing problem and on their tightness. The main scope of this thesis is to investigate this topic.

For a recent survey on models and algorithms for the RC(E)SPP I refer the reader to Irnich and Desaulniers [84].

From section 4.1 to section 4.5 I provide the definitions of RCESPP and the dynamic programming algorithm of Desrochers [25] and Feillet et al. [8] for its solution. In particular I consider three variants arising from the set covering reformulation of the capacitated vehicle routing problem (CVRP) the vehicle routing problem with delivery and collection (VRPDC) and the capacitated vehicle routing problem with time windows (CVRPTW) presented in chapter 2. In section 4.6 I illustrate the main ideas to improve the elementary dynamic programming algorithm. In section 4.7 provide the definitions of the state space relaxation of the RCESPP. In section 4.7.2 I illustrate the ideas to compute an elementary solution using the state space relaxation. In section 4.8 I propose a new algorithm for the RCESPP based on state space relaxation.

4.1 Problem definition

The RCESPP is defined as follows: a graph $\mathcal{G}(\mathcal{V}, \mathcal{A})$ is given, where the vertex set $\mathcal{V}$ is made by a set of vertices $\mathcal{N}$ representing N customers and two vertices s and t representing the depot. A non-negative prize λ_i is associated with each vertex $i \in \mathcal{N}$, a non-negative cost λ_0 is associated with the depot and a non-negative cost c_{ij} is associated with each arc $(i, j) \in \mathcal{A}$. Costs represent shortest paths and therefore they satisfy the triangle inequality. A vehicle must go from s to t, visiting a subset of the other vertices; no cycles are allowed. The objective is to minimize the cost, given by the sum of the costs of the arcs traversed minus the sum of the prizes collected at the vertices visited.

These definitions of the problem are common to all RCESPP versions arising from the different routing problems I consider. Additional constraints must be taken into account, depending on the kind of vehicle routing problem at hand. All these additional constraints are modeled as resource constraints and they will be specified in the remainder.

4.2 Dynamic programming

The basic dynamic programming approach to the RCESPP is based on the algorithm devised by Desrochers [25] for the RCSPP. It is an extension of the Bellman-Ford algorithm with the addition of resource constraints. The algorithm assigns *states* to each vertex: each state of vertex i represents a path from s to i. Each state has an associated resource consumption vector R whose component R^r represents the quantity of resource r used along the corresponding path. Each state has an associated cost C and the optimal solution is the minimum cost path reaching t. Different states associated with the same vertex i correspond to different feasible paths reaching i. Hence states are represented by a label of the form (R, C, i). The dynamic programming algorithm iteratively selects a vertex and extends its states to all possible successors. When a state (R, C, i) is extended to generate other feasible states (R', C', j), the cost and the resource consumption vector of the new state must be computed and those states for which one or more components of R' exceed the available capacity are fathomed. The cost is initialized at 0 at vertex s and it is updated according to the formula

$$C' := C + c_{ij} - \lambda_i/2 - \lambda_j/2$$

where $\lambda_i = -\lambda_0$ if $i = s$ and $\lambda_j = -\lambda_0$ if $j = t$ while vector R is initialized and updated according to the specific problem at hand. In addition dominance rules are applied in order to delete dominated states.

4.3 Resource constraints

Hereafter I consider three different specializations of the resource constraints arising from the CVRP, the VRPDC and the CVRPTW. I chose these three problems to validate the proposed approach, because they offer a significant mix of different characteristics of resource types and constraints. In the first case there is only one resource, whose consumption depends on the vertices visited. In the second case there are two resources associated with the vertices visited and they are interacting: the consumption of one of them also depends on the consumption level of the other. In the third case there are two resources, one associated with the vertices visited and the other associated with the arcs traversed. Resources are subject to a constraint on their overall consumption along the s-t path, with the exception of the case with time window, where a resource (time) is subject to different local constraints at each vertex.

Capacity. In the CVRP a non-negative integer pick-up demand p_i is associated with each vertex $i \in \mathcal{N}$ and a non-negative vehicle capacity Q is given. The sum of the demands of the nodes visited by the same vehicle cannot exceed Q. This constraint is modelled by one resource, representing the amount of capacity still available along a path. Let q be the amount of resource consumed. When a vehicle leaves vertex s it is empty, that is $q = 0$. Every time a node i is visited the corresponding amount of load p_i is stored on board; therefore q is increased by p_i. Each state is represented by a label (q, C, i), where q is the amount of load picked-up from s to i (included). Each time a state is extended along arc (i, j)

from a label (q, C, i) to a label (q', C', j), the resource consumption update rule is

$$q' := q + p_j$$

A state (q, C, i) is feasible only if $q \leq Q$.

Distribution and collection. In the VRPDC each vertex i has two non-negative integer quantities p_i and d_i associated with it, representing respectively the amount of load to be collected and to be delivered at that vertex. Each vehicle has a finite capacity Q, it leaves the depot carrying the total amount of load it must deliver and returns to the depot carrying the total amount of load it has collected. The capacity cannot be exceeded anywhere along the path. In the corresponding RCSPP the capacity constraint is taken into account by two additional resources, whose consumption is indicated by π and δ. The first resource at vertex j is the amount of load that the vehicle can pick-up after visiting j. Its consumption π increases after every pick-up operation, because when the vehicle visits vertex j, it consumes p_j units of this resource. The second resource at node j indicates the amount of load that the vehicle can deliver after visiting j. Initially Q units are available for this resource and the available resource decreases each time a delivery operation is performed but it may decrease also after pick-up operations since the maximum amount the vehicle can deliver after visiting j cannot be greater than the maximum amount it can pick-up after visiting j. Hence both π and δ are initialized at 0 and when a path is extended along arc (i, j) from a state (π, δ, C, i) to a state (π', δ', C', j), the update rule for the resource consumptions π and δ is:

$$\begin{aligned} \pi' &= \pi + p_j \\ \delta' &= \max\{\delta + d_j, \pi + p_j\} \end{aligned}$$

A state (π, δ, C, i) is feasible only if $\pi \leq Q$ and $\delta \leq Q$; for the formulae above the latter condition implies the former.

Capacity and time windows. In the CVRPTW a non-negative integer service time θ_i and a time window $[a_i, b_i]$ are associated with each vertex $i \in \mathcal{N}$ and each visited vertex must be reached inside its time window. If the vehicle arrives at i before a_i, it waits until time a_i. The traveling time from i to j is represented by the arc cost c_{ij} (this hypothesis has been made for simplification purposes but it does not affect the ideas and the algorithms outlined in the remainder). In this case time elapsed is a consumed resource, monotonically non-decreasing along the route. In the well-known Solomon's instances, which are commonly used as benchmarks for routing algorithms, a capacity constraint is also considered as in the CVRP. Hence in the corresponding RCSPP we need two resources, whose consumption is indicated by τ and q, that are respectively the time and the capacity consumed up to the beginning of service at each vertex. Both of them are initialized at 0 and each time a feasible path is extended along arc (i, j) from a state (τ, q, C, i) to a state (τ', q', C', j) the update rules for τ and q are:

$$\tau' := \max\{\tau + \theta_i + c_{ij}, a_j\}$$
$$q' := q + d_j$$

A state (τ, q, C, i) is feasible only if $\tau \leq b_i$ and $q \leq Q$.

4.4 Elementary path constraints

The dynamic programming algorithm described above solves the RCSPP with pseudo-polynomial worst-case time complexity. The same algorithm can be used to solve the RCESPP, where feasible paths are not allowed to contain cycles. To this purpose Beasley and Christofides [43] proposed to add to the state an additional binary resource for each node $i \in \mathcal{N}$. There is only one unit available for each dummy resource and it is consumed when the corresponding vertex is visited. Hence I consider N resources, whose consumption is indicated by a vector S initialized at 0. When a feasible path is extended along arc (i, j) from a state (S, R, C, i) to a state (S', R', C', j), the update rule for S is

$$S'_k := \begin{cases} S_k + 1 & k = j \\ S_k & k \neq j \end{cases}$$

A state (S, R, C, i) corresponds to an elementary path only if $S_k \leq 1 \ \forall k \in \mathcal{N}$. Note that S does not keep any information about the order in which the vertices are visited.

4.5 Dominance tests

The effectiveness of the dynamic programming algorithm outlined above heavily relies upon the possibility of fathoming feasible states that cannot lead to the optimal solution. To this purpose suitable dominance tests are always performed when states are extended, so that the algorithm only records non-dominated states.

The dominance test is the following. Let (S', R', C', i) and (S'', R'', C'', i) be the labels of two states associated to vertex i. Then (S', R', C', i) dominates (S'', R'', C'', i) if

$$\begin{aligned} (a) \quad & S' \leq S'' \\ (b) \quad & R' \leq R'' \\ (c) \quad & C' \leq C'' \end{aligned}$$

and at least one of the inequalities is strict.

Dealing with the RCESPP arising as aprcing subproblem in branch-and-price algorithms for the CVRPTW, Feillet et al. [8] observed that it is sometimes possible to identify vertices which cannot be visited in any extension of a given state, because of the resource limitations. These vertices are called *unreachable*. More formally a vertex k is unreachable from state (S, R, C, i) if there exists a resource r which is non-decreasing, obeys the triangle inequality and is such that extending state (S, R, C, i) to vertex k would generate a state (S', R', C', k) with R'_r exceeding the maximum amount of available resource. This implies that vertex k cannot be reached from (S, R, C, i) in any feasible way. In such cases it is useful to set the consumption of the dummy resources corresponding to the unreachable vertices to 1, as if they had already been visited. Formally, if vertex k is unreachable from state (S, R, C, i), we can set $S_k := 1$. This enhancement allows the dynamic programming algorithm to fathom a larger number of states and to reduce the computation time.

This method can be applied to all three versions of the RCESPP considered here, since capacity and time consumptions are non-negative and satisfy the triangle inequality. In the case of multiple resources, as for the problem with distribution and collection and the problem with capacities and time windows, all of them are used to identify unreachable vertices.

In figure [1] I report the pseudo-code of the dynamic programming algorithm of Feillet et al. in [8]. The notation I use is the following: Γ_i is the list of states associated with vertex i; Δ_i^+ is the set of successors of vertex i; E is the set of vertices to be examined; $Extend(l, k)$ is extension procedures: it extends the state specified as a first argument to a vertex specified as a second argument; this procedure checks the resource constraints and produces only feasible states; finally $EFF(\Gamma, l)$ is the procedure that inserts state l into set Γ applying the domination rules.

4.6 Bounded bi-directional dynamic programming

The dynamic programming algorithm outlined in the previous section generates a number of states which rapidly increases with the size of the problem instance at hand. Every time a label of vertex i is extended, it generates as many other labels as the number of possible successors of i. Therefore in the worst case the number of labels grows exponentially with the number of arcs in the path. States are fathomed only when they are dominated.

I propose here two ideas, that work well together: bi-directional dynamic programming and bounding.

Algorithm 1 RCESPP - Mono-directional dynamic programming

```
// Initialization //
Γ_s ← {(0, 0, 0, s)}
for all i ∈ V \ {s} do
  Γ_i ← ∅
end for
E ← {s}
// Search //
repeat
  // Vertex selection //
  Select i ∈ E
  // Extension //
  for all l_i = (S^i, R^i, C^i, i) ∈ Γ_i do
    for all j ∈ Δ_i^+ do
      if S^i_j = 0 then
        l_j ← Extend(l_i, j)
        Γ_j ← EFF(Γ_j, l_j)
        if Γ_j has changed then
          E ← E ∪ {j}
        end if
      end if
    end for
  end for
  E ← E \ {i}
until E = ∅
```

Bi-directional dynamic programming has been sometimes considered as a useful technique to speed-up Dijkstra's algorithm for the computation of an s-t shortest path on a digraph with non-negative arc weights (see [81]). In the RCESPP when labels are propagated both forward from s to t and backward from t to s the algorithm must examine two subsets of states whose size grows exponentially with the number of arcs in the corresponding forward and backward paths. Due to the exponential dependence on the number of steps, it is intuitive that exploring two smaller sets of states may yield a significant advantage in terms of number of states considered, provided that duplicate solutions are avoided. This is precisely the effect of bounding, whose purpose is to limit the length of the paths corresponding to non-dominates states.

Hereafter I formally define the bounded bi-directional dynamic programming algorithm.

4.6.1 Bi-directional search

In bi-directional search states are extended both forward from vertex s to its successors and backward from vertex t to its predecessors. States, recurrence equations and domination rules are symmetrical to those presented above.

With each vertex $i \in \mathcal{V}$ are associated forward and backward states indicated by $(S^{fw}, R^{fw}, C^{fw}, i)$ and $(S^{bw}, R^{bw}, C^{bw}, i)$, respectively. A path from s to t is detected each time a forward state $(S^{fw}, R^{fw}, C^{fw}, i)$ and a backward state $(S^{bw}, R^{bw}, C^{bw}, j)$ can be feasibly joined.

The backward cost C^{bw} is initialized at 0 at vertex t and whenever a backward state $(S^{bw}, R^{bw}, C^{bw}, j)$ is extended to a state $(S'^{bw}, R'^{bw}, C'^{bw}, i)$. the cost is updated according to the formula:

$$C'^{bw} := C^{bw} + c_{ij} - \lambda_i/2 - \lambda_j/2$$

where $\lambda_i = -\lambda_0$ if $i = s$ and $\lambda_j = -\lambda_0$ if $j = t$.

Forward and backward paths must be joined together to produce complete s-t paths. This can be done subject to to certain feasibility conditions on the resources. In particular a feasibility test on dummy resources S imposes that a same vertex cannot be visited by both paths and a feasibility test on problem-dependent resources R imposes that for each resource the consumption in the overall path does not exceed the overall amount of available resource. Hereafter I define the feasibility tests for each specific case considered.

Capacity. Resource consumption q^{bw} in a backward state associated with vertex j represents the amount of load picked-up at customers visited from j (included) to t. Therefore $(S^{bw}, q^{bw}, C^{bw}, j)$ corresponds to an elementary backward path of cost C^{bw}, originating at j, terminating at t, visiting the vertices indicated by S^{bw} and consuming q^{bw} units of capacity. Initialization and extension of backward labels follow exactly the same rules of forward labels.

The feasibility test on the capacity for joining a forward path $(S^{fw}, q^{fw}, C^{fw}, i)$ with a backward path $(S^{bw}, q^{bw}, C^{bw}, j)$ to produce an s-t path is

$$\begin{aligned} & S_k^{fw} + S_k^{bw} \leq 1 && \forall k \in \mathcal{N} \\ & q^{fw} + q^{bw} \leq Q \end{aligned}$$

and the cost of the resulting s-t path is equal to $C^{fw} - \lambda_i/2 + c_{ij} - \lambda_j/2 + C^{bw}$.

Distribution and collection. Two resources, whose consumption is indicated by π^{bw} and δ^{bw}, are associated with each backward state. Their meaning, initialization and extension rules are symmetrical to those of forward labels: δ^{bw} indicates the amount of load delivered between j and t and π^{bw} indicates the maximum overall amount of load on board of the vehicle between j and t. When a backward path is extended from a state $(S^{bw}, \pi^{bw}, \delta^{bw}, C^{bw}, j)$ to a state $(S'^{bw}, \pi'^{bw}, \delta'^{bw}, C'^{bw}, i)$ along arc (i, j), the update rule is:

$$\pi'^{bw} := \max\{\delta^{bw} + d_i, \pi^{bw} + p_i\}$$
$$\delta'^{bw} := \delta^{bw} + d_i$$

A backward path is feasible only if $\pi^{bw} \leq Q$ and $\delta^{bw} \leq Q$.

The feasibility conditions to join a forward path $(S^{fw}, \pi^{fw}, \delta^{fw}, C^{fw}, i)$ with a backward path $(S^{bw}, \pi^{bw}, \delta^{bw}, C^{bw}, j)$ to produce an s-t path are:

$$S_k^{fw} + S_k^{bw} \leq 1 \qquad \forall k \in \mathcal{N}$$
$$\pi^{fw} + \pi^{bw} \leq Q$$
$$\delta^{fw} + \delta^{bw} \leq Q$$

and the cost of the resulting s-t path is $C^{fw} - \lambda_i/2 + c_{ij} - \lambda_j/2 + C^{bw}$.

Capacity and time windows. In the case of time windows it is useful to define forward and backward time windows $[a_i^{fw}, b_i^{fw}]$ and $[a_i^{bw}, b_i^{bw}]$ as follows:

$$a_i^{fw} = a_i$$
$$a_i^{bw} = a_i + \theta_i$$
$$b_i^{fw} = b_i$$
$$b_i^{bw} = b_i + \theta_i$$

The forward time window represents the range of feasible arrival times at node i, while the backward time window represents the range of feasible departure times from node i. The overall resource availability T is equal to the maximum feasible arrival time at vertex t that is $T = \max_{i \in \mathcal{V}}\{b_i^{fw} + \theta_i + c_{it}\}$.

The time resource consumption τ^{bw} in a backward path associated with vertex j represents the time between the departure from j and the arrival at t. I also consider a capacity resource, whose consumption in backward states is indicated by q^{bw} as in the RCESPP arising from the CVRP.

When a feasible backward path is extended from a state $(S^{bw}, \tau^{bw}, q^{bw}, C^{bw}, j)$ to a state $(S'^{bw}, \tau'^{bw}, q'^{bw}, C'^{bw}, i)$ along arc (i, j), the update rules are:

$$\tau'^{bw} := \max\{\tau^{bw} + \theta_j + c_{ij}, T - b_i^{bw}\}$$
$$q'^{bw} := q^{bw} + d_i$$

A backward path associated with vertex j is feasible only if $\tau^{bw} \leq T - a_j^{bw}$ and $q^{bw} \leq Q$.

When joining a forward path $(S^{fw}, \tau^{fw}, q^{fw}, C^{fw}, i)$ with a backward path $(S^{bw}, \tau^{bw}, q^{bw}, C^{bw}, j)$ the feasibility conditions and the cost of the resulting s-t

path depend on the pair (i, j).
If $i \neq j$ the conditions are:

$$\begin{aligned} & S_k^{fw} + S_k^{bw} \leq 1 && \forall k \in \mathcal{N} \\ & \tau^{fw} + \theta_i + c_{ij} + \theta_j + \tau^{bw} \leq T \\ & q^{fw} + q^{bw} \leq Q \end{aligned}$$

and the cost of the resulting *s*-*t* path is $C^{fw} - \lambda_i/2 + c_{ij} - \lambda_j/2 + C^{bw}$.
Otherwise, if $i = j$, the conditions are:

$$\begin{aligned} & S_k^{fw} + S_k^{bw} \leq 1 && \forall k \in \mathcal{N} \backslash \{i\} \\ & \tau^{fw} + \theta_i + \tau^{bw} \leq T \\ & q^{fw} + q^{bw} \leq Q \end{aligned}$$

and the cost of the resulting *s*-*t* path is $C^{fw} + C^{bw}$.

4.6.2 Search strategy

The set of states generated by the dynamic programming algorithm can be explored according to different search strategies, resembling those used in branch-and-bound algorithms. In this case each vertex has associated a number of non-dominated states, corresponding to paths of different length; here by *length* I mean the number of arcs of a path. Moreover there are both forward and backward paths when the search is bi-directional. The effectiveness of the algorithm may depend on the order in which the states are extended. Here I mention three possible search strategies.

Extension based on path length. This strategy resembles breadth-first search: all paths of minimum length are extended first. Therefore the algorithm generates all non-dominated paths of length $l + 1$ extending all paths of length l.

Extension based on vertices. All vertices are visited in a cyclic order and all states associated to the same vertex are extended.

Extension based on resources. A resource $\hat{r}$ is chosen, whose consumption is monotone along the paths. Non-dominated states are sorted in non-decreasing order of the consumption of $\hat{r}$ and they are extended according to that order.

These strategies can be also mixed together, producing hybrid strategies. In order to have a more significant comparison with the algorithm of Feillet et al. [8], in mine implementation I adopted the extension based on vertices, where for each vertex the states are ordered by non-decreasing resource consumption. When examining a vertex, the algorithm extends both forward and backward states associated with it.

In figure [2] I present the pseudo-code of the bi-directional dynamic programming algorithm. The notation I use is the following: Γ_i^{fw} and Γ_i^{bw} are the lists of forward and backward states associated with vertex i; Δ_i^+ and Δ_i^- are the sets of successors and predecessors of vertex i; E is the set of vertices to be examined; $Extend^{fw}(l, k)$ and $Extend^{bw}(l, k)$ are respectively the forward and backward extension procedures: they extend the state specified as a first argument to a vertex specified as a second argument; these procedures check the resource constraints and produce only feasible states; finally $EFF(\Gamma, l)$ is the procedure that inserts state l into set Γ applying the domination rules.

4.6.3 Bounding

In this context bounding is used to limit the length of forward and backward paths in order to avoid unnecessary duplications: without bounding the same s-t path would be found twice, as a forward path from s to t and as a backward path from t to s. The effect of bounding is to stop the extension of forward and backward paths "at half way" between s and t so that all feasible matchings of forward and backward paths correspond to all feasible complete s-t paths without duplications.

To stop the extension of paths we select a *critical resource*, whose consumption is monotone along the paths, and we allow paths to consume at most half of the available amount of that resource. All non-dominated states generated in this way are recorded, in both directions. Finally all forward and backward states are tentatively matched and checked for feasibility: this produces all feasible s-t paths.

In a branch-and-price context, when the RCESPP is solved as a pricing problem, this procedure may be truncated when a sufficent large number of negative reduced cost columns has been found. In any case it can provide not only the optimal solution but a number of different columns with negative reduces cost, if they exist, and this usually very helpful to speed-up branch-and-price algorithms.

Hereafter I describe how I chose the critical resource for each different kind of problem.

Capacity. The critical resource in this case is obviously the only resource, that is capacity. Forward and backward states are extended only if their associated resource consumption value, q^{fw} or q^{bw} respectively, is less than $Q/2$, where Q is the vehicle capacity.

Distribution and collection. In this case there are two resources; I consider as a critical resource the sum of the resource consumptions $\rho^{fw} = \pi^{fw} + \delta^{fw}$ for forward states and $\rho^{bw} = \pi^{bw} + \delta^{bw}$ for backward states and I impose the bounding condition $\rho \leq Q$ in both cases.

Capacity and time windows. In this last case I consider time as the critical resource and I impose $\tau^{fw} \leq T/2$ to forward states and $\tau^{bw} \leq T/2$ to backward states.

4.6.4 Solutions uniqueness

The last algorithmic detail I present directly comes from the need of generating many different columns with negative reduced cost when I solve the pricing problem in a branch-and-price framework. The bounded bi-directional dynamic programming algorithm can still provide duplicate solutions: consider for instance an s-t path including vertices i, j and k in this order. If the resource constraints are not tight, it is possible that forward states for vertices i and j and backward states for vertices j and k are generated. Therefore the same solution is obtainable by joining a forward state of i with a backward state of j as well as joining a forward state of j with a backward state of k. If only the optimal solution is sought, these duplicates are discarded with no additional computational effort, when they are evaluated, since they have the same cost. But if one needs to store in some data-structure all columns with negative reduced cost, the duplicate columns cannot be discarded on the basis of their cost and their identification may be computationally expensive. For this reason I have devised an additional test, represented by the function $HalfWay$ in the pseudo-code of figure [3]. The meaning of this test is that we accept an s-t path only when it is produced by the join of a forward state and a backward state, for which the consumption of the critical resource is not more than half of the overall consumption *for that s-t path*, that is the two states are across the "half way point" along the s-t path. This condition must be specified for each particular problem, as follows.

Capacity The "half-way-point" test on a pair of forward and backward states is

$$\begin{aligned} q^{fw} &<= (q^{fw} + q^{bw})/2 \\ q^{bw} &< (q^{fw} + q^{bw})/2 \end{aligned} \tag{4.1}$$

Distribution and collection The "half-way-point" test on a pair of forward and backward states is

$$\begin{aligned} \rho^{fw} &<= (\rho^{fw} + \rho^{bw})/2 \\ \rho^{bw} &< (\rho^{fw} + \rho^{bw})/2 \end{aligned} \tag{4.2}$$

Capacity and time windows The "half-way-point" test on a pair of forward and backward states associated with a same vertex i is

$$\begin{aligned} \tau^{fw} &<= (\tau^{fw} + \theta_i + \tau^{bw})/2 \\ \tau^{bw} &< (\tau^{fw} + \theta_i + \tau^{bw})/2 \end{aligned} \tag{4.3}$$

while the test for a forward state associated with vertex i and a backward state associated with vertex j, with $i \neq j$, is

$$\begin{aligned} \tau^{fw} + \theta_i &<= (\tau^{fw} + \theta_i + c_{ij} + \theta_j + \tau^{bw})/2 \\ \tau^{bw} + \theta_j &< (\tau^{fw} + \theta_i + c_{ij} + \theta_j + \tau^{bw})/2 \end{aligned} \tag{4.4}$$

Since for each solution there is only one pair of forward and backward states around the “half way point”, this test guarantees that each s-t path is generated only once.

In figure [3] I present the pseudo-code for the joining procedure of the bi-directional bounded dynamic programming algorithm. I use the following terminology: $Feasible(l_i, l_j)$ checks the resource compatibility of states l_i and l_j according to problem-dependent rules; $HalfWay(l_i, l_j)$ checks if the s-t path obtainable joining the two states l_i and l_j satisfies the "half-way-point” conditions; $Save(l_i, l_j)$ saves the solution obtained from the two states l_i and l_j;

4.7 State space relaxation for the RCESPP

State space relaxation was introduced by Christofides et al. [67] in 1981. The state space $\mathcal{S}$ explored by the dynamic programming algorithm is projected onto a lower dimensional space $\mathcal{T}$ so that each state in $\mathcal{T}$ retains the minimum cost among those of its corresponding states in $\mathcal{S}$ (assuming the objective function must be minimized). In this way the number of states to be explored is drastically reduced; the drawback is that some original state corresponding to an infeasible solution in $\mathcal{S}$ may be projected onto a state corresponding to a feasible solution in $\mathcal{T}$ and therefore the search in the relaxed state space does not guarantee to find an optimal solution but rather a lower bound.

State space relaxation has been used as a method alternative to exact optimization of the pricing problem in branch-and-price algorithms for the VRP with additional constraints (see for instance Desrochers et al. [49]): instead of the optimal value of the pricing problem, a lower bound is obtained. This allows faster convergence of the column generation algorithm at the expense of a weaker lower bound. Columns containing cycles must be eliminated through branching. Here on the contrary I focus on the use of state space relaxation for the exact optimization of the pricing problem by a branch-and-bound algorithm. I are not aware of any previous attempt to use state space relaxation to this purpose.

Our state space relaxation consists of mapping each state (S, R, C, i) onto a new state (σ, R, C, i), where $\sigma = \sum_{k=1}^{N} S_k$ represents the length of the path, that is the number of vertices visited (excluding s). Since each component of the resource consumption vector R may take on a finite number of values and σ can vary between 0 and N, a dynamic programming algorithm based on state space relaxation must explore only a pseudo-polynomial number of states. ¿From the viewpoint of complexity and computing time this makes a big difference with respect to the exact dynamic programming algorithm in which vector S yields an exponential number of possible states. The surrogate resource consumption σ is initialized as 0 and it is increased by one unit each time a state is extended. Since the state does no longer keep information about the set of already visited vertices, cycles are no longer forbidden; therefore the path is guaranteed to be feasible with respect to the resource constraints but it is not guaranteed to be elementary.

In the state space relaxation algorithm the domination rule is modified as follows: a state (σ', R', C', i) dominates a state (σ'', R'', C'', i) only if

$$\begin{aligned} \sigma' &\leq \sigma'' \\ R' &\leq R'' \\ C' &\leq C'' \end{aligned}$$

and at least one of the inequalities is strict.

This state space relaxation of the RCESPP into the RCSPP can be tightened by eliminating all cycles of length two. This is easily accomplished by a duplication of the labels (see for instance Desrochers et al. [49]). Irnich and Villeneuve [83] have also proposed a method to eliminate cycles of length $k \geq 3$, but the computational complexity of their method dramatically increases with k. Hence I incorporated in our algorithms the technique to avoid cycles of length two.

The definitions above apply to both forward and backward states when bi-directional search is employed. In such case σ^{fw} and σ^{bw} represent respectively the number of forward extensions from s and the number of backward extensions from t. I bound bi-directional search in the same way described above, that is on the basis of the value of a critical resource.

When bounded bi-directional search is coupled with state space relaxation the join of forward and backward paths becomes critical: both the forward path and the backward path to be joined may contain cycles; moreover a cycle can be produced by the join, even if the two paths are elementary. These two cases are illustrated in figure 4.1. In addition there may be many different ways to join forward and backward paths providing the same solution. The former issue is addressed in the next section, where branching strategies are illustrated; the latter is addressed hereafter.

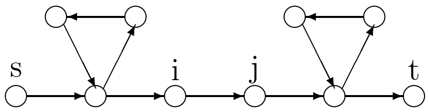

Figure 4.1: an s-t path made of non-elementary paths s-i and j-t.

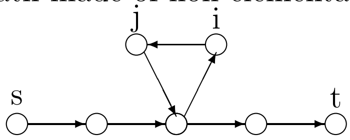

Figure 4.2: a non-elementary s-t path made of elementary paths s-i and j-t.

4.7.1 Paths join and solutions uniqueness

The bounded bi-directional dynamic programming algorithm can provide duplicate solutions: consider for instance an s-t path including vertices i, j and k in this order. If the constraint on the critical resource is not tight, it is possible that forward states for vertices i and j and backward states for vertices j and k are generated. Therefore the same s-t path can be obtained by joining a forward state of i with a backward state of j as well as joining a forward state of j with a backward state of k. This unpleasant phenomenon can be avoided with an additional test: we accept an s-t path only when it is produced by the join of a forward state and a backward state, for which the forward and backward consumptions of the critical resource are as close as possible to half the overall consumption *for that s-t path*, that is the two states are as close as possible to the "half way point" along the s-t path. Let r^{fw} and r^{bw} be the critical resource consumptions in forward and backward paths. Among all possible pairs of forward and backward states producing the same s-t path we choose the one for which $\phi = |r^{fw} - r^{bw}|$ is minimum. The test is done in constant time for each candidate pair of states, since the position closest to the "half-way point" is detected by direct comparison with the next position along the path if $r^{fw} < r^{bw}$ and with the previous position if $r^{fw} > r^{bw}$. In case of tie between two positions for which ϕ is minimum, we choose the one with $r^{fw} > r^{bw}$. This test guarantees that each s-t path is generated only once.

4.7.2 Solving the RCESPP with state space relaxation and Branch-and-bound

In this section I describe a branch-and-bound algorithm which solves the RCESPP to optimality, exploiting the RCSPP lower bound given by the bounded bi-directional dynamic programming algorithm with state space relaxation. In section 4.7.3 I describe the branching policies needed to eliminate cycles: every time the optimal solution of the RCSPP is not elementary, the current node of the search tree is replaced by children nodes in which some additional constraints are added to the RCSPP.

Search policy. The search policy I use to explore the branch-and-bound tree is best-first, that is the open nodes of the tree are ranked according to the value of their associated lower bound and the most promising node is explored first.

Upper bounding. At each node of the branch-and-bound tree and at each iteration of the column generation algorithm a feasible solution is computed with a nearest neighbor heuristic. Starting from the depot s the most convenient vertex among the feasible ones is chosen until the path reaches t. For a vertex to be feasible we check that no resource constraint is exceeded and the vertex have not been visited yet. At each vertex i the algorithm chooses the next feasible vertex j such that

$$j = argmin_k\{c_{ik} - \lambda_k\}$$

4.7.3 Branching strategies

I present three different ways to perform branching, namely branching on cycles, branching on arcs and branching on resources. Our algorithm uses hybrid branching strategies in which all these techniques are exploited.

Branching on cycles. First we determine the minimum length cycle in the optimal RCSPP solution. Then k children nodes are generated, where k is the length of the cycle, that is the number of arcs traversed between two visits to the same vertex: at child node $h = 0, \ldots, k-1$ we fix the first h arcs of the cycle and we forbid the $h+1$-th arc. I experimentally observed that, forbidding cycles of length 2, k was very often equal to 3.

Branching on arcs. This binary branching scheme consists of selecting a vertex entered or left by more than one arc in the RCSPP solution. One of these arcs is then fixed in one child node and forbidden in the other.

Branching on resources. When the optimal solution of the RCSPP has a cycle, there exists at least one vertex $\hat{\imath}$ that is visited more than once. The branching strategy consists of adding a constraint on the quantity of critical resource consumed up to the visit of vertex $\hat{\imath}$. This idea was proposed by Gélinas et al. [82] for routing problems with time windows and it can be adapted to any problem with a critical resource whose consumption r is strictly monotone along the path.

Given a branching vertex $\hat{\imath}$, let r' and r'' the two values of resource consumption in two states associated with $\hat{\imath}$ with $r' < r''$. Then an integer value $\bar{r}$ is chosen such that $r' < \bar{r} \leq r''$. Two children nodes are generated imposing that the value of r at vertex $\hat{\imath}$ satisfies $r \geq \bar{r}$ in one child node and $r \leq \bar{r} - 1$ in the other.

It is remarkable that the dynamic programming algorithm that computes the lower bound can easily take into account the constraints imposed by all branching techniques. In particular fixing and forbidding arcs is easy to take into account at children nodes: when arc (i, j) is forbidden, it is deleted from the graph; when arc (i, j) is fixed, all arcs leaving i and all arcs entering j, excepted arc (i, j), are deleted from the graph.

The consequence of branching on the critical resource is that each vertex has an associated window $[a^r, b^r]$ of feasible values for the critical resource; when a path reaches that vertex with a critical resource consumption less than a^r, the consumption is set to a^r; when it reaches the vertex with a critical resource consumption greater than b^r, it is declared infeasible and it is discarded. This rule can be applied to both forward and backward states, with different resource windows for constraining forward and backward consumptions.

I obtained the best results when I employed hybrid branching strategies in our branch-and-bound algorithm. If either the forward path or the backward path forming the optimal RCSPP solution contains a cycle, I branch on the critical resource: I choose for branching the first vertex visited more than once which is encountered moving along the forward (resp. backward) path from s to t (resp. from t to s); I consider r' and r'' as the resource consumptions at the first (resp. last) two visits of the branching vertex and I choose $\bar{r} = \lceil \frac{r'+r''}{2} \rceil$. If the forward and the backward paths are both elementary but a cycle is generated by their join, I branch on arcs or cycles. When I branch on arcs, the branching vertex is the first vertex visited more than once which is encountered when moving along the path from the half way point forward.

I could not observe a clear domination between the hybrid branching strategies on resource/arcs and resource/cycles.

4.8 Decremental state space relaxation

The exact dynamic programming algorithm forbids multiple visits for each vertex, while the algorithm with state space relaxation does not. I pursued a compromise between these two extreme cases by the following idea: some vertices are identified as *critical*, according to the structure of the optimal RCSPP solution obtained with state space relaxation. Let Θ indicate the set of critical vertices at the current iteration. In the subsequent iteration the dynamic programming algorithm prevents multiple visits the vertices in Θ, still allowing multiple visits to the others. This is easily accomplished by extending the state space relaxation labels with a binary vector S_Θ playing the same role as S in exact dynamic programming. The size of S_Θ is however restricted only to the critical vertices. When S_Θ contains all the vertices the algorithm is equivalent to exact dynamic programming; when S_Θ is empty it is equivalent to the algorithm with state space relaxation. Therefore I indicate this algorithm by decremental state space relaxation (DSSR). The algorithm is run iteratively: every time it produces an optimal solution with cycles, the vertices visited more than once are marked as critical and the algorithm restarts. Let Ψ the set of vertices visited more than once in the optimal solution computed by the DSSR algorithm. If Ψ is not empty, then another iteration is performed with a set of critical vertices equal to $\Theta' = \Theta \cup \Psi$. Hence the set of critical vertices is enlarged at each iteration and eventually the algorithm provides the optimal solution to the RCESPP without having recourse to branching. I report hereafter the pseudo-code of the decremental state space relaxation algorithm. where S_Θ is the vector of dummy resources associated to the critical vertices; procedure $MultipleVisits$ returns the set Ψ of vertices visited more than once in the current optimal path.

4.9 Experimental Results

I derived the benchmark instances for the RCESPP from Solomon's dataset [61]. For each kind of RCESPP problem I tested the algorithms on two classes of instances obtained from Solomon's instances by considering the first 50 and 100 nodes. These datasets are divided into *random*, *clustered* and *random-clustered* categories, according to the displacement of the customers. Instances belonging to the same dataset have the customers located in the same way and with the same delivery requests; the instances differ only for the time windows.

When solving the RCESPP with capacities I considered one instance taken from each one of the three Solomon's testsets, I kept the original customer locations and demands and I neglected the time windows. Then I derived from each original instance 10 RCESPP instances with 50 nodes and 10 RCESPP instances with 100 nodes. In both cases the vehicle capacity varies from 10 to 100 with an increasing step of 10.

For the RCESPP with distribution and collection I kept the original delivery requests and I derived the pickup requests as follows: $p_i = \lfloor 0.8 d_i \rfloor$ if i is odd and $p_i = \lfloor 1.2 d_i \rfloor$ if i is even. I varied the capacity as in the previous case.

Finally, for the RCESPP with capacities and time windows I considered the original instances of Solomon's dataset.

In addition I defined another dataset built on the difficult Solomon's instance c_104; I kept the original starting times of the time windows, a_i, and I set the end times as follows: $b_i = a_i + (1+\gamma)\theta_i$ for $\gamma = 0.25 * k$ and $k = 0, \ldots, 24$, where θ_i is the original service time at vertex i.

I generated the dual variables λ_i as random integer variables uniformly distributed in $\{0, \ldots, 20\}$. I set the limit to 20, as proposed by [8], in order to have a reasonable number of negative arcs. I rounded up all the Euclidean distances between customers to one decimal point values in order to mantain the triangle inequality.

All tests were performed on a PC equipped with a PentiumIV 1.6GHz processor with 512Mb RAM. The algorithms have been coded in ANSI-C and compiled with gcc 3.0.4.

Tables 4.1 to 4.8 report on the experimental comparison between the mono-directional dynamic programming algorithm, the bi-directional algorithm without bounds and the bi-directional algorithm with bounds. For each algorithm I report the total number of non-dominated states generated at the end of the extension procedure and the time needed to compute the optimal path. Empty cells mean that the solution has not been computed within the time limit of one hour.

Capacities. Results reported in Tables 4.1 and 4.2 show that the bi-directional algorithm with bounds definitely outperforms the other two on all instances. For the loosely constrained instances it reduces the computing time by one order of magnitude and it reduces significantly the number of non-dominated states. The bi-directional algorithm without bounds outperforms the mono-directional algorithm only on c-instances, it has comparable performances on rc-instances and it is more time-consuming on r-instances. This indicates that its performances are more dependent on the structure of the cost matrix than the resource availability. For 100 nodes instances one should notice that the space and the time complexity grow very rapidly for all three algorithms. However the bounded bi-directional algorithm solves more and larger instances than the mono-directional algorithm and it reduces the computing time by an order of magnitude.

Distribution and collection. When solving the RCESPP with distribution and collection I obtained results similar to those above: the results are reported in Tables 4.3 and 4.4. The bounded bi-directional algorithm solved all instances with 50 nodes in less than 320 seconds and it failed to solve 6 instances with 100 nodes.

Capacities and time windows. All but one of Solomon's instances with 50 and 100 nodes were solved by the bounded bi-directional algorithm as reported in Tables 4.5 and 4.6. The superiority of the bounded bi-directional algorithm is quite evident and systematic.

Tightness of the constraints.

Table 4.1: RCESPP with capacity - 50 vertices

Instance	Monodirectional		Bidirectional		Bidirectional + Bounding	
Name	Labels	Time	Labels	Time	Labels	Time
c101_50_01	30	0.00	56	0.00	56	0.00
c101_50_02	104	0.00	118	0.00	268	0.00
c101_50_03	311	0.01	530	0.01	692	0.00
c101_50_04	885	0.05	1169	0.03	2574	0.03
c101_50_05	2593	0.28	3924	0.12	4692	0.07
c101_50_06	8707	2.47	10926	1.07	15236	0.91
c101_50_07	30973	26.30	30612	5.16	23394	1.75
c101_50_08	111814	287.50	87141	54.77	75026	20.35
c101_50_09	393680	3240.86	203586	194.61	101128	33.24
c101_50_10			574981	2217.46	331402	394.97
c101_50_11					430032	615.10
r101_50_01	40	0.00	62	0.00	62	0.00
r101_50_02	135	0.00	225	0.01	210	0.01
r101_50_03	312	0.00	562	0.02	525	0.01
r101_50_04	652	0.04	1196	0.06	1250	0.02
r101_50_05	1345	0.09	2566	0.17	2418	0.05
r101_50_06	2868	0.24	5513	0.44	4570	0.11
r101_50_07	6296	0.77	12184	1.35	7874	0.24
r101_50_08	14226	2.91	27188	4.80	13590	0.60
r101_50_09	32561	12.25	62306	20.64	22800	1.49
r101_50_10	73456	52.40	143040	97.54	36838	3.87
r101_50_11	159720	225.74	306718	409.13	59911	10.15
rc101_50_01	21	0.00	42	0.00	44	0.00
rc101_50_02	87	0.00	103	0.00	124	0.00
rc101_50_03	164	0.00	250	0.00	268	0.00
rc101_50_04	302	0.01	421	0.02	560	0.01
rc101_50_05	511	0.02	810	0.04	800	0.01
rc101_50_06	876	0.05	1367	0.07	1551	0.03
rc101_50_07	1331	0.10	2246	0.14	1774	0.04
rc101_50_08	2038	0.18	3346	0.23	3217	0.08
rc101_50_09	3115	0.35	5134	0.42	3322	0.09
rc101_50_10	4846	0.67	7710	0.74	5864	0.19
rc101_50_11	7740	1.62	12442	1.53	6050	0.22

Algorithm 2 RCESPP - Bi-directional dynamic programming

```
// Initialization //
Γ_s^{fw} ← {(0, 0, 0, s)}
Γ_t^{bw} ← {(0, 0, 0, t)}
for all i ∈ V \ {s} do
   Γ_i^{fw} ← ∅
end for
for all i ∈ V \ {t} do
   Γ_i^{bw} ← ∅
end for
E ← {s, t}
// Search //
repeat
   // Vertex selection //
   Select i ∈ E
   // Forward extension //
   for all l_i = (S^i, R^i, C^i, i) ∈ Γ_i^{fw} do
      for all j ∈ Δ_i^+ such that S_j^i = 0 do
         l_j ← Extend^{fw}(l_i, j)
         Γ_j^{fw} ← EFF(Γ_j^{fw}, l_j)
         if Γ_j^{fw} has changed then
            E ← E ∪ {j}
         end if
      end for
   end for
   // Backward extension //
   for all l_i = (S^i, R^i, C^i, i) ∈ Γ_i^{bw} do
      for all k ∈ Δ_i^- such that S_k^i = 0 do
         l_k ← Extend^{bw}(l_i, k)
         Γ_k^{bw} ← EFF(Γ_k^{bw}, l_k)
         if Γ_k^{bw} has changed then
            E ← E ∪ {k}
         end if
      end for
   end for
   E ← E \ {i}
until E = ∅
// Join between forward and backward paths //
Join
```

Algorithm 3 RCESPP - Bi-directional dynamic programming: Join

for all $i \in \mathcal{V}$ **do**
  for all $l_i = (S^{fw}, R^{fw}, C^{fw}, i) \in \Gamma_i^{fw}$ **do**
    for all $j \in \mathcal{V}$ **do**
      for all $l_j = (S^{bw}, R^{bw}, C^{bw} \in \Gamma_j^{bw}$ **do**
        if $C^{fw} - \lambda_i/2 + c_{ij} - \lambda_j/2 + C^{bw} < 0$ **then**
          if $Feasible(l_i, l_j)$ AND $HalfWay(l_i, l_j)$ **then**
            $Save(l_i, l_j)$
          end if
        end if
      end for
    end for
  end for
end for

Table 4.2: RCESPP with capacity - 100 vertices

Instance	Monodirectional		Bidirectional		Bidirectional + Bounding	
Name	Labels	Time	Labels	Time	Labels	Time
c101_100_01	55	0.00	106	0.00	106	0.00
c101_100_02	205	0.01	237	0.01	559	0.01
c101_100_03	640	0.09	1084	0.04	1456	0.03
c101_100_04	2136	0.41	2765	0.24	5900	0.19
c101_100_05	7056	2.49	9396	0.86	11546	0.44
c101_100_06	26135	21.87	29153	7.51	45138	6.60
c101_100_07	116247	327.42	81488	30.36	75698	13.78
c101_100_08			294184	517.75	310651	276.88
c101_100_09					4799333	520.82
c101_100_10						
r101_100_01	163	0.00	248	0.01	266	0.00
r101_100_02	1076	0.09	1946	0.16	2120	0.06
r101_100_03	5106	1.24	9508	2.25	11866	0.70
r101_100_04	25613	19.59	47792	34.46	53668	8.37
r101_100_05	133007	417.56	249482	684.78	215976	104.41
r101_100_06					764476	1300.33
r101_100_07						
r101_100_08						
r101_100_09						
r101_100_10						
rc101_100_01	21	0.00	43	0.00	90	0.00
rc101_100_02	257	0.01	428	0.02	636	0.01
rc101_100_03	705	0.06	1308	0.13	1732	0.04
rc101_100_04	1857	0.28	3506	0.55	5706	0.23
rc101_100_05	5024	1.20	9760	2.33	12561	0.69
rc101_100_06	14260	5.86	27968	11.55	29786	2.80
rc101_100_07	40375	31.40	78906	59.59	60499	9.74
rc101_100_08	111591	181.25	219142	351.14	124752	37.46
rc101_100_09	299056	1086.05	585850	2083.42	237652	130.20
rc101_100_10					459269	470.24

Algorithm 4 RCESPP - Decremental state space relaxation

// Initialization //
$\Psi \leftarrow \emptyset$
$\Theta \leftarrow \emptyset$
repeat
  $\Theta \leftarrow \Theta \cup \Psi$
  $\Gamma_s^{fw} \leftarrow \{(\mathbf{0}, \mathbf{0}, 0, s)\}$
  $\Gamma_t^{bw} \leftarrow \{(\mathbf{0}, \mathbf{0}, 0, t)\}$
  for all $i \in \mathcal{V} \setminus \{s\}$ **do** $\Gamma_i^{fw} \leftarrow \emptyset$
  for all $i \in \mathcal{V} \setminus \{t\}$ **do** $\Gamma_i^{bw} \leftarrow \emptyset$
  $E \leftarrow \{s, t\}$
  // Search //
  repeat
    // Vertex selection //
    Select $i \in E$
    // Forward extension //
    for all $l_i = (S_\Theta^i, R^i, C^i, i) \in \Gamma_i^{fw}$ **do**
      for all $j \in \Delta_i^+$ such that $j \notin \Theta$ **or** $S_j^i = 0$ **do**
        $l_j \leftarrow Extend^{fw}(l_i, j)$
        $\Gamma_j^{fw} \leftarrow EFF(\Gamma_j^{fw}, l_j)$
        if Γ_j^{fw} has changed **then** $E \leftarrow E \cup \{j\}$
    // Backward extension //
    for all $l_i = (S_\Theta^i, R^i, C^i, i) \in \Gamma_i^{bw}$ **do**
      for all $k \in \Delta_i^-$ such that $k \notin \Theta$ **or** $S_k^i = 0$ **do**
        $l_k \leftarrow Extend^{bw}(l_i, k)$
        $\Gamma_k^{bw} \leftarrow EFF(\Gamma_k^{bw}, l_k)$
        if Γ_k^{bw} has changed **then** $E \leftarrow E \cup \{k\}$
    $E \leftarrow E \setminus \{i\}$
  until $E = \emptyset$
  // Join between forward and backward paths //
  for all $i \in \mathcal{V}$
    for all $l_i = (S^i, T^i, C^i, i) \in \Gamma_i^{fw}$
      for all $j \in \mathcal{V}$
        for all $l_j = (S^j, T^j, C^j, j) \in \Gamma_j^{bw}$
          if $Feasible(l_i, l_j)$ **and** $HalfWay(l_i, l_j)$
           then $Save(l_i, l_j)$
  // Search for vertices visited more than once //
  $\Psi \leftarrow MultipleVisits()$
until $\Psi = \emptyset$

Table 4.3: RCESPP with distribution and collection - 50 vertices

Instance	Monodirectional		Bidirectional		Bidirectional + Bounding	
Name	Labels	Time	Labels	Time	Labels	Time
c101_50_01	25	0.00	26	0.00	26	0.00
c101_50_02	191	0.00	159	0.00	159	0.00
c101_50_03	1127	0.01	654	0.00	554	0.01
c101_50_04	4788	0.19	2915	0.03	1751	0.02
c101_50_05	21420	4.30	7882	0.17	4675	0.07
c101_50_06	88706	79.75	27123	3.10	11311	0.41
c101_50_07	346218	1201.05	58244	9.91	24006	1.72
c101_50_08			194732	156.31	51401	7.95
c101_50_09			384974	436.79	110354	35.46
c101_50_10					233478	165.21
c101_50_11					474147	672.29
r101_50_01	51	0.00	61	0.00	59	0.00
r101_50_02	207	0.01	208	0.01	188	0.00
r101_50_03	633	0.01	1089	0.02	486	0.01
r101_50_04	1910	0.04	3329	0.07	1113	0.02
r101_50_05	5338	0.23	9593	0.38	2085	0.04
r101_50_06	13925	1.53	23545	2.07	3882	0.10
r101_50_07	34947	9.65	57109	11.75	6986	0.23
r101_50_08	83238	52.00	127163	56.77	12138	0.51
r101_50_09	188997	257.86	276646	245.08	20384	1.23
r101_50_10	410572	1695.94	405789	1420.66	33107	3.15
r101_50_11					55835	8.19
rc101_50_01	23	0.00	24	0.00	24	0.00
rc101_50_02	96	0.00	83	0.00	83	0.00
rc101_50_03	231	0.01	216	0.00	199	0.01
rc101_50_04	511	0.01	531	0.01	397	0.01
rc101_50_05	1104	0.02	1230	0.02	764	0.02
rc101_50_06	2080	0.07	2265	0.05	1108	0.02
rc101_50_07	3797	0.19	4167	0.11	1817	0.03
rc101_50_08	6807	0.63	6905	0.25	2546	0.05
rc101_50_09	12367	2.17	11690	0.61	3435	0.10
rc101_50_10	22823	7.55	20541	2.21	4998	0.15
rc101_50_11	42162	25.40	35904	6.56	6213	0.23

Table 4.4: RCESPP with distribution and collection - 100 vertices

Instance	Monodirectional		Bidirectional		Bidirectional + Bounding	
Name	Labels	Time	Labels	Time	Labels	Time
c101_100_01	47	0.00	48	0.00	48	0.00
c101_100_02	382	0.00	328	0.01	328	0.00
c101_100_03	2415	0.08	1348	0.01	1179	0.02
c101_100_04	13009	1.42	6409	0.20	3829	0.08
c101_100_05	83462	49.91	17473	0.89	11112	0.41
c101_100_06	520592	1999.57	75587	17.39	30823	2.62
c101_100_07			171868	57.81	76548	12.72
c101_100_08			770885	1902.60	197386	87.88
c101_100_09					509042	516.42
c101_100_10						
r101_100_01	245	0.01	310	0.00	253	0.00
r101_100_02	3688	0.21	6208	0.30	1948	0.06
r101_100_03	43242	20.67	69800	26.2000	10874	0.68
r101_100_04	409513	1806.75	636444	1943.60	49258	7.34
r101_100_05					189041	84.00
r101_100_06					676338	1040.25
r101_100_07						
r101_100_08						
r101_100_09						
r101_100_10						
rc101_100_01	72	0.00	85	0.00	67	0.00
rc101_100_02	401	0.00	641	0.01	501	0.01
rc101_100_03	1950	0.07	3477	0.12	1422	0.04
rc101_100_04	8290	0.70	15362	1.27	4540	0.17
rc101_100_05	32216	8.19	57397	13.31	10790	0.55
rc101_100_06	117793	98.23	202545	141.49	25657	2.29
rc101_100_07	418620	1109.78	643159	1267.20	52378	7.73
rc101_100_08					107414	28.62
rc101_100_09					207049	99.56
rc101_100_10						

Table 4.5: RCESPP with capacity and time windows - 50 vertices

Instance Name	Monodirectional Labels	Monodirectional Time	Bidirectional Labels	Bidirectional Time	Bidirectional + Bounding Labels	Bidirectional + Bounding Time
c101_50	524	0.02	808	0.04	323	0.01
c102_50	4548	0.93	5822	0.66	3026	0.14
c103_50	106795	393.47	58868	49.32	30736	10.25
c104_50						
c105_50	609	0.03	892	0.05	435	0.02
c106_50	565	0.03	826	0.04	359	0.01
c107_50	652	0.04	941	0.06	504	0.02
c108_50	1019	0.07	1525	0.12	736	0.04
c109_50	2255	0.22	3399	0.34	2031	0.15
r101_50	166	0.00	344	0.01	121	0.00
r102_50	663	0.03	1577	0.06	596	0.02
r103_50	2546	0.16	5859	0.37	2322	0.06
r104_50	32697	10.55	47780	11.86	15441	0.66
r105_50	344	0.01	625	0.01	238	0.01
r106_50	970	0.04	2184	0.10	818	0.02
r107_50	3457	0.24	7638	0.55	2784	0.08
r108_50	34460	12.36	50473	13.84	16457	0.75
r109_50	683	0.03	1235	0.05	584	0.02
r110_50	2003	0.12	3600	0.22	1600	0.04
r111_50	2571	0.19	5156	0.35	2289	0.07
r112_50	4552	0.39	9153	0.74	3987	0.12
rc101_50	386	0.01	845	0.01	270	0.00
rc102_50	1368	0.04	2990	0.08	906	0.01
rc103_50	4788	0.42	10217	0.83	3509	0.07
rc104_50	12805	3.47	27732	6.92	8801	0.28
rc105_50	1208	0.03	2654	0.08	937	0.01
rc106_50	1194	0.04	2587	0.07	889	0.01
rc107_50	5380	0.31	9894	0.57	3525	0.07
rc108_50	12780	2.29	24967	3.31	10166	0.21

Table 4.6: RCESPP with capacity and time windows - 100 vertices

Instance	Monodirectional		Bidirectional		Bidirectional + Bounding	
Name	Labels	Time	Labels	Time	Labels	Time
c101_100	994	0.16	1658	0.31	679	0.06
c102_100	18126	22.53	23422	16.82	11839	2.99
c103_100					123804	133.56
c104_100						
c105_100	1149	0.23	1839	0.43	915	0.11
c106_100	1448	0.37	2609	0.70	1159	0.16
c107_100	1225	0.31	1960	0.58	1058	0.16
c108_100	2094	0.64	3643	1.24	1690	0.33
c109_100	4739	2.03	7962	3.48	4608	1.28
r101_100	746	0.04	1474	0.10	452	0.01
r102_100	36969	49.66	109340	169.09	14792	2.21
r103_100					135575	95.73
r104_100					655858	1242.56
r105_100	2191	0.21	4524	0.48	1161	0.06
r106_100	52182	126.21	8712	420.03	22970	5.52
r107_100					138027	100.83
r108_100					570910	891.81
r109_100	6389	1.35	12914	2.76	3504	0.37
r110_100	39042	47.09	88069	109.33	25063	4.89
r111_100					69890	25.86
r112_100					394702	647.36
rc101_100	1196	0.09	2954	0.2100	955	0.03
rc102_100	8268	1.73	23405	5.2500	5384	0.30
rc103_100	76457	100.67			38308	6.01
rc104_100					232961	148.73
rc105_100	3253	0.45	9522	1.18	2964	0.15
rc106_100	3130	0.44	8675	1.12	2574	0.12
rc107_100	14224	3.56	36463	11.74	10505	0.72
rc108_100	57637	46.54	150081	151.20	45430	6.75

Table 4.7: RCESPP with capacity and time windows - c_104, 50 vertices

Instance	Monodirectional		Bidirectional		Bidirectional + Bounding	
Name	Labels	Time	Labels	Time	Labels	Time
c104_50_01	96	0.00	260	0.01	59	0.00
c104_50_02	166	0.00	494	0.00	126	0.00
c104_50_03	246	0.01	1224	0.03	200	0.01
c104_50_04	257	0.01	1739	0.08	206	0.01
c104_50_05	271	0.01	1876	0.13	208	0.01
c104_50_06	370	0.01	1980	0.14	269	0.01
c104_50_07	614	0.02	3379	0.20	475	0.01
c104_50_08	730	0.04	5744	0.52	586	0.04
c104_50_09	871	0.05	8822	1.56	730	0.05
c104_50_10	991	0.07	8977	1.79	804	0.06
c104_50_11	1751	0.11	11861	2.43	1440	0.09
c104_50_12	2664	0.23	18914	6.74	2292	0.21
c104_50_13	4158	0.48	36920	18.66	3771	0.43
c104_50_14	4495	0.58	41282	22.38	4031	0.53
c104_50_15	6257	0.93	44915	24.33	5508	0.74
c104_50_16	10426	2.55	55163	40.70	9411	2.10
c104_50_17	20072	6.01	140626	282.45	18579	5.02
c104_50_18	23086	7.52	167141	342.34	21738	6.54
c104_50_19	27539	11.12	179777	364.69	25638	9.25
c104_50_20	39652	27.11			36762	23.16
c104_50_21	89920	97.87			81804	73.19
c104_50_22	112830	136.46			105756	110.25
c104_50_23	135902	189.24			126645	149.81
c104_50_24	170507	350.34			157915	275.93
c104_50_25					323641	1016.98

Table 4.8: RCESPP with capacity and time windows - c_104, 100 vertices

Instance	Monodirectional		Bidirectional		Bidirectional + Bounding	
Name	Labels	Time	Labels	Time	Labels	Time
c104_100_01	199	0.00	554	0.03	160	0.01
c104_100_02	299	0.02	842	0.04	227	0.01
c104_100_03	447	0.03	2198	0.13	368	0.02
c104_100_04	495	0.05	3280	0.38	415	0.04
c104_100_05	510	0.05	3622	0.72	427	0.05
c104_100_06	698	0.07	4070	0.99	523	0.06
c104_100_07	1121	0.13	6585	1.45	895	0.11
c104_100_08	1416	0.23	11646	3.17	1153	0.19
c104_100_09	1685	0.36	18558	6.86	1393	0.29
c104_100_10	1882	0.44	20809	9.28	1557	0.37
c104_100_11	3105	0.71	26550	13.40	2655	0.59
c104_100_12	5122	1.43	43049	33.17	4504	1.24
c104_100_13	8168	2.74	85440	93.45	7336	2.37
c104_100_14	9244	3.74	112634	146.83	8457	3.30
c104_100_15	12088	5.22	130779	187.82	10932	4.49
c104_100_16	20841	11.40	172844	350.95	19133	9.75
c104_100_17	42948	28.64			39114	23.83
c104_100_18	56769	45.05			53650	39.11
c104_100_19	69921	65.26			66165	56.43
c104_100_20	96971	125.26			91825	110.71
c104_100_21					197498	336.64
c104_100_22					315475	702.11
c104_100_23					437113	1169.71
c104_100_24					547902	1945.73
c104_100_25						

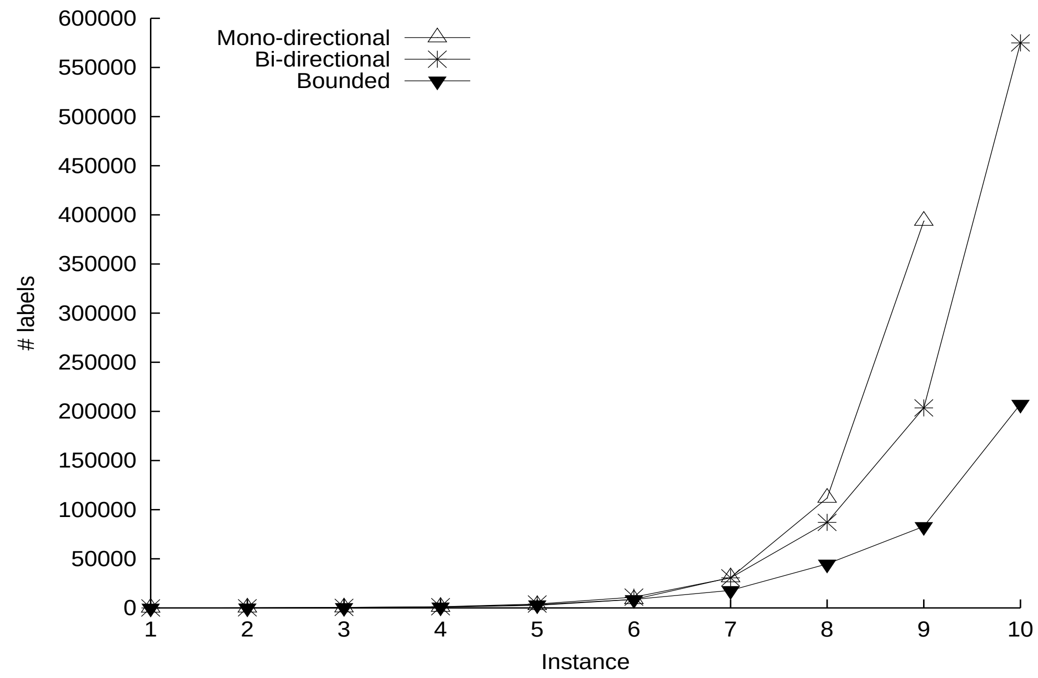

Figure 4.3: Number of states for increasing capacity

Tables, 4.7 and 4.8, show that the difficulty of a RCESPP instance does not depend only on its size but it is strongly affected by the tightness of the constraints. When time windows become larger and larger, the number of non-dominated states increases dramatically and irregularly. The irregular growth in number of states and computing time is due to the local nature of the time windows constraints. Also in these experiments the superiority of bounded bi-directional dynamic programming is outstanding and it is worth remarking that bi-directional search without bounding is inferior to the classical mono-directional algorithm.

Figures 4.3 and 4.4 show similar results: they plot the number of states and the computing time for instances of the RCESPP with capacities, for increasing values of the capacity of the vehicle. All three algorithms show an exponential behaviour when the resource availability grows. Here the growth is more regular because the capacity constraint is a global one.

Tables 4.9 to 4.16 report on the experimental comparison between the elementary bi-directional algorithm with bounds, the state space relaxation algorithm coupled with branch-and-bound and the decremental state space algorithm. For the elementary bi-directional algorithm with bounds, named *Elementary D.P.* in the tables, I report the total number of non-dominated labels and the computing time; for the state space relaxation algorithm with branch-and-bound I report the total number of nodes of the search tree, the computing time and the percentage gap between the upper and the lower bounds for the case of hybrid branching based

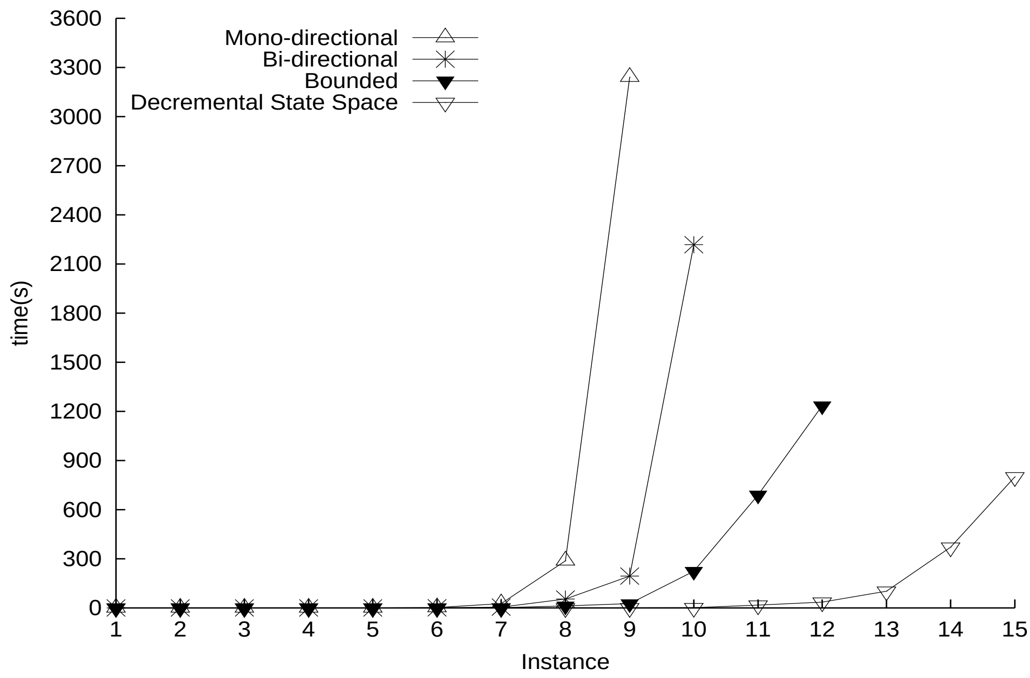

Figure 4.4: Computing time for increasing capacity

on resources+arcs and for the case based on resources+cycles; for the decremental state space algorithm, named $SSR + C.V.$, I report the number of iterations (that is the number of times that the state space algorithm has been invoked), the number of critical nodes for the last iteration and the computing time. Empty cells mean that the solution has not been computed within the time limit of one hour.

Capacities. Results reported in Tables 4.9 and 4.10 show that for 50 vertices instances the decremental state space algorithm clearly outperforms all other algorithms for all classes of instances except for the rc-class; however it should be pointed out that for this class the computing times are all below 1 second. The B&B algorithm (with both branching rules) sometimes dominates the exact D.P. but often it is not able to converge within reasonable computing time. For 100 vertices instances, where the exponential behaviour of the exact D.P. become evident, the decremental state space relaxation algorithm reduces the computing time by 2 orders of magnitude in some cases. The B&B algorithm can be compared with the exact D.P. and the branching strategy with cycles seems to be preferable. However B&B fails to converge within one hour for 6 instances.

Distribution and collection. When solving the RCESPP with distribution and collection I obtained results similar to those above: the results are reported in Tables 4.11 and 4.12. The decremental state space relaxation algorithm solved all instances in less than 340 seconds outperforming the other algorithms and reducing the computing time by 2 orders of magnitude. The B&B is useful only for 100 vertices instances where the performances are somewhat better for the resource+cycles branching.

Capacities and time windows. All Solomon's instances with 50 and 100 nodes were solved by the decremental state space relaxation algorithm; the most difficult instance, c_104, has been solved within 350 seconds. For the other original Solomon's instances the state space based algorithms are not very competitive, because of the tightness and the displacement of the time windows.

Tightness of the constraints. The last two tables, 4.15 and 4.16, show that the difficulty of a RCESPP instance does not depend only on its size but it is strongly affected by the tightness of the constraints. When time windows become larger and larger, the number of non-dominated states and the computing time increase. The growth in number of states and computing time is due to the local nature of the time windows constraints. In these experiments the superiority of state space relaxation based algorithms is evident. Both the B&B and the decremental state space relaxation solved all instances within 50 seconds where the exact D.P. failed. DSSR slightly dominates the B&B. It should be pointed out that the actual implementation of the state space relaxation algorithms does not consider reoptimization as proposed by Desrochers and Soumis [47]. Future work should be done in this direction.

4.10 Conclusions

In this chapter I proposed some improving techniques for dynamic programming algorithms based on bi-directional search, state space relaxation based on branch and bound and decremental state space relaxation, for the exact optimization of resource constrained shortest path problem derived from the set covering reformulation of three vehicle routing problems: CVRP, VRPDC, CVRPTW. For the pricing problem I have shown how bounded bi-directional dynamic programming can be applied to the RCESPP with different resource constraints, namely the RCESPP with one or more resource constraints, interacting or independent resources, local or global constraints, and resource consumptions depending on visited vertices or traversed arcs. The experiments show that bounded bi-directional dynamic programming definitely outperforms the mono-directional algorithm, commonly used and reported in the literature. The experimental comparison on the RCESPP of the exact methods with those based on state space relaxation is favourable to decremental state space relaxation. In the next chapters I analyze their performances when embedded in a Branch-and-Price algorithm.

Table 4.9: RCESPP with capacity - 50 vertices

Instance	Exact D.P.		Resources + Arcs			Resource + Cycles			DSSR		
Name	Labels	Time	Nodes	Time	Gap	Nodes	Time	Gap	Iter	C.N.	Time
c_50_01	56	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
c_50_02	268	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
c_50_03	692	0.00	7	0.01	0.0	1	0.00	0.02	1	0	0.00
c_50_04	2574	0.03	19	0.08	0.0	10	0.05	0.0	3	2	0.02
c_50_05	4692	0.07	11	0.05	0.0	22	0.09	0.0	2	2	0.01
c_50_06	15236	0.91	87	0.48	0.0	116	0.91	0.0	3	3	0.04
c_50_07	23394	1.75	35	0.24	0.0	52	0.33	0.0	2	3	0.02
c_50_08	75026	20.35	315	3.19	0.0	124	1.81	0.0	5	7	0.25
c_50_09	101128	33.24	673	5.80	0.0	224	1.78	0.0	4	8	0.18
c_50_10	331402	394.97	3919	44.56	0.0	3039	53.34	0.0	5	10	1.82
c_50_11	430032	615.10	140517		1.4	38508	604.12	0.0	6	14	17.58
r_50_01	62	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
r_50_02	210	0.01	1	0.01	0.0	1	0.00	0.0	1	0	0.00
r_50_03	525	0.01	7	0.04	0.0	1	0.00	0.0	3	3	0.02
r_50_04	1250	0.02	7	0.06	0.0	1	0.00	0.0	2	3	0.02
r_50_05	2418	0.05	591	6.64	0.0	7	0.09	0.0	6	7	0.17
r_50_06	4570	0.11	271	4.49	0.0	83	1.18	0.0	4	6	0.12
r_50_07	7874	0.24	6009	90.34	0.0	122	1.12	0.0	2	4	0.06
r_50_08	13590	0.60	3149	38.75	0.0	95	0.98	0.0	5	11	0.48
r_50_09	22800	1.49	191	6.12	0.0	17	0.53	0.0	4	9	0.40
r_50_10	36838	3.87	59	1.16	0.0	7	0.26	0.0	3	8	0.34
r_50_11	59911	10.15	4991	207.70	0.0	128	2.64	0.0	4	8	0.65
rc_50_01	44	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
rc_50_02	124	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
rc_50_03	268	0.00	11	0.02	0.0	1	0.01	0.0	4	3	0.00
rc_50_04	560	0.01	725	1.58	0.0	73	0.17	0.0	5	4	0.02
rc_50_05	800	0.01	1573	4.21	0.0	173	0.42	0.0	4	4	0.02
rc_50_06	1551	0.03	73573	562.40	0.0	1774	8.29	0.0	7	8	0.09
rc_50_07	1774	0.04	3417	19.66	0.0	84	0.50	0.0	4	7	0.05
rc_50_08	3217	0.08	239467		6.25	10505	49.19	0.0	7	11	0.20
rc_50_09	3322	0.09	29021	348.97	0.0	960	7.68	0.0	6	11	0.20
rc_50_10	5864	0.19	254931		2.0	16679	156.19	0.0	6	11	0.27
rc_50_11	6050	0.22	168869	3194.50	0.0	15042	210.29	0.0	6	13	0.49

Table 4.10: RCESPP with capacity - 100 vertices

Instance	Exact D.P.		Resources + Arcs			Resource + Cycles			DSSR		
Name	Labels	Time	Nodes	Time	Gap	Nodes	Time	Gap	Iter	C.N.	Time
c_100_01	106	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
c_100_02	559	0.01	1	0.01	0.0	1	0.01	0.0	1	0	0.00
c_100_03	1456	0.03	7	0.04	0.0	1	0.02	0.0	2	1	0.03
c_100_04	5900	0.19	19	0.26	0.0	10	0.15	0.0	3	2	0.06
c_100_05	11546	0.44	43	0.36	0.0	22	0.31	0.0	3	4	0.07
c_100_06	45138	6.60	193	5.19	0.0	699	16.31	0.0	4	5	0.21
c_100_07	75698	13.78	65	1.84	0.0	99	2.65	0.0	3	6	0.18
c_100_08	310651	276.88	1275	77.50	0.0	517	30.53	0.0	6	10	1.34
c_100_09	4799333	520.82	8125	326.11	0.0	862	32.71	0.0	6	14	2.02
c_100_10			11465	627.08	0.0	12076	910.73	0.0	6	13	7.68
r_100_01	266	0.00	15	0.04	0.0	10	0.04	0.0	3	3	0.02
r_100_02	2120	0.06	459	3.61	0.0	225	1.45	0.0	2	4	0.06
r_100_03	11866	0.70	5337	126.87	0.0	776	15.03	0.0	4	5	0.59
r_100_04	53668	8.37	7639	306.49	0.0	1819	69.68	0.0	4	7	2.80
r_100_05	215976	104.41	81677		7.3	4464	198.93	0.0	3	5	2.87
r_100_06	764476	1300.33	51755		9.5	13209	1282.61	0.0	4	8	34.64
r_100_07			31121		11.2	29182		1.1	5	10	143.63
r_100_08			12548		25.4	18741		6.3	5	11	281.62
r_100_09			6912		51.1	12549		18.9	3	10	303.34
r_100_10			2551		67.4	6118		43.2	3	10	319.68
rc_100_01	90	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
rc_100_02	636	0.01	1	0.01	0.0	1	0.00	0.0	1	0	0.00
rc_100_03	1732	0.04	31	0.33	0.0	10	0.13	0.0	1	0	0.00
rc_100_04	5706	0.23	7	0.20	0.0	10	0.27	0.0	2	1	0.07
rc_100_05	12561	0.69	1669	71.45	0.0	64	2.35	0.0	4	4	0.29
rc_100_06	29786	2.80	71	2.52	0.0	60	3.01	0.0	3	4	0.35
rc_100_07	60499	9.74	4403	376.85	0.0	752	59.22	0.0	4	5	0.92
rc_100_08	124752	37.46	5735	353.00	0.0	254	25.00	0.0	4	7	1.77
rc_100_09	237652	130.20	739	109.08	0.0	391	28.76	0.0	3	5	1.40
rc_100_10	459269	470.24	25055		3.7	7385	1191.96	0.0	5	10	7.33

Table 4.11: RCESPP with distribution and collection - 50 vertices

Instance	Elementary D.P.		Resources + Arcs			Resource + Cycles			SSR + C.N.		
Name	Labels	Time	Nodes	Time	Gap	Nodes	Time	Gap	Iter	C.N.	Time
c_50_01	26	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
c_50_02	159	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
c_50_03	554	0.01	1	0.02	0.0	1	0.01	0.0	2	1	0.00
c_50_04	1751	0.02	7	0.02	0.0	10	0.04	0.0	2	1	0.01
c_50_05	4675	0.07	15	0.07	0.0	22	0.10	0.0	2	2	0.02
c_50_06	11311	0.41	27	0.17	0.0	122	0.94	0.0	3	3	0.05
c_50_07	24006	1.72	17	0.16	0.0	53	0.48	0.0	2	3	0.03
c_50_08	51401	7.95	83	0.90	0.0	91	1.44	0.0	5	7	0.47
c_50_09	110354	35.46	69	0.88	0.0	207	2.47	0.0	3	5	0.19
c_50_10	233478	165.21	119	3.56	0.0	129	3.66	0.0	5	10	2.20
c_50_11	474147	672.29	21697	909.08	0.0	13404	351.33	0.0	6	14	45.97
r_50_01	59	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
r_50_02	188	0.00	5	0.02	0.0	7	0.02	0.0	2	1	0.01
r_50_03	486	0.01	1	0.01	0.0	1	0.01	0.0	3	3	0.01
r_50_04	1113	0.02	1	0.03	0.0	1	0.03	0.0	2	3	0.02
r_50_05	2085	0.04	11	0.14	0.0	7	0.08	0.0	5	6	0.13
r_50_06	3882	0.10	17	0.28	0.0	45	0.62	0.0	3	5	0.07
r_50_07	6986	0.23	3	0.07	0.0	3	0.08	0.0	2	4	0.06
r_50_08	12138	0.51	71	0.97	0.0	122	1.13	0.0	4	7	0.19
r_50_09	20384	1.23	13	0.42	0.0	19	0.59	0.0	3	7	0.21
r_50_10	33107	3.15	5	0.21	0.0	5	0.21	0.0	3	7	0.25
r_50_11	55835	8.19	49	2.21	0.0	61	2.39	0.0	4	8	0.73
rc_50_01	24	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
rc_50_02	83	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.01
rc_50_03	199	0.01	1	0.01	0.0	1	0.01	0.0	3	2	0.01
rc_50_04	397	0.01	55	0.12	0.0	73	0.14	0.0	5	4	0.02
rc_50_05	764	0.02	171	0.51	0.0	114	0.28	0.0	4	5	0.03
rc_50_06	1108	0.02	3321	19.86	0.0	1156	5.01	0.0	6	7	0.09
rc_50_07	1817	0.03	397	3.00	0.0	233	1.40	0.0	6	7	0.09
rc_50_08	2546	0.05	595	6.41	0.0	405	4.07	0.0	4	7	0.05
rc_50_09	3435	0.10	4363	63.94	0.0	666	8.13	0.0	7	9	0.22
rc_50_10	4998	0.15	12045	210.69	0.0	6551	79.95	0.0	6	11	0.24
rc_50_11	6213	0.23	166346		1.5	18791		1.1	6	11	0.29

Table 4.12: RCESPP with distribution and collection - 100 vertices

Instance	Elementary D.P.		Resources + Arcs			Resource + Cycles			SSR + C.N.		
Name	Labels	Time	Nodes	Time	Gap	Nodes	Time	Gap	Iter	C.N.	Time
c_100_01	48	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
c_100_02	328	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.02
c_100_03	1179	0.02	1	0.00	0.0	1	0.00	0.0	2	1	0.02
c_100_04	3829	0.08	7	0.10	0.0	10	0.13	0.0	3	2	0.06
c_100_05	11112	0.41	15	0.26	0.0	22	0.40	0.0	3	4	0.10
c_100_06	30823	2.62	15	0.54	0.0	597	17.26	0.0	4	5	0.25
c_100_07	76548	12.72	35	1.24	0.0	101	3.67	0.0	3	6	0.27
c_100_08	197386	87.88	175	6.56	0.0	192	12.57	0.0	6	10	2.17
c_100_09	509042	516.42	239	15.83	0.0	884	58.01	0.0	5	11	2.34
c_100_10			737	89.37	0.0	952	11.27	0.0	7	15	20.64
r_100_01	253	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
r_100_02	1948	0.06	801	7.47	0.0	222	1.52	0.0	2	4	0.06
r_100_03	10874	0.68	7275	186.84	0.0	831	16.90	0.0	4	5	0.64
r_100_04	49258	7.34	15297	790.61	0.0	1814	69.59	0.0	3	5	1.34
r_100_05	189041	84.00	40991	3503.40	0.0	5242	357.35	0.0	3	5	3.71
r_100_06	676338	1040.25	42932		5.0	14627	1074.52	0.0	4	8	39.63
r_100_07			36124		8.9	24782		1.4	5	10	180.41
r_100_08			19733		22.3	30764		12.5	4	10	217.66
r_100_09			8153		45.5	11489		22.0	3	10	337.47
r_100_10			3559		62.8	4025		61.0	3	10	337.43
rc_100_01	67	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
rc_100_02	501	0.01	1	0.00	0.0	1	0.00	0.0	1	0	0.00
rc_100_03	1422	0.04	7	0.10	0.0	10	0.13	0.0	2	1	0.03
rc_100_04	4540	0.17	7	0.19	0.0	10	0.26	0.0	2	1	0.07
rc_100_05	10790	0.55	27	1.16	0.0	61	2.25	0.0	4	4	0.30
rc_100_06	25657	2.29	23	1.18	0.0	54	2.98	0.0	3	4	0.33
rc_100_07	52378	7.73	801	74.78	0.0	736	58.96	0.0	4	5	0.97
rc_100_08	107414	28.62	361	44.60	0.0	242	24.13	0.0	3	5	1.11
rc_100_09	207049	99.56	235	23.30	0.0	389	29.55	0.0	3	5	1.55
rc_100_10			5671	971.28	0.0	7162	1207.45	0.0	4	8	6.11

Table 4.13: RCESPP with capacity and time windows - 50 vertices

Instance	Exact D.P.		Resources + Arcs			Resource + Cycles			DSSR		
Name	Labels	Time	Nodes	Time	Gap	Nodes	Time	Gap	Iter	C.N.	Time
c101_50	323	0.01	1	0.00	0.0	1	0.00	0.0	1	0	0.00
c102_50	3026	0.14	15	0.19	0.0	12	0.19	0.0	2	2	0.12
c103_50	30736	10.25	2533	127.32	0.0	966	27.74	0.0	4	6	14.06
c104_50			35781		1.2	24785	1835.47	0.0	4	11	344.65
c105_50	435	0.02	1	0.02	0.0	1	0.02	0.0	1	0	0.01
c106_50	359	0.01	1	0.03	0.0	1	0.03	0.0	1	0	0.02
c107_50	504	0.02	1	0.04	0.0	1	0.04	0.0	1	0	0.02
c108_50	736	0.04	1	0.04	0.0	1	0.04	0.0	1	0	0.04
c109_50	2031	0.15	81	1.90	0.0	78	1.69	0.0	8	11	1.35
r101_50	121	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.00
r102_50	596	0.02	5	0.05	0.0	4	0.04	0.0	4	5	0.08
r103_50	2322	0.06	817	9.83	0.0	103	1.07	0.0	3	4	0.27
r104_50	15441	0.66	83	2.46	0.0	264	7.08	0.0	3	5	0.77
r105_50	238	0.01	1	0.01	0.0	1	0.01	0.0	1	0	0.0
r106_50	818	0.02	13	0.14	0.0	9	0.08	0.0	4	6	0.18
r107_50	2784	0.08	15435	248.92	0.0	268	3.10	0.0	4	5	0.47
r108_50	16457	0.75	5	0.15	0.0	5	0.15	0.0	2	4	0.48
r109_50	584	0.02	1	0.02	0.0	1	0.02	0.0	1	0	0.02
r110_50	1600	0.04	1	0.04	0.0	1	0.04	0.0	1	0	0.04
r111_50	2289	0.07	43	0.73	0.0	46	0.66	0.0	3	5	0.33
r112_50	3987	0.12	1	0.05	0.0	1	0.05	0.0	1	0	0.05
rc101_50	270	0.00	1	0.00	0.0	1	0.00	0.0	1	0	0.01
rc102_50	906	0.01	17	0.13	0.0	17	0.13	0.0	3	3	0.09
rc103_50	3509	0.07	112	3.45	0.0	6168	90.88	0.0	5	11	0.95
rc104_50	8801	0.28	258	7.65	0.0	312	8.31	0.0	7	15	4.77
rc105_50	937	0.01	63	0.48	0.0	60	0.32	0.0	3	4	0.11
rc106_50	889	0.01	71	0.52	0.0	676	4.64	0.0	4	7	0.16
rc107_50	3525	0.07	29567	653.53	0.0	13452	195.907	0.0	6	9	0.83
rc108_50	10166	0.21	34205	1143.65	0.0	58476	1474.19	0.0	6	13	2.12

Table 4.14: RCESPP with capacity and time windows - 100 vertices

Instance	Exact D.P.		Resources + Arcs			Resource + Cycles			DSSR		
Name	Labels	Time	Nodes	Time	Gap	Nodes	Time	Gap	Iter	C.N.	Time
c101_100	679	0.06	1	0.02	0.0	1	0.02	0.0	1	0	0.02
c102_100	11839	2.99	15	0.89	0.0	12	0.67	0.0	2	2	0.63
c103_100	123804	133.56	2539	507.78	0.0	3075	485.68	0.0	5	9	40.15
c104_100			17553		17.8	23402		16.5	4	11	311.44
c105_100	915	0.11	1	0.06	0.0	1	0.06	0.0	1	0	0.06
c106_100	1159	0.16	1	0.07	0.0	1	0.07	0.0	1	0	0.07
c107_100	1058	0.16	1	0.08	0.0	1	0.08	0.0	1	0	0.08
c108_100	1690	0.33	1	0.17	0.0	1	0.17	0.0	1	0	0.17
c109_100	4608	1.28	111	14.18	0.0	101	12.66	0.0	8	13	9.19
r101_100	452	0.01	1	0.00	0.0	1	0.00	0.0	1	0	0.00
r102_100	14792	2.21	4203	438.82	0.0	1003	96.25	0.0	3	6	21.69
r103_100	135575	95.73	377	128.71	0.0	252	61.10	0.0	4	7	159.74
r104_100	655858	1242.56	4531		0.9	1945	1013.04	0.0	3	5	78.32
r105_100	1161	0.06	1	0.03	0.0	1	0.03	0.0	1	0	0.03
r106_100	22970	5.52	26059		3.4	3344	467.29	0.0	4	7	71.60
r107_100	138027	100.83	1417	717.37	0.0	732	232.91	0.0	4	12	335.98
r108_100	570910	891.81	1593	1098.05	0.0	1451	611.61	0.0	3	5	146.58
r109_100	3504	0.37	49	3.39	0.0	49	3.45	0.0	3	5	2.93
r110_100	25063	4.89	307	69.25	0.0	406	77.55	0.0	3	6	19.31
r111_100	69890	25.86	2669	866.34	0.0	361	129.24	0.0	3	7	53.16
r112_100	394702	647.36	2167	2227.74	0.0	665	385.00	0.0	3	9	340.61
rc101_100	955	0.03	1	0.02	0.0	1	0.02	0.0	1	0	0.02
rc102_100	5384	0.30	17	1.12	0.0	21	1.34	0.0	4	4	3.17
rc103_100	38308	6.01	861	110.78	0.0	6494	587.39	0.0	4	8	35.37
rc104_100	232961	148.73	607	194.68	0.0	1054	203.36	0.0	4	8	102.56
rc105_100	2964	0.15	7	0.43	0.0	13	0.67	0.0	4	6	1.14
rc106_100	2574	0.12	31	1.85	0.0	12	0.56	0.0	3	3	1.01
rc107_100	10505	0.72	87	8.50	0.0	27	2.79	0.0	4	6	3.65
rc108_100	45430	6.75	239	32.34	0.0	143	12.43	0.0	3	5	4.84

Table 4.15: RCESPP with capacity and time windows - c_104, 50 vertices

Instance	Exact D.P.		Resources + Arcs			Resource + Cycles			DSSR		
Name	Labels	Time	Nodes	Time	Gap	Nodes	Time	Gap	Iter	C.N.	Time
c104_50_01	59	0.00	1	0.01	0.0	1	0.01	0.0	1	0	0.01
c104_50_02	126	0.00	1	0.01	0.0	1	0.01	0.0	1	0	0.01
c104_50_03	200	0.01	1	0.01	0.0	1	0.01	0.0	1	0	0.01
c104_50_04	206	0.01	1	0.01	0.0	1	0.01	0.0	1	0	0.01
c104_50_05	208	0.01	1	0.01	0.0	1	0.01	0.0	1	0	0.01
c104_50_06	269	0.01	1	0.02	0.0	1	0.02	0.0	1	0	0.02
c104_50_07	475	0.01	1	0.02	0.0	1	0.02	0.0	1	0	0.02
c104_50_08	586	0.04	1	0.02	0.0	1	0.02	0.0	1	0	0.02
c104_50_09	730	0.05	1	0.02	0.0	1	0.02	0.0	1	0	0.02
c104_50_10	804	0.06	1	0.02	0.0	1	0.02	0.0	1	0	0.02
c104_50_11	1440	0.09	1	0.03	0.0	1	0.03	0.0	1	0	0.03
c104_50_12	2292	0.21	1	0.03	0.0	1	0.03	0.0	1	0	0.03
c104_50_13	3771	0.43	1	0.04	0.0	1	0.04	0.0	1	0	0.04
c104_50_14	4031	0.53	19	0.24	0.0	13	0.18	0.0	2	3	0.14
c104_50_15	5508	0.74	13	0.22	0.0	9	0.12	0.0	3	4	0.49
c104_50_16	9411	2.10	39	0.88	0.0	12	0.25	0.0	:3	4	0.53
c104_50_17	18579	5.02	45	1.09	0.0	15	0.35	0.0	3	4	0.60
c104_50_18	21738	6.54	145	3.53	0.0	76	1.66	0.0	3	4	0.61
c104_50_19	25638	9.25	77	2.43	0.0	64	1.41	0.0	3	5	1.20
c104_50_20	36762	23.16	85	2.02	0.0	91	2.35	0.0	4	6	2.11
c104_50_21	81804	73.19	179	5.95	0.0	144	3.99	0.0	4	6	2.69
c104_50_22	105756	110.25	199	5.35	0.0	91	2.52	0.0	3	6	2.31
c104_50_23	126645	149.81	255	11.00	0.0	214	9.34	0.0	8	11	17.45
c104_50_24	157915	275.93	276	19.7	0.0	238	17.98	0.0	5	7	13.43
c104_50_25	323641	1016.98	321	18.36	0.0	381	19.83	0.0	4	7	13.31

Table 4.16: RCESPP with capacity and time windows - c_104, 100 vertices

Instance	Exact		Resources + Arcs			Resource + Cycles			DSSR		
Name	Labels	Time	Nodes	Time	Gap	Nodes	Time	Gap	Iter	C.N.	Time
c104_100_01	160	0.01	1	0.04	0.0	1	0.04	0.0	1	0	0.04
c104_100_02	227	0.01	1	0.05	0.0	1	0.05	0.0	1	0	0.05
c104_100_03	368	0.02	1	0.06	0.0	1	0.06	0.0	1	0	0.06
c104_100_04	415	0.04	1	0.07	0.0	1	0.07	0.0	1	0	0.07
c104_100_05	427	0.05	1	0.07	0.0	1	0.07	0.0	1	0	0.07
c104_100_06	523	0.06	1	0.08	0.0	1	0.08	0.0	1	0	0.08
c104_100_07	895	0.11	1	0.08	0.0	1	0.08	0.0	1	0	0.08
c104_100_08	1153	0.19	1	0.09	0.0	1	0.09	0.0	1	0	0.09
c104_100_09	1393	0.29	1	0.09	0.0	1	0.09	0.0	1	0	0.09
c104_100_10	1557	0.37	1	0.09	0.0	1	0.09	0.0	1	0	0.09
c104_100_11	2655	0.59	1	0.11	0.0	1	0.11	0.0	1	0	0.11
c104_100_12	4504	1.24	1	0.12	0.0	1	0.12	0.0	1	0	0.12
c104_100_13	7336	2.37	1	0.11	0.0	1	0.11	0.0	1	0	0.11
c104_100_14	8457	3.30	33	1.76	0.0	11	0.74	0.0	3	3	1.49
c104_100_15	10932	4.49	65	3.24	0.0	21	1.05	0.0	2	3	1.26
c104_100_16	19133	9.75	249	15.12	0.0	23	1.62	0.0	2	3	1.56
c104_100_17	39114	23.83	211	20.02	0.0	21	1.60	0.0	2	3	1.65
c104_100_18	53650	39.11	75	8.64	0.0	30	3.54	0.0	2	3	1.65
c104_100_19	66165	56.43	45	5.85	0.0	30	2.99	0.0	4	6	6.32
c104_100_20	91825	110.71	59	4.66	0.0	42	2.04	0.0	4	5	7.84
c104_100_21	197498	336.64	67	7.29	0.0	48	4.26	0.0	4	5	8.79
c104_100_22	315475	702.11	89	10.78	0.0	59	5.94	0.0	3	6	7.54
c104_100_23	437113	1169.71	53	7.94	0.0	53	7.60	0.0	7	9	32.06
c104_100_24	547902	1945.73	141	31.02	0.0	143	33.46	0.0	5	7	29.87
c104_100_25			187	47.05	0.0	17	44.32	0.0	4	7	27.74

Chapter 5

A Branch-and-Price algorithm

In this chapter I provide the implementation details of a Branch-and-Price algorithm for vehicle routing problems. Specific problem-related considerations will be provided in the next three chapters.

5.1 Implementing a Branch and Price algorithm

The theoretical aspects of column generation are simple, clearly understandable and powerful. Nevertheless the general scheme provided in section 3.2 must be enriched by a lot of hidden issues when developing a branch-and-price algorithm.

5.1.1 Initialization

All column generation methods need an initial feasible restricted linear master problem to ensure that proper dual information is passed to the pricing problem and it has been shown that proper initialization is an important issue to solve a linear program by column generation. (see Vanderbeck [17]).

For CVRP and VRPDC I used the tabu search algorithm proposed by Bianchessi and Righini [65] to initialize the restricted master problem. The initialization algorithm provided a set of feasible columns that vary from 20 to 50 in the reported experiments.

For the CVRPTW the restricted master problem was initialized with a set of feasible columns obtained with a modified Clarke&Wright's savings algorithm [20]. The modification deals with the time windows feasibility that is checked at each iteration before the joining procedure. Five runs of the initialization provided a set of feasible columns that vary from 10 to 30 in the reported experiments. For each run the travel costs have been modified by increasing the cost of the arcs belonging to the routes obtained from the previous run by their double.

To ensure the existence of a feasible solution I insert a dummy column with a very high cost, representing a fictitious (and infeasible) route visiting all the customers. For each specific VRP I provide a problem specific initialization heuristic (see next three chapters).

At each non-root node of the branch-and-bound tree, the algorithm initializes the $RLMP$ with the same set of columns used by the last considered node, excepted the columns which have become infeasible because of branching. Note that the last node considered does not necessarily coincide with the father of the current node, but it depends on the search strategy.

5.1.2 Column pool and column management

In section 3.3 I proposed a set partitioning reformulation for the CVRP. In that model the constant a_{ip} represents the covering of user i by route p i.e. it takes value 1 if the column p visits node i and 0 otherwise. Then the column itself does not contain explicit information on the sequence in which the customers are visited.

It is then necessary an appropriate data structure, the column pool, where each generated column is associated with the list of the arcs used in the corresponding route. This additional data-structure is necessary to test whether the column is feasible in each node of the branching tree.

The implementation used in this thesis is a double-linked list. Associated with the list, an array of pointers ensures direct access to the structure of the columns stored in the RLMP. Two implementations based on trees with logarithmic insertion and deletion time and based on hash functions with constant insertion and deletion time has been tested without any significant improvement because the column pool management time is a very small fraction (less than 0.2% in the tests) of the overall computing time.

All columns with negative reduced cost generated at each iteration of the column generation algorithm are inserted into the $RLMP$. When the number of columns in the $RLMP$ exceeds a predefined fixed value (set to 5000 in the tests), I delete the columns with a reduced cost greater than a threshold value (set to $N\ \min_{(i,j)\in\mathcal{A}}\{c_{ij}\}$ in the tests) but not from the pool. After ten consecutive evaluations yielding a positive reduced cost, the columns are erased also from the pool.

5.1.3 Heuristic pricing

At each iteration of the column generation process it is not necessary to solve the pricing problem to optimality to find columns with negative reduced cost. It is possible to devise ad-hoc heuristic algorithms to perform a first search. Next, if heuristics fail, an optimization algorithm is used to find more columns or to prove the RLMP optimality.

A heuristic algorithm based on dynamic programming is obtained by using the state definition of RCESPP and using the dominance criterion of the RCSPP. Another heuristic is obtained discarding all arcs $(i, j) \in \mathcal{A}$ such that:

$$c_{ij} > \alpha \max_{k \in \mathcal{N}} \{\lambda_k\}$$

where parameter α is set between 0 and 1. When the dynamic programming algorithm described above is run on the reduced graph, it is very fast and though it may miss the optimal solution it can find negative reduced cost columns.

In mine experiments I set $\alpha = 0.1$ and observed that the first time the algorithm is executed at each node of the branching tree it finds a lot of negative reduced cost columns, including the optimal one, in a small fraction of the computing time required by the same algorithm on the complete graph.

However if the value of α is left unchanged, the algorithm fails to find negative reduced cost columns in subsequent runs; therefore we raise α by 0.2 in every subsequent trial. If no column with negative reduced cost is found with $\alpha = 0.9$, the heuristic algorithm stops.

5.1.4 Overall pricing algorithm

In column generation algorithms it is common to execute some fast heuristics to generate columns with negative reduced cost quickly, whenever possible. In our algorithm at each column generation iteration the search for new columns is made by successive steps of increasing computational complexity. The search is stopped as soon as M columns with negative reduced cost have been found.

Step 1: Search in the pool. A pool of previously computed columns is scanned and the reduced cost is evaluated for all columns which are feasible for the current node.

Step 2: Deterministic greedy algorithm. A nearest neighbor algorithm produces a solution by adding arcs to a path starting from vertex s and going forward or starting from vertex t and going backward. At each iteration the most valuable arc is added among those which do not close cycles. The algorithm is executed once forward and once backward.

Step 3: Randomized greedy algorithm. This step is similar to the previous one but at every iteration the next arc is randomly chosen among the four most valuable arcs extending the path. The algorithm is multistarted forward and backward for a number of times depending on the size of the problem instance.

Step 4: Heuristic dynamic programming. The heuristic (bi-directional) dynamic programming algorithm described above is executed on the complete graph reduced .

Step 5: Reduced dynamic programming. The exact (bi-directional) dynamic programming algorithm is executed on the graph reduced as explained above, with values of α increasing from 0.1 to 0.9 until some new column is found.

Step 6: Dynamic programming. This last step consists of running the exact algorithm or the decremental state space relaxation algorithm. This step is executed only if no column with negative reduced cost has been generated in steps 1-5.

5.1.5 Stabilization

It has been shown that the column generation methods based on the simplex algorithm have poor convergence (see Lübbecke and Desrosiers [15] and Amor [35]). Moreover it has been observed that the dual variable values do not monotonically converge to their optima but they oscillate. This fluctuation is connected to the coordination between the master problem and the subproblem. For combinatorial problems the optimal solution is an extreme point of the optimal face of the dual polyhedron. Then a new column for the master problem (a new row in the dual problem, that is a hyper-plane in the dual space) can vary this optimal face with small changes of the objective value but big changes of the dual variable values.

Several ideas to stabilize the dual space have been proposed in the literature.

Trust Region method
The basic idea is to control the dual variables considering the dual RMP where additional box constraints are imposed around the current optimal solution of the dual variables λ. That is $\hat{\lambda}_i - \delta \leq \lambda_i \leq \hat{\lambda}_i + \delta$ with $\delta > 0$. The parameter δ is tuned dynamically during the column generation process. The method has been proposed by Marsten [80] and used by Kallehague et al. [5]

Distance penalization
The main idea is to linearly penalize the distance of the new dual variable from the ones of the previous iteration the penalization parameter is tuned along the generation process, see Kim et al. [85].

Primal-Dual stabilization
A more flexible stabilized column generation approach, that can be seen as a mixed approach of the two above, has been followed by Ben Amor [35] and du Merle et

al. [12]. They used a linear programming box on the dual variables added with a ε-perturbation of the right hand sides. This means that a deviation of λ from the current dual optimal value outside the stabilization box is penalized by a factor ϵ per unit. The parameters are tuned during the column generation process. If the new dual variables are inside the specified box then it is reduced and the penalty is augmented; otherwise the box is enlarged and ε is decreased to allow new dual variables to be generated since the current estimation is bad.

Interior point stabilization

The two methods described impose a great attention on the parameter tuning during the whole generation process. A potentially parameter-free stabilization method is the one proposed by Rousseau et al. [45]. Since the set covering formulation of the VRP is degenerate the dual problem has an infinite number of optimal solutions. The main idea of the interior point stabilization is not to use one of the extreme points of the dual polyhedron, returned randomly by a standard LP solver, as an estimate of the marginal costs but rather the center of the optimal face or at least an *interior point*.

Let us consider the dual of the master problem:

$$
\begin{aligned}
d^* = \max & \sum_{i \in \{1 \dots N\}} b\lambda_i & \\
\text{s.t.} & \sum_{i \in \{1 \dots N\}} \lambda_i a_{ip} \leq c_p & \forall p \in P \\
& \lambda \geq 0 &
\end{aligned}
$$

Once the RLMP is solved to optimality let P^* be the set of columns for which $z_p > 0$ and S the set of nodes for wich the covering constraint is not tight, that is the associated slack variable is strictly positive. The optimal dual polyhedron has the form:

$$
\begin{aligned}
& \sum_{i \in \{1 \dots N\}} \lambda_i a_{ip} \leq c_p & \forall p \in P \setminus P^* \\
& \sum_{i \in \{1 \dots N\}} \lambda_i a_{ip} = c_p & \forall p \in P^* \\
& \lambda_i = 0 & \forall i \in S \\
& \lambda_i \geq 0 & \forall i \in \{1 \dots N\} \setminus S
\end{aligned}
$$

A point of this polyhedron can be obtained solving the following linear problem:

$$
\begin{aligned}
\max \sum_{i \in \{1 \ldots N\}} & u_i \lambda_i & \\
\sum_{i \in \{1 \ldots N\}} & \lambda_i a_{ip} \leq c_p & \forall p \in P \setminus P^* \\
\sum_{i \in \{1 \ldots N\}} & \lambda_i a_{ip} = c_p & \forall p \in P^* \\
\lambda_i & = 0 & \forall i \in S \\
\lambda_i & \geq 0 & \forall i \in \{1 \ldots N\} \setminus S
\end{aligned}
$$

where the objective a convex combination, $\mathbf{u}\lambda$ with $u_i \in (0, 1)$, of the dual variables. Iterating this procedure and perturbing the vector $\mathbf{u}$ (considering also $-\mathbf{u}$) one should obtain several extreme points of the dual polyhedron. Any convex combination of such extreme points gives an interior point of the optimal dual face and thus a best estimate of the marginal costs associated with the constraints of the RLMP.

I used the interior point stabilization technique described above and I found that for VRP problems it works much better than the one based on dual boxes proposed by du Merle et al. [12]. From my experience I found that stabilization is useful to generate an initial set of good columns in the root node of the search tree. In the subsequent nodes of the search tree the feasible columns are sufficient to provide a good estimation of the dual values, even without stabilization.

5.1.6 Lower bounding and termination

It is well-known that one of the drawbacks of column generation is the so-called "tailing-off" effect: a lot of iterations that do not significantly modify the optimal value of the $RLMP$ are often necessary to prove optimality. Therefore many authors (see for instance Farley [3] and Vanderbeck and Wolsey [18]) proposed alternative lower bounding techniques, that allow an earlier termination of column generation yet providing a valid lower bound.

The algorithm exploits the bound obtained from the Lagrangean relaxation of covering constraints (3.16). The optimal value $\tilde{c}^*$ of the pricing subproblem is used to compute the lower bound

$$z_{lb} = K\tilde{c}^* + z^*_{RLMP}$$

where z^*_{RLMP} is the optimal value of the current $RLMP$.

When z_{lb} is greater than the best incumbent feasible solution, the current node is fathomed. If column generation is terminated because the time-out has expired, z_{lb} is kept as the final lower bound of the node.

5.1.7 Search strategy

The column generation algorithm described above is executed at every node of a decision tree in a branch-and-bound framework. I explore the decision tree according to a best-first policy, where subproblems are ranked on the basis of the associated lower bound. Experiments with depth-first search yielded inferior results in terms of number of problems solved to optimality within the same amount of computing time.

5.1.8 Upper bounding

I did not insert in the branch-and-bound algorithm any sophisticated upper bounding technique, because computing a feasible solution to the VRPSPD is an $\mathcal{NP}$-complete problem in itself. To quickly produce other feasible solutions during the search I devised the following greedy procedure, exploiting the structure of the current (possibly fractional) optimum of the $RLMP$ at each column generation iteration. The cycle-free column with the highest fractional value, say r^*, is inserted into the primal solution (i.e. z_{r^*} is temporarily fixed to 1) and the resulting $RLMP$ is reoptimized. This is iterated until either a complete integer solution is obtained or the algorithm fails. This heuristic is quite fast and therefore it is executed at each column generation iteration at each node of the search tree. This is especially useful for large instances, where the initial upper bound is not so close to the optimum as it is for small instances.

5.1.9 Branching

In the RLMP there are only cycle-free columns but there may be still fractional variables, In this case I perform branching on arcs: this binary branching scheme consists of selecting a vertex i^* belonging to different fractional routes, in which the arcs that are fractionally selected entering it or leaving it are different. Half of such arcs are then fixed forbidden in one branch and the other half is “forbidden” in the other branch.

5.2 Increasing the lower bound by valid inequalities

It is common that the solution of the linear relaxation of the RMP is not feasible for the original problem. For this reason the Branch-and-price method require to take decisions on fractional variables to reach feasibility. Another way to reach the feasibility is based on the Branch-and-cut method where the introduction of valid inequalities is used to improve the bounds to tighten the formulation before the branching decisions. There are a lot of general methods to calculate valid inequalities for integer programs but more efficient methods can be devised when the structure of the problem is known. The TSP polytope has been widely studied in the literature (see for instance Jünger et al. [58]) and many inequalities for VRP can be derived from those for TSP (see Cornuejols and Harche [21]).

Kohl et al. [69] introduced the use of valid inequalities in column generation methods for CVRPTW. When column generation, valid inequalities and branching decisions on variables are used together to solve to optimality an integer problem a *Branch&Cut&Price* algorithm arises.

Here I revise briefly the method and how to embed the cut generation into a Branch-and-price framework.

The k-Path cuts A general valid inequality for VRPs has the following form:

$$\sum_{k\in\mathcal{K}}\left(\sum_{(i,j)\in\mathcal{A}}\alpha_{ij}^k X_{ij}^k\right)\geq\gamma$$

where α and γ are constants. When vehicles in $\mathcal{K}$ are identical it is common to aggregate the inequality over the index k. The simpler inequality that arises has the form

$$\sum_{(i,j)\in\mathcal{A}}\alpha_{ij}X_{ij}\geq\gamma$$

Kohl et al. [69] introduced the concept of k-path cuts for CVRPTW extending the idea of Laporte et al. [30]. The basic idea is to identify a subset of vertices that is visited by less than k vehicles in the current fractional solution but it requires, in the optimal solution, at least k vehicles to be serviced.

Let $F=(\mathcal{V},\mathcal{A}')$ be the flow network induced by the current solution x_{ijp} where $\mathcal{A}'=\{(i,j)\in\mathcal{A}|x_{ijp}>0\}$ and the flow capacity is given by $x_{ij}=\sum_{p\in\mathcal{P}}x_{ijp}$

For any given set $S\subseteq\mathcal{V}$, $|S|\geq 1$ the flow $x(S)$ into S is given by:

$$x(S)=\sum_{i\in\bar{S}}\sum_{j\in S}x_{ij}$$

where $\bar{S}=\mathcal{V}\setminus S$. The subtour elimination constraints (see [58]) of the form $X(S)\geq 1$ can be reformulated for the VRPs replacing the right hand side by $k(S)$ that represents the smallest number of vehicles needed to service all vertices in S. The inequality:

$$\sum_{(i,j)\in\mathcal{A}}\alpha_{ij}X_{ij}\geq\gamma$$

is now easily transformed into a k-path inequality by setting $\alpha_{ij} = 1$ if $i \in \bar{S} \wedge j \in S$ and $\gamma = k(S)$

For time constrained VRPs the identification of $k(S)$ requires to solve a time constrained VRP over the set S using exactly k vehicles, for capacity constrained VRPs the identification of $k(S)$ requires to solve a bin-packing problem over S using exactly k bins.

For this reason the authors decided to limit the search for 2-path cut inequalities for CVRPTW (that is checking whether $x(S) < 2$ and $k(S) > 1$). To check $k(S) > 1$ for a particular S one should determine whether the vehicle capacity is not sufficient for the set S (this can be done in linear time) and whether there is not a time-feasible hamiltonian tour through S. When $|S|$ is small the TSPTW feasibility problem can be easily solved by dynamic programming (see Dumas et al. [92]).

Kohl et al. proposed a heuristic and an exact method to find sets S with $x(S) < 2$ while more sophisticated and efficient separation algorithms have been proposed by Cook and Rich [90].

The main idea is to add valid inequalities to the master problem by adding appropriate rows to the RLMP. The added inequalities introduce an extra set of dual variables that must be taken into account. Let $\mu_r \geq 0$ be the dual variable associated with the r-th inequality. The reduced cost of the arcs is changed as follows:

$$\hat{c}_{ij} = c_{ij} - \alpha_{ij}\mu_r$$

The advantage of this kind of inequalities is that they do not change the structure of the pricing subproblem, which is still a RCESPP.

Since the purpose of this thesis is not a deep analysis of cutting plane methods, only the basic implementation of the above algorithms has been taken into account, see Kohl et al. [69] .

5.3 Computational results

The next three chapters report on computational experiments for CVRP, VRPDC and CVRPTW. All computational experiments were performed on a 1.6GHz PC equipped with 512Mb of memory and Linux Red Hat 7.3. Algorithms were coded in C and compiled with gcc 3.0.4. with the -o3 (optimization) option. The CPU times reported are in seconds and include all I/O and memory allocation operations. Times have been computed with the *user_time()* function of the *ptime* library. The restricted master problems were solved with the ILOG-CPLEX 8.1 callable library using the primal simplex algorithm (*primopt*). Computing time was limited to two hours, except for previously unsolved instances where computing time limitations have been removed.

Chapter 6

Computational results: The CVRP

6.1 Instances

Several instance sets have been proposed in the literature to evaluate the performances of algorithms for the CVRP. Most of them and the most widely used are Euclidean instances. This means that the customers are located on a Euclidean plane and that the travel cost c_{ij} from customer i to customer j is equal to the Euclidean distance between the customers. Augerat et al. [70] proposed three sets of Euclidean instances where customers coordinates are drawn randomly within a square made of 100x100 discrete points. Christofides and Elion instance set [68] is made of Euclidean instances where both customers coordinates and delivery demands are randomly generated. Other Euclidean and non Euclidean instance sets have been proposed by Fisher [54], Fischetti et al. [53], Hadjiconstantinou et al [14] and Christofides, Mingozzi and Toth instance set [66]. These sets of instances are available at www.brancandcut.org.

The best available results for the mentioned test instances have been provided recently by Fukasawa et al. [79]. The authors proposed an efficient combination of both column generation and cut generation techniques embedded in a branch-and-bound algorithm. The proposed algorithm has been called robust Branch-and-Cut-and-Price (BCP in the reminder). It should be pointed out that the work of Fukasawa et al. is based on the efficient separation procedures derived from the CVRPSEP package [40] proposed by Lysgaard [41] while the column generation method is a standard q-route generation method with k-cycle elimination (with $k \leq 3$). A q-route is a path that starts at the depot and visits a subset of customers with total demand less or equal to the vehicle capacity. Some customers may be visited more than once and then the set of valid CVRP routes is included in the set of q-routes.

The algorithm proposed in this thesis does not aim to represent the state-of-the-art algorithm for the CVRP. In this work the cut generation component is not explored deeply. Only the 2-path cuts generation (see Kohl et al. [69]) has been embedded in the original set-partitioning formulation.

The main scope of this thesis is to analyze the benefits of solving the pricing problem to optimality compared with the relaxed approach in terms of lower bound quality without using any sophisticated cut generation. It will be the topic of future work to study the most profitable mix of better pricing algorithms and better cut generation algorithms.

Following the approach of the cited papers on the CVRP in order to perform significant comparisons, all the Euclidean distances have been rounded to the nearest integer although this does not guarantee the presence of the triangular inequality.

6.2 Computational results

Tables 6.1 to 6.6 report on the computational experiment performed on the instances described above.

The root node lower bound comparison between instances solved with the relaxed pricing algorithm and the exact pricing algorithm is reported both with and without the use of 2-path cuts.

Tables are organized as follows: the instance name (the instance name contains itself the problem dimension and the number of available vehicles) and the optimal solution are reported first; then the root node lower bound LB_r obtained with the relaxed pricing algorithm and the root node lower bound LB_e obtained with the exact pricing algorithm both without the use of 2-path cuts, the two percentage gaps G_r and G_e for the relaxed and exact pricing algorithm, respectively; finally the root node lower bounds LB_r^c and LB_e^c obtained with the relaxed and exact pricing algorithm, respectively, with the addition of 2-path cuts and the two percentage gaps G_r^c and G_e^c.

All the percentage gaps are computed as follows:

$$G = 100 \cdot (UB - LB)/UB$$

The value UB represents the value of the best known integer solution at the end of the computation. Tables 6.4 to 6.6 report on the comparison between the exact pricing algorithm and BCP.

Tables are organized as follows: the instance name and the optimal solution are reported again for the sake of completeness. Then, for the two algorithms, the total number of nodes of the search tree, the computing time (in seconds)

Instance	IP_{opt}	LB_r	LB_e	G_r	G_e	LB_r^c	LB_e^c	G_r^c	G_e^c
A-n37-k5	669	657.2	659.7	1.76	1.39	662.1	667.2	1.03	0.27
A-n37-k6	949	900.3	930.8	5.13	1.92	932.1	934.9	1.78	1.49
A-n38-k5	730	701.2	730.0	3.95	0.00	711.1	730.0	2.59	0.00
A-n39-k5	822	801.4	806.8	2.51	1.85	805.1	811.7	2.06	1.25
A-n39-k6	831	800.1	805.8	3.72	3.03	809.9	810.4	2.54	2.48
A-n44-k6	937	911.3	922.3	2.74	1.57	922.1	925.2	1.59	1.26
A-n45-k6	944	894.2	944.0	5.28	0.00	936.7	944.0	0.77	0.00
A-n45-k7	1146	1012.1	1137.8	1136.8	0.80	1122.9	1139.2	2.02	0.59
A-n46-k7	914	887.2	900.9	2.93	1.43	900.7	905.6	1.46	0.92
A-n48-k7	1073	1031.1	1049.3	3.90	2.21	1054.3	1061.0	1.74	1.12
A-n53-k7	1010	978.5	995.6	3.12	1.43	991.7	999.8	1.81	1.01
A-n54-k7	1167	1114.0	1141.6	4.54	2.18	1141.7	1145.0	2.17	1.89
A-n55-k9	1073	1025.4	1056.6	4.44	1.53	1055.4	1061.3	1.64	1.09
A-n60-k9	1354	1305.6	1337.3	3.57	1.23	1341.7	1351.0	0.91	0.22
A-n61-k9	1034	996.8	1034.0	3.60	0.00	1020.3	1034.0	1.32	0.00
A-n62-k8	1288	1222.7	1261.3	5.07	2.07	1266.1	1275.1	1.70	1.00
A-n63-k9	1616	1564.8	1589.3	3.17	1.65	1597.1	1605.3	1.17	0.66
A-n63-k10	1314	1267.4	1296.3	3.55	1.35	1291.4	1299.8	1.72	1.08
A-n64-k9	1401	1353.3	1356.1	3.40	3.20	1355.2	1357.2	3.27	3.13
A-n65-k9	1174	1133.0	1143.1	3.49	2.63	1143.1	1157.2	2.63	1.43
A-n69-k9	1159	1113.2	1132.7	3.95	2.27	1136.6	1147.4	1.93	1.00
A-n80-k10	1763	1712.2	1742.1	2.88	1.19	1742.4	1760.9	1.17	0.12

Table 6.1: Lower bound comparison - CVRP set A

are reported. For the exact pricing algotithm the duality gap at the end of the computation is also reported. Computing time reported in Fukasawa et al. have been multiplied by a factor of 1.5 to compare the computer performances using the Linpack benchmark, available at http://performance.netlib.org.

The computational results for instance set B by Augerat et al [70] and instance set M by Christofides, Mingozzi and Toth [66] have not been reported since the exact pricing algorithm failed to compute significant results in a reasonable amount of time. For example the instance B-n38-k6, that has been solved by Fukasawa et al. [79] in less than 15 seconds and with 14 branching decisions, required more than 800 seconds and 500 branching decisions to be solved by the exact pricing algorithm. This unexpected behaviour of the exact pricing algorithm should be investigated further.

Table 6.1 shows that, for the instance set A, the relaxed pricing algorithm without any cut provides a lower bound that is under the optimal solution from 1.76% up to 5.13% while the worst gap provided by the exact pricing algorithm without cuts is 3.20%. 2-path cuts are useful both for relaxed and exact pricing

Instance	IP_{opt}	LB_r	LB_e	G_r	G_e	LB_r^c	LB_e^c	G_r^c	G_e^c
P-n16-k8	450	431.3	443.7	4.16	1.40	446.1	448.0	0.87	0.44
P-n19-k2	212	200.4	211.0	5.47	0.47	210.3	212.0	0.80	0.00
P-n20-k2	216	201.5	212.0	6.71	1.85	211.2	215.5	2.22	0.23
P-n21-k2	211	203.5	211.0	3.55	0.00	204.7	211.0	2.99	0.00
P-n22-k2	216	202.1	215.5	6.44	0.23	212.1	215.5	1.81	0.23
P-n22-k8	603	600.1	603.0	0.48	0.00	603.0	603.0	0.00	0.00
P-n23-k8	529	526.1	529.0	0.55	0.00	529.0	529.0	0.00	0.00
P-n40-k5	458	449.1	452.5	1.94	1.20	450.2	453.9	1.70	0.90
P-n45-k5	510	491.5	502.5	3.63	1.47	492.1	503.1	3.51	1.35
P-n50-k7	554	539.1	546.2	2.69	1.41	541.3	547.7	2.29	1.14
P-n50-k8	631	612.1	615.6	3.00	2.44	614.0	616.1	2.69	2.36
P-n50-k10	696	671.4	687.5	3.53	1.22	673.2	688.1	3.28	1.14
P-n51-k10	741	721.5	733.8	2.63	0.97	724.8	734.2	2.19	0.92
P-n55-k7	568	551.8	556.8	2.85	1.97	553.1	558.3	2.62	1.71
P-n55-k8	588	561.3	575.5	4.54	2.13	563.1	577.8	4.23	1.73
P-n55-k10	694	676.2	680.2	2.56	1.99	679.4	681.3	2.10	1.83
P-n55-k15	989	963.0	969.0	2.63	2.02	965.9	971.8	2.34	1.74
P-n60-k10	744	729.6	736.9	1.94	0.95	733.6	739.7	1.40	0.58
P-n60-k15	968	950.0	960.0	1.86	0.83	952.0	962.8	1.65	0.54
P-n65-k10	792	766.1	785.6	3.27	0.81	768.2	787.1	3.01	0.62
P-n70-k10	827	785.2	810.7	5.05	1.97	786.4	813.4	4.91	1.64
P-n76-k4	593	582.3	584.4	1.80	1.45	584.8	586.1	1.38	1.16
P-n76-k5	627	601.8	609.4	4.02	2.81	605.2	611.3	3.48	2.50
P-n101-k4	681	655.3	667.1	3.77	2.04	659.0	672.3	3.23	1.28

Table 6.2: Lower bound comparison - CVRP set P

Instance	IP_{opt}	LB_r	LB_e	G_r	G_e	LB_r^c	LB_e^c	G_r^c	G_e^c
E-n13-k4	247	244.1	247.0	1.17	0.00	245.6	247.0	0.57	0.00
E-n22-k4	375	349.3	375.0	6.85	0.00	351.2	375.0	6.35	0.00
E-n23-k3	569	559.8	567.1	1.62	0.33	561.5	569.0	1.32	0.00
E-n30-k3	534	522.7	531.2	2.12	0.52	523.7	531.7	1.93	0.43
E-n31-k7	379	369.0	374.5	2.64	1.19	371.2	376.8	2.06	0.58
E-n33-k4	835	811.2	822.7	2.85	1.47	819.1	828.1	1.90	0.83
E-n51-k5	521	512.9	515.4	1.55	1.07	514.7	516.8	1.21	0.81
E-n76-k7	682	663.3	665.6	2.74	2.40	664.1	667.1	2.62	2.18
E-n76-k8	735	716.7	720.1	2.49	2.03	722.5	725.1	1.70	1.35
E-n76-k10	830	811.4	814.7	2.24	1.84	814.8	816.9	1.83	1.58
E-n76-k14	1021	999.6	1000.9	2.10	1.97	1001.7	1005.7	1.89	1.50
E-n101-k8	815	786.4	796.0	3.51	2.33	795.4	806.1	2.40	1.09
E-n101-k14	1067	1045.1	1046.3	2.05	1.94	1046.1	1049.6	1.96	1.63

Table 6.3: Lower bound comparison - CVRP set E

algorithms. The percentage gap for the relaxed pricing is reduced and it varies between 0.91% and 3.27%. For the exact pricing the gap is reduced between 0.0% and 3.13%. The small gap reduction for instance A-n64-k9 needs future investigations. Table 6.2 shows results comparable with those of set A. The relaxed pricing algorithm without any cut provides lower bounds from 0.48% to 6.44% far from the optimal solution. The exact pricing algorithm between 0.00% and 2.81%. The use of 2-path cuts reduced the gaps: the relaxed pricing gap varies between 0.00% and 4.23% and the exact pricing gap between 0.0% and 2.50%. The contribution of elementary paths providing good lower bounds is more evident in table 6.3. For example instance E-n22-k4 is solved at the root node of the search tree by the exact pricing algorithm while the relaxed pricing algorithm has a considerable gap (6.83% and 6.35%). It should be pointed out that this instance is solved to optimality at the root node of the search tree by Fukasawa et al. [79] using more sophisticated cut-generation algorithms. Gaps vary from 1.17% to 6.85% and from 0.57% and 6.35% for the relaxed pricing algorithm without and with 2-path cuts, respectively. They vary from 0.00% to 2.40% and from 0.00% and 2.18% for the exact pricing algorithm without and with 2-path cuts, respectively.

Tables 6.4 to 6.6 show, as expected, that the exact pricing algorithm does not represent the state-of-the-art algorithm for the CVRP. It should be noticed, however, that two instances of set A (A-38-k5 and A-45-k6) were solved to optimality at the root node of the search tree while BCP did not. For other instances the exact pricing algorithm is competitive only for a subset of instances. For other instances it is dominated by the BCP algorithm. Most difficult instances have not been solved by the exact pricing algorithm because the proposed branching decisions do not help to close the gap in a reasonable number of iterations. Moreover the pricing subproblem becomes harder to solve in the early nodes of the search tree for quasi-unconstrained instances. It is to notice that the duality gap is always acceptable. The worst gap is 2.55% while the average gap, for unsolved instances, is 1.12%.

Conclusions The computational experience shows that column generation coupled with the exact solution of the pricing problem allows to compute better lower bounds compared with the relaxed pricing approach. The use of simple cut-generation strategies (2-path cuts) gives limited benefits. It has been shown by Fukasawa et al. [79] that more sophisticated cut-generation strategies are able to close the duality gap in smaller amount of time. It is clear that, for unconstrained routing problems, the branching decisions do not help to increase significantly the lower bounds. This consideration suggests to embed both exact pricing algorithms and cut-generation in the same algorithm in order to obtain better lower bounds.

Instance	IP_{opt}	BCP		Exact pricing		
		$Nodes$	$Time(s)$	$Nodes$	$Time(s)$	$Gap(\%)$
A-n37-k5	669	8	28.50	27	162.64	0.00
A-n37-k6	949	74	568.50	973	1017.13	0.00
A-n38-k5	730	52	39.00	1	1.16	0.00
A-n39-k5	822	11	250.50	1995	3419.61	0.00
A-n39-k6	831	11	147.00	2019	3719.11	0.00
A-n44-k6	937	6	135.00	69	182.22	0.00
A-n45-k6	944	11	255.00	1	7.84	0.00
A-n45-k7	1146	26	496.50	7	35.22	0.00
A-n46-k7	914	3	138.00	7	112.82	0.00
A-n48-k7	1073	8	249.00	135	954.70	0.00
A-n53-k7	1010	16	916.50	463	5132.70	0.00
A-n54-k7	1167	90	2113.50	7857		0.01
A-n55-k9	1073	7	126.00	391	978.10	0.00
A-n60-k9	1354	224	4620.00	4198		0.17
A-n61-k9	1034	121	2824.50	697	3128.10	0.00
A-n62-k8	1288	101	4653.00	5918		0.12
A-n63-k9	1616	49	1569.00	8911		0.31
A-n63-k10	1314	387	7482.00	2118		0.91
A-n64-k9	1401	648	16881.00	2081		2.55
A-n65-k9	1174	17	774.00	1159		0.55
A-n69-k9	1159	391	10756.50	991		0.71
A-n80-k10	1763	153	9696.00	59		0.07

Table 6.4: Search tree comparison - CVRP set A

Instance	IP_{opt}	BCP		Exact pricing		
		$Nodes$	$Time(s)$	$Nodes$	$Time(s)$	$Gap(\%)$
P-n16-k8	450	3	1.50	3	0.12	0.00
P-n19-k2	212	1	1.50	1	21.10	0.00
P-n20-k2	216	9	1.50	5	7.56	0.00
P-n21-k2	211	1	1.50	1	8.44	0.00
P-n22-k2	216	2	3.00	3	11.46	0.00
P-n22-k8	603	1	4.50	1	0.05	0.00
P-n23-k8	529	1	27.00	1	0.08	0.00
P-n40-k5	458	5	51.00	111	216.30	0.00
P-n45-k5	510	11	291.00	533	1072.53	0.00
P-n50-k7	554	7	214.50	235	336.12	0.00
P-n50-k8	631	1084	13908.00	5029	7321.32	0.00
P-n50-k10	696	65	456.00	433	297.19	0.00
P-n51-k10	741	22	157.50	181	225.15	0.00
P-n55-k7	568	450	6973.50	3511		0.18
P-n55-k8	588	196	2733.00	4881		0.17
P-n55-k10	694	1556	13614.00	7675		0.29
P-n55-k15	989	398	2916.00	3571	2773.52	0.00
P-n60-k10	744	52	855.00	181	524.32	0.00
P-n60-k15	968	76	663.00	85	59.19	0.00
P-n65-k10	792	23	633.00	167	1012.06	0.00
P-n70-k10	827	1752	36058.50	4451		0.36
P-n76-k4	593	59	858.00	2181		0.11
P-n76-k5	627	3399	21819.00	3415		0.44
P-n101-k4	681	23	1879.50	1025		1.01

Table 6.5: Search tree comparison - CVRP set P

Instance	IP_{opt}	BCP		Exact pricing		
		$Nodes$	$Time(s)$	$Nodes$	$Time(s)$	$Gap(\%)$
E-n13-k4	247	1	0.00	1	0.00	0.00
E-n22-k4	375	1	0.00	1	7.50	0.00
E-n23-k3	569	1	0.00	1	16.91	0.00
E-n30-k3	534	6	10.50	21	195.30	0.00
E-n31-k7	379	2	9.00	15	159.20	0.00
E-n33-k4	835	5	22.50	145	678.30	0.00
E-n51-k5	521	8	97.50	671	2195.40	0.00
E-n76-k7	682	1712	69780.00	5631		1.88
E-n76-k8	735	1031	34336.50	4923		1.01
E-n76-k10	830	4292	121083.00	6791		0.79
E-n76-k14	1021	6678	72955.50	1521		1.41
E-n101-k8	815	11622	1202944.50	3		1.09
E-n101-k14	1067	5848	174426.00	i 13		1.61

Table 6.6: Search tree comparison - CVRP set E

When the computation of exact pricing is too time consuming a good compromise could be the partial relaxation of elementarity constraint: the decremental state space relaxation algorithm, proposed in chapter 4, can be adapted to compute only t-nodes cycle free paths where the number of critical nodes can be imposed to be less than t. The most profitable mix of better pricing algorithms and better cut generation algorithms will be the topic of future research.

Chapter 7

Computational results: The VRPDC

7.1 Instances

The instance sets for the VRPDC used in the literature have been derived from instance sets for the CVRP adding the pick-up requests of the customers. Other instance sets have been derived from Solomon's [61] data set for the CVRPTW neglecting the time windows and adding the pick-up requests.

Since the VRPDC is a less studied variant of the CVRP there are no homogeneous and comparable results. In the reminder I will focus on results presented in a previous work (Dell'Amico, Righini and Salani [46]). No other optimal methods have been presented so far, although Angelelli and Mansini [13] presented an optimal method for the VRPDC with time windows and, in principle, this problem covers the VRPDC.Dethloff [10] and Bianchessi and Righini [65] presented some heuristics and meta-heuristics to solve the VRPDC where the same problem formulation has been called VRP with Simultaneous Pickup and Delivery. I recall that the VRPDC differs from the VRPPD. In VRPPD the pickup and the delivery operations associated with a single request are performed in two different vertices because the source and the destination of the load to be carried are specified. In the VRPDC the delivery and pick-up are associated with the same vertex and represent a quantity of good to be delivered and a quantity of waste to be picked-up.

Class 1S was made of 18 instances generated as described by Toth and Vigo [71]. The number of customers is equal to 20 or 40. Customer coordinates are uniformly distributed in the intervals [0, 24000] for the x values and [0, 32000] for the y values; the depot is located at $(12000, 16000)$. The cost of each arc was defined as the Euclidean distance rounded up to an integer value. Demands were generated at random from a normal distribution with mean value equal to 50 and standard deviation equal to 20. The fraction of delivery customers is equal to $\frac{1}{2}$, $\frac{2}{3}$ or $\frac{4}{5}$ (this is indicated by the percentage values 50, 66 and 80 in the name of the instances). The vehicles capacity varies in {100, 150, 200}.

Class 1C was obtained from class 1S using the following method: all demands of the corresponding instance in class 1S were considered as delivery demands and the pick-up demand of each customer i was computed as $p_i = (0.5 + r)d_i$, where r was taken from a uniform distribution in the interval [0,1].

Class 2S consists of 36 instances, obtained from four real world CVRP instances of the VRPLIB with N equal to 20 or 40. For each CVRP instance nine VRPSDC instances were generated, each one corresponding to a fraction of delivery customer equal to $\frac{1}{2}$, $\frac{2}{3}$ and $\frac{4}{5}$ and to a vehicle capacity equal to 150, 200 and 300.

Class 2C was obtained from class 2S with the same technique as class 1C.

Class S consists of twelve instances derived from Solomon's instances r101, c101 and rc101, originally proposed for the CVRPTW by Solomon [61]. I considered the first 20 or 40 customers, I neglected the time-windows and I followed the method proposed by Angelelli and Mansini [13] to generate composite demands: the demands given in Solomon's instances were assumed to represent delivery demands d_i, while pick-up demands p_i were generated as $p_i = \lfloor (1 - \gamma)\ d_i \rfloor$ if i is even, and $p_i = \lfloor (1 + \gamma)\ d_i \rfloor$ if i is odd. For each instance in Solomon's benchmark I generated two VRPSDC instances with $\gamma = 0.2$ and $\gamma = 0.8$. Vehicles capacity was set to 100. The Euclidean distance between customers was rounded up to a multiple of 0.1 to guarantee that the triangular inequality held.

7.2 Computational results

Tables 7.1 to 7.11 report on the computational experiment performed on the instances described above.

The root node lower bound comparison between instances solved with the relaxed pricing algorithm and the exact pricing algorithm is reported both with and without the use of 2-path cuts.

Tables are organized as follows: the instance name (the instance name contains the problem dimension and the distribution of delivery and pick-up requests), the number of vehicles and the optimal solution are reported first; then the root node lower bound LB_r and LB_r^c obtained with the relaxed pricing without and with the use of 2-path cuts, respectively. Then the two percentage gaps G_r and G_r^c for the relaxed pricing algorithm. Similarly for the exact pricing algorithm the root node lower bound LB_e and LB_e^c without and with 2-path cuts and the percentage gaps G_e and G_e^c.

All the percentage gaps are computed as follows:

$$G = 100 \cdot (UB - LB)/UB$$

where UB represents the best feasible solution available at the end of the computation. Table 7.6 reports on the average percentage gaps.

Tables 7.7 to 7.11 report on the comparison between the exact pricing algorithm and the relaxed pricing algorithm. The comparison was made leaving other algorithms and parameters unchanged, such as the branching decisions and the cut generations, although the exact and the relaxed pricing algorithms can be differently tuned.

It should be pointed out that differences between the results reported in [46] are due to the different search strategy and to the fact that the minimum number of vehicles was imposed while here it has been left free. An asterisk on the number of vehicles means that the optimal solution uses one vehicle more than the minimum required.

The computing time was limited to one hour.

Tables are organized as follows: the instance name and the optimal solution are reported again for the sake of completeness. Then, for the two algorithms, the total number of nodes of the search tree, the computing time (in seconds) and the duality gap at the end of the computation are reported. The time limit was set to two hours.

Table 7.1 shows that none of the instances where closed at the root node by using the relaxed pricing algorithm without cuts. The use of 2-path cuts allowed to close three instances at the root node with the relaxed pricing algorithm. On the other hand the use of exact pricing allowed to solve three instances at the root node of the search tree without cuts. The use of 2-path cuts increased their number to nine. The worst percentage gap for the relaxed pricing is 3.85% while for the exact pricing is 2.97%. The exact pricing algorithm allowed to solve nine instances at the root node of the search tree while the relaxed one only three.

Table 7.2 shows similar results. Both the exact pricing algorithm and the relaxed one allowed to solve several instances at the root node of the search with the use of 2-path cuts because the C set is more constrained. The use of 2-path cuts was useful for both algorithms. The worst percentage gap for the relaxed pricing is 3.30% while for the exact pricing is 2.87%.

In table 7.3 the benefits of the exact solution of the pricing problem are more evident. The worst gap of the relaxed pricing reaches 5.68% while for the exact pricing it is under the 2.95%.

Table 7.4 shows that several instances have been solved at the root node of the search tree by the exact pricing algorithm. The worst gap of the relaxed pricing is

Instance	K	IP_{opt}	LB_r	LB_r^c	G_r	G_r^c	LB_e	LB_e^c	G_e	G_e^c
1S_20_50_1	6	181689	180422	180845	0.70	0.46	181689	181689	0.00	0.00
1S_20_50_2	4	151472	150332	150776	0.75	0.46	150332	150776	0.75	0.46
1S_20_50_3	3	136107	132268	134877	2.82	0.90	135475	136107	0.46	0.00
1S_20_66_1	7	189396	189184	189396	0.11	0.00	189184	189396	0.11	0.00
1S_20_66_2	5	155853	154304	155462	0.99	0.25	155310	155853	0.35	0.00
1S_20_66_3	4	136489	133302	136489	2.33	0.00	136489	136489	0.00	0.00
1S_20_80_1	8	210732	205167	208436	2.64	1.09	205167	208436	2.64	1.09
1S_20_80_2	6	166408	163104	165389	1.99	0.61	163483	166129	1.76	0.17
1S_20_80_3	4	147820	143357	147019	3.02	0.54	147312	147820	0.34	0.00
1S_40_50_1	10	357430	346819	352953	2.97	1.25	346819	353559	2.97	1.08
1S_40_50_2	7	269590	261515	265254	3.00	1.61	266518	269388	1.14	0.07
1S_40_50_3	5	229044	221605	226771	3.25	0.99	225748	229044	1.44	0.00
1S_40_66_1	13	377279	372905	375233	1.16	0.54	375089	376809	0.58	0.12
1S_40_66_2	9	291008	279810	286857	3.85	1.43	285206	290311	1.99	0.24
1S_40_66_3	7	241347	233872	236652	3.10	1.95	239075	239401	0.94	0.81
1S_40_80_1	16	425911	424955	425911	0.22	0.00	425911	425911	0.00	0.00
1S_40_80_2	11	324920	316456	320881	2.60	1.24	321867	324920	0.94	0.00
1S_40_80_3	8	270313	261809	265735	3.15	1.69	268180	269767	0.79	0.20

Table 7.1: Lower bound comparison - VRPDC set 1S

Instance	K	IP_{opt}	LB_r	LB_r^c	G_r	G_r^c	LB_e	LB_e^c	G_e	G_e^c
1C_20_50_1	11	265504	264918	265504	0.22	0.00	264918	265504	0.22	0.00
1C_20_50_2	8	206425	203609	206005	1.36	0.20	203609	206005	1.36	0.20
1C_20_50_3	6	171236	170984	171236	0.15	0.00	171166	171236	0.04	0.00
1C_20_66_1	14	298493	297476	298493	0.34	0.00	297476	298493	0.34	0.00
1C_20_66_2	7	192727	192727	192727	0.00	0.00	192727	192727	0.00	0.00
1C_20_66_3	6	178629	172726	176650	3.30	1.11	173500	176899	2.87	0.97
1C_20_80_1	14	304412	304412	304412	0.00	0.00	304412	304412	0.00	0.00
1C_20_80_2	8	218072	216148	218072	0.88	0.00	216148	218072	0.88	0.00
1C_20_80_3	6	177215	174937	175893	1.29	0.75	176767	177215	0.25	0.00
1C_40_50_1	23*	601817	597900	601817	0.65	0.00	597925	601817	0.65	0.00
1C_40_50_2	15	402309	397719	399231	1.14	0.77	397719	399231	1.14	0.77
1C_40_50_3	11	334830	326286	328114	2.55	2.01	331522	333265	0.99	0.47
1C_40_66_1	24	630257	629355	630257	0.14	0.00	629355	630257	0.14	0.00
1C_40_66_2	15	426517	421559	422152	1.16	1.02	421931	422403	1.08	0.96
1C_40_66_3	11	331459	321946	326486	2.87	1.50	327775	328942	1.11	0.76
1C_40_80_1	25	647539	647539	647539	0.00	0.00	647539	647539	0.00	0.00
1C_40_80_2	15	424368	421665	421828	0.64	0.60	421682	422171	0.63	0.52
1C_40_80_3	11	332957	329326	330639	1.09	0.70	332935	332957	0.01	0.00

Table 7.2: Lower bound comparison - VRPDC set 1C

Instance	K	IP_{opt}	LB_r	LB_r^c	G_r	G_r^c	LB_e	LB_e^c	G_e	G_e^c
2S_20_50_1	6	8769	8608	8731	1.84	0.43	8608	8731	1.84	0.43
2S_20_50_2	5*	7348	7348	7348	0.00	0.00	7348	7348	0.00	0.00
2S_20_50_3	3	6445	6283	6387	2.51	0.90	6415	6445	0.47	0.00
2S_20_66_1	7	9129	9129	9129	0.00	0.00	9129	9129	0.00	0.00
2S_20_66_2	5	7470	7470	7470	0.00	0.00	7470	7470	0.00	0.00
2S_20_66_3	4*	6890	6699	6828	2.77	0.90	6874	6874	0.23	0.23
2S_20_80_1	8	10707	10707	10707	0.00	0.00	10707	10707	0.00	0.00
2S_20_80_2	7*	8773	8773	8773	0.00	0.00	8773	8773	0.00	0.00
2S_20_80_3	4	7058	6989	7058	0.98	0.00	7058	7058	0.00	0.00
2S_40_50_1	10	18282	18220	18220	0.34	0.34	18220	18222	0.34	0.33
2S_40_50_2	8	14603	14402	14495	1.38	0.74	14549	14592	0.37	0.08
2S_40_50_3	5	11610	10956	11012	5.63	5.15	11268	11304	2.95	2.64
2S_40_66_1	13*	17932	17699	17805	1.30	0.71	17823	17849	0.61	0.46
2S_40_66_2	9	15307	15001	15094	2.00	1.39	15091	15176	1.41	0.86
2S_40_66_3	6	11725	11243	11405	4.11	2.73	11702	11725	0.20	0.00
2S_40_80_1	17	20665	20653	20665	0.06	0.00	20653	20665	0.06	0.00
2S_40_80_2	13*	17201	16629	16887	3.33	1.83	16969	17056	1.35	0.84
2S_40_80_3	8	13317	12561	12750	5.68	4.26	13042	13063	2.07	1.91

Table 7.3: Lower bound comparison - VRPDC set 2S

Instance	K	IP_{opt}	LB_r	LB_r^c	G_r	G_r^c	LB_e	LB_e^c	G_e	G_e^c
2C_20_50_1	11	12720	12720	12720	0.00	0.00	12720	12720	0.00	0.00
2C_20_50_2	8*	10054	10054	10054	0.00	0.00	10054	10054	0.00	0.00
2C_20_50_3	6	8387	8320	8372	0.80	0.18	8375	8387	0.14	0.00
2C_20_66_1	12	14578	14539	14578	0.27	0.00	14539	14578	0.27	0.00
2C_20_66_2	9*	10861	10838	10861	0.21	0.00	10838	10861	0.21	0.00
2C_20_66_3	5	8160	7822	7915	4.14	3.00	8018	8079	1.74	0.99
2C_20_80_1	10	12802	12802	12802	0.00	0.00	12802	12802	0.00	0.00
2C_20_80_2	8	10087	9901	10036	1.84	0.51	9970	10087	1.16	0.00
2C_20_80_3	5	8317	8051	8220	3.20	1.17	8225	8317	1.11	0.00
2C_40_50_1	23*	26988	26988	26988	0.00	0.00	26988	26988	0.00	0.00
2C_40_50_2	17*	21710	21490	21571	1.01	0.64	21513	21581	0.91	0.59
2C_40_50_3	10	15523	14968	15066	3.58	2.94	15310	15323	1.37	1.29
2C_40_66_1	22	25981	25971	25981	0.04	0.00	25981	25981	0.00	0.00
2C_40_66_2	16*	21317	20994	21202	1.52	0.54	21056	21234	1.22	0.39
2C_40_66_3	10	15293	14771	14949	3.41	2.25	15072	15114	1.45	1.17
2C_40_80_1	22*	26122	26122	26122	0.00	0.00	26122	26122	0.00	0.00
2C_40_80_2	16	20652	20426	20516	1.09	0.66	20436	20521	1.05	0.63
2C_40_80_3	10	15365	14930	15121	2.83	1.59	15292	15317	0.48	0.31

Table 7.4: Lower bound comparison - VRPDC set 2C

Instance	K	IP_{opt}	LB_r	LB_r^c	G_r	G_r^c	LB_e	LB_e^c	G_e	G_e^c
c101_20_02	4	272	266	269	2.21	1.10	268	269	1.47	1.10
c101_20_08	4	279	273	275	2.15	1.43	275	277	1.43	0.72
r101_20_02	3	329	320	321	2.74	2.43	323	324	1.82	1.52
r101_20_08	3	342	324	329	5.26	3.80	334	336	2.34	1.75
rc101_20_02	5	428	426	428	0.47	0.00	428	428	0.00	0.00
rc101_20_08	5	458	448	454	2.18	0.87	448	454	2.18	0.87
c101_40_02	8	551	529	532	3.99	3.45	532	536	3.45	2.72
c101_40_08	8	569	548	553	3.69	2.81	555	558	2.46	1.93
r101_40_02	6	601	584	589	2.83	2.00	593	594	1.33	1.16
r101_40_08	7*	627	610	612	2.71	2.39	621	624	0.96	0.48
rc101_40_02	9	886	836	878	5.64	0.90	841	881	5.08	0.56
rc101_40_08	9	926	860	914	7.13	1.30	867	920	6.37	0.65

Table 7.5: Lower bound comparison - VRPDC set S

Instance set	*Avg.* G_r(%)	*Avg.* G_r^c(%)	*Avg.* G_e(%)	*Avg.* G_e^c(%)
1S_20	1.71	0.48	0.71	0.19
1S_40	2.59	1.19	1.20	0.28
1C_20	0.84	0.23	0.66	0.13
1C_40	1.14	0.73	0.64	0.39
2S_20	0.90	0.25	0.28	0.07
2S_40	2.65	1.90	1.04	0.79
2C_20	1.16	0.54	0.51	0.11
2C_40	1.50	0.96	0.72	0.49
S_20	2.50	1.61	1.54	0.99
S_40	4.33	2.14	3.27	1.25

Table 7.6: Lower bound comparison

Instance	K	IP_{opt}	Relaxed Pricing			Exact pricing		
			$Nodes$	$T(s)$	$G(\%)$	$Nodes$	$T(s)$	$G(\%)$
1S_20_50_1	6	181689	11	0.3	0.00	1	0.04	0.00
1S_20_50_2	4	151472	17	3.08	0.00	5	0.58	0.00
1S_20_50_3	3	136107	9	6.82	0.00	1	2.13	0.00
1S_20_66_1	7	189396	1	0.08	0.00	1	0.09	0.00
1S_20_66_2	5	155853	3	0.77	0.00	1	0.59	0.00
1S_20_66_3	4	136489	1	2.35	0.00	1	1.41	0.00
1S_20_80_1	8	210732	11	0.23	0.00	11	0.26	0.00
1S_20_80_2	6	166408	11	1.12	0.00	3	0.4	0.00
1S_20_80_3	4	147820	9	6.68	0.00	1	2.61	0.00
1S_40_50_1	10	357430	1389	361.36	0.00	377	82.41	0.00
1S_40_50_2	7	269590	549	669.41	0.00	5	20.09	0.00
1S_40_50_3	5	229044	25	183.74	0.00	1	148.06	0.00
1S_40_66_1	13	377279	73	10.82	0.00	5	1.49	0.00
1S_40_66_2	9	291008	537	358.59	0.00	11	38.44	0.00
1S_40_66_3	7	241347	329	637.13	0.00	17	90.7	0.00
1S_40_80_1	16	425911	1	0.43	0.00	1	0.13	0.00
1S_40_80_2	11	324920	1949	830.87	0.00	1	9.46	0.00
1S_40_80_3	8	270313	4395		0.04	9	39.62	0.00

Table 7.7: Search tree comparison - VRPDC set 1S

4.14% while for the exact pricing it is 1.74%.

Table 7.5 shows that on instances derived from Solomon's sets the use of cutting planes is more useful than the exact solution of the pricing problem. Both the exact and the relaxed algorithm have a significant percentage gap without cuts. The worst gap is 7.13% and 6.37% for the relaxed and exact pricing without cuts, respectively. The use of 2-path cuts reduced the worst gap to 3.80% and 2.72%.

The average gaps table 7.6 shows that both the use of exact pricing and 2-path cuts is crucial to increase the lower bound at the root node of the search tree. There is no dominance between the two methods ($G_r^c vs. G_e$).

Tables 7.7 to 7.11 show the dominance of the exact pricing algorithm without cuts. These results complete the conclusions presented in our previous work [46] with a more deep knowledge of the problem. In this work I analyzed more deeply the pricing problem and I was able to implement more efficient algorithms for its solution (see chapter 4). The bidirectional bounded algorithm based on the concept of critical resources allowed to speed up the computation of both elementary and non-elementary paths. It should be noticed that also for instances solved at the root node of the search tree by both algorithms the exact pricing algorithm is faster. This depends on the fact that the exact pricing algorithm did not find any

Instance	K	IP_{opt}	Relaxed Pricing			Exact pricing		
			$Nodes$	$T(s)$	$G(\%)$	$Nodes$	$T(s)$	$G(\%)$
1C_20_50_1	11	265504	1	0.01	0.00	1	0.01	0.00
1C_20_50_2	8	206425	3	0.05	0.00	3	0.05	0.00
1C_20_50_3	6	171236	1	0.3	0.00	1	0.12	0.00
1C_20_66_1	14	298493	1	0.01	0.00	1	0.01	0.00
1C_20_66_2	7	192727	1	0.01	0.00	1	0.02	0.00
1C_20_66_3	6	178629	71	2.04	0.00	43	1.13	0.00
1C_20_80_1	14	304412	1	0	0.00	1	0.01	0.00
1C_20_80_2	8	218072	1	0.02	0.00	1	0.03	0.00
1C_20_80_3	6	177215	15	0.95	0.00	1	0.3	0.00
1C_40_50_1	23*	601817	1	0.04	0.00	1	0.05	0.00
1C_40_50_2	15	402309	81	4.68	0.00	51	3.03	0.00
1C_40_50_3	11	334830	2085	252.54	0.00	31	16.4	0.00
1C_40_66_1	24	630257	1	0.02	0.00	1	0.02	0.00
1C_40_66_2	15	426517	39	3.32	0.00	37	2.94	0.00
1C_40_66_3	11	331459	1839	299.71	0.00	145	43.22	0.00
1C_40_80_1	25	647539	1	0.01	0.00	1	0.01	0.00
1C_40_80_2	15	424368	55	3.86	0.00	25	2.33	0.00
1C_40_80_3	11	332957	8633	986.41	0.00	1	1.94	0.00

Table 7.8: Search tree comparison - VRPDC set 1C

negative reduced cost columns in the earlier iterations of the column generation process while the relaxed pricing algorithm, that clearly is faster than the exact one, requires more column generation iterations. All 20 nodes instances of sets 1S to 2C have been solved in less than 3 seconds by the exact pricing algorithm while the 40 nodes instances required at most 166 seconds and 445 search tree nodes. The instances derived from the Solomon's sets seem to be more difficult. In particular instance c_101_40_02 has not been solved within one hour of computation by both algorithm. Fixing the number of vehicles to 8 (that corresponds to the minimum number of vehicles required), the exact pricing algorithm required 993 seconds and 201 nodes of the search tree while the relaxed pricing algorithm required 1918 seconds and 719 nodes.

Conclusions Computational results showed that the exact pricing algorithm outperforms the relaxed pricing one. This complete our previous conclusions (see [46]) giving to the algorithms that solve the pricing problem the crucial role in the solution of the VRPDC. The research performed on the pricing problem (see 4) allowed to devise more efficient algorithms both exact and relaxed. It has been showed that the lower bound increase at the early nodes of the search tree is useful

Instance	K	IP_{opt}	Relaxed Pricing			Exact pricing		
			$Nodes$	$T(s)$	$G(\%)$	$Nodes$	$T(s)$	$G(\%)$
2S_20_50_1	6	8769	5	0.22	0.00	5	0.22	0.00
2S_20_50_2	5*	7348	1	0.08	0.00	1	0.05	0.00
2S_20_50_3	3	6445	7	5.4	0.00	1	0.88	0.00
2S_20_66_1	7	9129	1	0.04	0.00	1	0.04	0.00
2S_20_66_2	5	7470	1	0.04	0.00	1	0.04	0.00
2S_20_66_3	4*	6890	19	6.39	0.00	3	0.94	0.00
2S_20_80_1	8	10707	1	0.03	0.00	1	0.03	0.00
2S_20_80_2	7*	8773	1	0.1	0.00	1	0.09	0.00
2S_20_80_3	4	7058	1	1.94	0.00	1	0.16	0.00
2S_40_50_1	10	18282	59	20.73	0.00	31	11.14	0.00
2S_40_50_2	8	14603	27	18.9	0.00	5	7.2	0.00
2S_40_50_3	5	11610	991	1458.86	0.00	5	110.03	0.00
2S_40_66_1	13*	17932	9	3.41	0.00	5	1.5	0.00
2S_40_66_2	9	15307	161	80.2	0.00	31	31.51	0.00
2S_40_66_3	6	11725	1649	2644.58	0.00	1	25.8	0.00
2S_40_80_1	17	20665	1	0.28	0.00	1	0.29	0.00
2S_40_80_2	13*	17201	15347		0.22	445	166.34	0.00
2S_40_80_3	8	13317	7441		1.33	73	147.41	0.00

Table 7.9: Search tree comparison - VRPDC set 2S

to speed up the computation. This goal can be achieved with the use of an exact pricing algorithm and a cutting plane algorithm. It will be the topic of future research to find the best mix of these two strategies.

Instance	K	IP_{opt}	Relaxed Pricing			Exact pricing		
			$Nodes$	$T(s)$	$G(\%)$	$Nodes$	$T(s)$	$G(\%)$
2C_20_50_1	11	12720	1	0.02	0.00	1	0.01	0.00
2C_20_50_2	8*	10054	1	0.04	0.00	1	0.04	0.00
2C_20_50_3	6	8387	5	0.8	0.00	1	0.22	0.00
2C_20_66_1	12	14578	1	0.03	0.00	1	0.03	0.00
2C_20_66_2	9*	10861	1	0.07	0.00	1	0.06	0.00
2C_20_66_3	5	8160	79	4.3	0.00	17	1.89	0.00
2C_20_80_1	10	12802	1	0.01	0.00	1	0.02	0.00
2C_20_80_2	8	10087	11	0.35	0.00	1	0.11	0.00
2C_20_80_3	5	8317	19	3.52	0.00	1	0.49	0.00
2C_40_50_1	23*	26988	1	0.05	0.00	1	0.05	0.00
2C_40_50_2	17*	21710	35	4.58	0.00	31	4.22	0.00
2C_40_50_3	10	15523	13933		0.28	155	48.12	0.00
2C_40_66_1	22	25981	1	0.13	0.00	1	0.06	0.00
2C_40_66_2	16*	21317	97	6.56	0.00	45	5.42	0.00
2C_40_66_3	10	15293	5195	1451.56	0.00	129	62.32	0.00
2C_40_80_1	22*	26122	1	0.12	0.00	1	0.07	0.00
2C_40_80_2	16	20652	79	10.48	0.00	69	7.77	0.00
2C_40_80_3	10	15365	535	163.18	0.00	7	7.18	0.00

Table 7.10: Search tree comparison - VRPDC set 2C

Instance	K	IP_{opt}	Relaxed Pricing			Exact pricing		
			$Nodes$	$T(s)$	$G(\%)$	$Nodes$	$T(s)$	$G(\%)$
c101_20_02	4	272	49	7.61	0.00	25	5.43	0.00
c101_20_08	4	279	11	5.65	0.00	3	2.42	0.00
r101_20_02	3	329	75	14.06	0.00	17	2.59	0.00
r101_20_08	3	342	149	57.2	0.00	17	7.59	0.00
rc101_20_02	5	428	1	0.43	0.00	1	0.12	0.00
rc101_20_08	5	458	59	3.03	0.00	37	1.59	0.00
c101_40_02	8	551	16749		1.27	8891		0.36
c101_40_08	8	569	7457		0.88	1873	2250.57	0.00
r101_40_02	6	601	145	115.22	0.00	7	41.14	0.00
r101_40_08	7*	627	3819		0.63	69	576.7	0.00
rc101_40_02	9	886	267	71.52	0.00	63	36.13	0.00
rc101_40_08	9	926	2703	1341.36	0.00	235	250.53	0.00

Table 7.11: Search tree comparison - VRPDC set S

Chapter 8

Computational results: The CVRPTW

8.1 Instances

The test instances proposed by Solomon [61] are the most commonly used to evaluate algorithms for the CVRPTW. These instances are divided into two sets. Problem set 1 allows routes with approximately 5 to 10 customers due to capacity and time constraints. Problem set 2 is less constrained and allows routes with more than 30 customers. Both sets 1 and 2 are made of customers distributed in a Euclidean plane. The two sets include three subsets of instances: the *r*-instances where customers are located randomly, the *c*-instances where customers are located in clusters and *rc*-instances where some customers are clustered and some others are randomly distributed. Each problem has 100 customers but smaller instances with 25 and 50 can be derived considering only the first customers as proposed by Kohl et al. [69]. The coordinates of the customers are the same for all problems within each type. The difference is in the width and the placement of the time window for each customer. Problem set 1 has 87 instances and most of them have been solved to optimality quite easily but there are some with 100 customers (r108, r112 and rc106) still unsolved. Recently Irnich and Villeneuve [83] solved to optimality four unsolved instances with 100 customers (r104, rc104, rc107 and rc108) in a huge amount of time (from 42770 to 986809 seconds on a 600MHz PC). Problem set 2 is the most difficult: while the *c*-set has been solved entirely except instance c204, the *r*-set and the *rc*-set have been solved partially and only for 25 and 50 customers instances. Recently Chabrier [1] solved some difficult 100 customers instances (rc202 and rc205) in more than four hours of computation on a 1.5GHz Pentium IV PC. Another very recent publication of Kallehauge et al. [5] reported new computational results obtained by their Lagrangean branch-and-cut-and-price algorithm on a Sun Fire 15K equipped with 384GB of RAM.

The convention used in this thesis for the calculation of cost and travel time is the same of Kohl et al. [69]. Cost and travel time are computed from the Euclidean distance with one decimal point truncation. Let (x_i, y_i) and (x_j, y_j) respectively the coordinates for customer i and j; then

$$c_{ij} = \frac{\lfloor 10\ \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \rfloor}{10}.$$

Travel times are calculated as $t_{ij} = c_{ij} + st_i$, where st_i represents the service time at customer i.

8.2 Computational results

The main purpose of the experiments on the CVRPTW reported here was to test the effectiveness of column generation when the pricing problem is solved to optimality with the dynamic programming algorithm described in chapter 4. This is done by a comparison with the relaxed pricing approach used in literature by the state-of-the-art algorithms by Kohl et al. [69], Cook and Rich [90] and Irnich and Villeneuve [83]. Also the comparison with Kallehauge et al. [5] is reported although the authors proposed a different approach to the solution of the CVRPTW.

The experiments are dedicated to the comparison of the lower bounds and the overall performances of the Branch-and-Price algorithms. The lower bound is "implementation-independent" and gives information on the strength of the formulation solved. On the contrary the computing time depends on the implementation and the tuning of the algorithm parameters

Tables 8.1 to 8.9 report on the experimental comparison between two branch-and-price-based algorithms. The first one solves the pricing problem with the relaxed dynamic programming algorithm with k-cycle elimination and k-cuts. The second solves the pricing problem with the decremental state space relaxation algorithm.

The results for the relaxed pricing have been taken from the papers by Kohl et al. [69], Cook and Rich [90], Irnich and Villeneuve [83] and Kallehauge et al. [5]. The papers of Kohl et al. [69] and Cook and Rich [90] provided the results for set 1, Irnich and Villeneuve [83] provided the best results for some instances of set 2 and Kallehauge et al. [5] provided additional results for both sets.

8.2.1 Lower bound comparison

Tables are organized as follows: the instance name, the number of vehicles and the optimal solution (IP_{opt}, when known) are reported first. Then the best known lower bound for relaxed pricing algorithms without cuts (LB_r) and with cuts (LB_r^c) among the results of the cited papers at the root node of the search tree for the instances of set 1. In the tables results by Kohl et al. [69] are marked with KD, by Cook and Rich [90] with CR, by Irnich and Villeneuve [83] with IV and by Kallehauge et al. [5] with KL. When the lower bound is not reported by the cited papers I report the lower bound computed with my implementation of the relaxed pricing algorithm where 2-cycle elimination is used (marked with SA in the tables). For the instances of set 2 only the lower bound at the root node with 2-cycle elimination but without any cutting plane is available since no author provided computational studies on relaxed pricing with the use of k-path cuts for set 2. Irnich and Villeneuve [83] provided some new results on set 2 with the use of 2-cuts but they did not report any lower bound. Lower bounds marked with C in the tables have been obtained from Chabrier [1]. Other lower bounds have been obtained from the computational study of Kallehauge et al. [5]. When no lower bounds are available I report the ones obtained with my implementation of relaxed pricing.

For the experimental comparison of the lower bounds obtained with the algorithms proposed in this thesis I report the lower bound at the root node before and after the use of 2-cuts: LB_e and LB_e^c respectively, the percentage distance G between LB_r and LB_e and the percentage distance G^c between LB_r^c and LB_e^c, computed as follows:

$$G = \frac{LB_e - LB_r}{LB_r}, G^c = \frac{LB_e^c - LB_r^c}{LB_r^c}.$$

Results reported in bold face mean that the exact pricing allowed to obtain the optimal solution at the root node of the search tree while the relaxed pricing did not.

The lower bound and time comparison tables for the c-instances of set 1 have not been reported since all algorithms provided a root node lower bound equal to the optimal solution and therefore the comparison is not significant. For all instances of this set the computing time is favourable to the relaxed pricing algorithm that is faster than the exact one. For the same reason for the c-instances of set 2 only the lower bound comparison for some instances has been reported.

Results for r-instances of set 1 show that the lower bound improvement is less than 2.77. The use of elementary paths always produces better lower bounds with and without the use of 2-cuts. It should be pointed out that the best root node lower bounds of column LB_r^c have been obtained with the use of k-cuts with $k \leq 6$ (see Cook and Rich [90]).

Table 8.2 shows that the lower bound improvement is always favourable to elementary paths except for instance 101 with 100 customers where k-cuts, with

Instance	K	IP_{opt}	LB_r	LB_r^c	LB_e	LB_e^c	G	G^c
R101.25	8	617.1	617.1	617.1 ^{KD}	617.1	617.1	0.00	0.00
R101.50	12	1044.0	1043.3	1044.0 ^{KD}	1043.4	1044.0	0.01	0.00
R101.100	20	1637.7	1631.1	1634.0 ^{KD}	1633.1	1636.0	0.12	0.12
R102.25	7	547.1	546.3	547.1 ^{KD}	546.4	547.1	0.02	0.00
R102.50	11	909.0	909.0	909.0 ^{KD}	909.0	909.0	0.00	0.00
R102.100	18	1466.6	1466.6	1466.6 ^{KD}	1466.6	1466.6	0.00	0.00
R103.25	5	454.6	454.6	454.6 ^{KD}	454.6	454.6	0.00	0.00
R103.50	9	772.9	765.9	767.3 ^{CR}	769.3	769.3	0.44	0.26
R103.100	14	1208.7	1206.3	1206.3 ^{CR}	1206.9	1206.9	0.05	0.05
R104.25	4	416.9	416.9	416.9 ^{KD}	416.9	416.9	0.00	0.00
R104.50	6	625.4	616.5	620.7 ^{CR}	619.1	620.8	0.42	0.02
R104.100			949.1	950.9 ^{KL}	953.1	955.2	0.42	0.45
R105.25	6	530.5	530.5	530.5 ^{KD}	530.5	530.5	0.00	0.00
R105.50	9	899.3	892.1	893.6 ^{CR}	892.2	893.7	0.01	0.01
R105.100	15	1355.3	1346.1	1349.3 ^{CR}	1347.6	1350.1	0.11	0.06
R106.25	3	465.4	457.3	465.4 ^{KD}	457.3	465.4	0.00	0.00
R106.50	5	793.0	791.3	793.0 ^{KD}	791.4	793.0	0.01	0.00
R106.100	13	1234.6	1226.4	1227.4 ^{CR}	1227.0	1227.9	0.05	0.04
R107.25	4	424.3	422.9	424.3 ^{KD}	424.3	424.3	0.33	0.00
R107.50	7	711.1	704.4	705.8 ^{CR}	707.4	707.6	0.43	0.26
R107.100	11	1064.6	1051.8	1051.8 ^{KL}	1053.0	1054.3	0.11	0.24
R108.25	4	397.3	396.1	396.7 ^{KL}	396.9	**397.3**	0.20	0.15
R108.50	6	617.7	588.9	595.6 ^{KL}	594.7	596.4	0.98	0.13
R108.100			907.1	910.6 ^{KL}	913.6	921.7	0.72	1.22
R109.25	5	441.3	441.3	441.3 ^{KD}	441.3	441.3	0.00	0.00
R109.50	8	786.8	775.0	776.2 ^{CR}	775.4	776.7	0.05	0.06
R109.100	13	1146.9	1130.5	1133.1 ^{KL}	1134.3	1135.1	0.34	0.18
R110.25	4	444.1	437.3	437.9 ^{KL}	438.4	438.8	0.25	0.21
R110.50	7	697.0	692.5	694.1 ^{CR}	695.4	695.4	0.42	0.19
R110.100	12	1068.0	1048.4	1048.4 ^{KL}	1055.6	1056.0	0.69	0.72
R111.25	5	428.8	423.7	423.7 ^{KD}	427.3	**428.8**	0.85	1.20
R111.50	7	707.2	691.8	692.6 ^{CR}	696.3	696.6	0.65	0.58
R111.100	12	1048.7	1032.0	1032.0 ^{KL}	1034.8	1034.9	0.27	0.28
R112.25	4	393.0	384.2	384.4 ^{KD}	387.1	388.0	0.75	0.94
R112.50	6	630.2	607.2	612.3 ^{CR}	614.9	615.2	1.27	0.47
R112.100			919.1	922.3 ^{KL}	926.8	936.5	0.84	1.54

Table 8.1: Lower bound comparison - VRPTW class R1

Instance	K	IP_{opt}	LB_r	LB_r^c	LB_e	LB_e^c	G	G^c
RC101.25	4	461.1	406.6	461.1 ^{KD}	406.7	461.1	0.02	0.00
RC101.50	8	944.0	850.0	944.0 ^{CR}	944.0	**944.0**	11.05	0.00
RC101.100	15	1619.8	1584.0	1616.9 ^{CR}	1584.1	1607.4	0.01	-0.58
RC102.25	3	351.8	351.8	351.8 ^{KD}	351.8	351.8	0.00	0.00
RC102.50	7	822.5	719.9	813.0 ^{CR}	722.3	813.8	0.33	0.10
RC102.100	14	1457.4	1403.6	1403.6 ^{CR}	1406.3	1439.8	0.19	2.58
RC103.25	3	332.8	332.0	332.0 ^{KD}	332.8	**332.8**	0.24	0.24
RC103.50	6	710.9	643.1	710.1 ^{CR}	646.6	**710.9**	0.54	0.11
RC103.100	11	1258.0	1218.4	1218.4 ^{CR}	1225.6	1231.9	0.59	1.11
RC104.25	3	306.6	305.8	305.8 ^{KD}	306.6	**306.6**	0.26	0.26
RC104.50	5	545.8	543.7	543.7 ^{CR}	545.8	**545.8**	0.39	0.39
RC104.100	10	1132.3 ^{IV}	1094.3	1112.3 ^{KL}	1101.9	1102.4	0.69	-0.89
RC105.25	4	411.3	410.9	410.9 ^{KD}	411.3	**411.3**	0.10	0.10
RC105.50	8	855.3	754.4	853.6 ^{CR}	762.9	855.1	1.13	0.18
RC105.100	15	1513.7	1471.1	1509.7 ^{CR}	1472	1509.8	0.06	0.01
RC106.25	3	345.5	342.8	343.2 ^{KD}	345.5	**345.5**	0.79	0.67
RC106.50	6	723.2	664.4	716.5 ^{CR}	664.5	720.0	0.02	0.49
RC106.100	**13**	**1401.2**	1308.7	1332.5 ^{KL}	1318.8	1335.4	0.77	0.22
RC107.25	3	298.3	298.3	298.3 ^{KD}	298.3	298.3	0.00	0.00
RC107.50	6	642.7	591.4	631.5 ^{CR}	603.6	640.1	2.06	1.36
RC107.100	12	1207.8 ^{IV}	1170.6	1178.4 ^{KL}	1183.4	1179.5	1.09	0.09
RC108.25	3	294.5	293.7	294.5 ^{KD}	294.5	294.5	0.27	0.00
RC108.50	6	598.1	538.9	587.1 ^{CR}	544.9	596.6	1.11	1.62
RC108.100	11	1114.2 ^{IV}	1063.0	1091.5 ^{KL}	1073.5	1089.1	0.99	-0.22

Table 8.2: Lower bound comparison - VRPTW class RC1

Instance	K	IP_{opt}	LB_r	LB_e	G
C204.25	3	213.1	211.0^{KL}	213.1	1.00
C204.100	3	588.1^{IV}	581.1^{SA}	583.2	0.36
C205.50	5	359.8	359.0^{KL}	359.8	0.22
C206.50	5	359.8	359.0^{KL}	359.8	0.22
C207.25	3	214.5	214.4^{KL}	214.5	0.05

Table 8.3: Lower bound comparison - VRPTW class C2

Instance	K	IP_{opt}	LB_r	LB_r^c	LB_e	LB_e^c	G	G^c
R201.25	4	463.3	460.1	460.1	460.1	460.1	0.00	0.00
R201.50	6	791.9	788.4	791.9	791.9	791.9	0.44	0.00
R201.100	8	1143.2	1136.2	1138.6	1140.1	1141.2	0.34	0.23
R202.25	4	410.5	406.3	408.3	408.6	**410.5**	0.57	0.54
R202.50	5	698.5	692.7	696.5	695.1	697.1	0.35	0.09
R202.100			1009.8	1009.8	1011.2	1011.5	0.14	0.17
R203.25	3	391.4	379.8	381.6	391.4	**391.4**	3.05	2.57
R203.50	5	605.3	590.9	593.4	599.1	600.6	1.39	1.21
R203.100			846.4	847	885.1	899.5	4.57	6.20
R204.25	2	355.0	333.0	335.3	352.0	352.0	5.71	4.98
R204.50			474.5	482.3	505.3	505.7	6.49	4.85
R205.25	3	393.0	381.2	388.4	390.6	390.6	2.47	0.57
R205.50	4	690.1	666.6	672.3	682.9	683.5	2.45	1.67
R205.100			916.9	923				
R206.25	3	374.4	363.1	365.9	373.6	373.6	2.89	2.10
R206.50	4	632.4	609.5	611.3	626.4	627.1	2.77	2.58
R206.100			835.3	840.7				
R207.25	3	361.6	347.5	349.7	361.0	361.0	3.88	3.23
R207.50			539.0	544.3	564.1	566.2	4.66	4.02
R208.25	1	328.2	318.1	318.9	328.2	**328.2**	3.18	2.92
R208.50			462.4	471.5	485.7	486.1	5.04	3.10
R209.25	2	370.7	353.8	358.3	364.2	364.2	2.94	1.65
R209.50	4	600.6	582.9	588.4	598.5	599.9	2.68	1.95
R209.100			819.8	823.7				
R210.25	3	404.6	395.8	397.9	404.6	**404.6**	2.22	1.68
R210.50	4	645.6	624.4	624.4	636.1	639.8	1.87	2.47
R210.100			849.4	855.8				
R211.25	2	350.9	330.1	330.4	341.4	345.7	3.42	4.63
R211.50	3	535.5	507.9	512.5	528.7	529.9	4.10	3.40
R211.100			705.8	710.7				

Table 8.4: Lower bound comparison - VRPTW class R2

Instance	K	IP_{opt}	LB_r	LB_r^c	LB_e	LB_e^c	G	G^c
RC201.25	3	360.2	356.6	360.2	360.2	**360.2**	1.01	0.00
RC201.50	5	684.8	670.1	681.9	684.8	**684.8**	2.19	0.43
RC201.100	9	1261.8	1240.3	1253.4	1256.0	1257.4	1.27	0.32
RC202.25	3	338.0	290.4	313.3	338.0	**338.0**	16.39	7.88
RC202.50	5	613.6	504.1	548.2	613.6	**613.6**	21.72	11.93
RC202.100	8	1092.3	1004.3	1013.8	1088.1	1089.5	8.34	7.47
RC203.25	3	326.9	214.4	260.8	326.9	**326.9**	52.47	25.35
RC203.50	4	555.3	409.2	480.0	555.3	**555.3**	35.70	15.69
RC203.100			815.2	831.9				
RC204.25	3	299.7	188.5	244.8	299.7	**299.7**	58.99	22.43
RC205.25	3	338.0	307.6	320.7	338.0	**338.0**	9.88	5.39
RC205.50	5	630.2	541.5	579.9	630.2	**630.2**	16.38	8.67
RC205.100	7	1154.0	1056.1	1070.8	1147.7	1149.2	8.67	7.32
RC206.25	3	324.0	250.1	288.9	324.0	**324.0**	29.55	12.15
RC206.50	5	610.0	441.3	532.1	610.0	**610.0**	38.23	14.64
RC206.100			952.4	982.8				
RC207.25	3	298.3	217.9	263.8	298.3	**298.3**	36.90	13.08
RC207.50	4	558.6	390.8	468.8	558.6	**558.6**	42.94	19.16
RC207.100			866.6	877.8				
RC208.25	2	269.1	169.6	233.0	269.1	**269.1**	58.67	15.49
RC208.50	**3**	**476.7**			470.4	473.2		

Table 8.5: Lower bound comparison - VRPTW class RC2

Instance	Solution		Relaxed Pricing		Exact Pricing		
	K	IP	$Nodes$	$T(s)$	$Nodes$	$T(s)$	$G(\%)$
R101.25	8	617.1	1	0.02 ^{KD}	1	0.11	0.00
R101.50	12	1044.0	1	0.07 ^{KD}	1	0.53	0.00
R101.100	20	1637.7	17	2.15 ^{KD}	23	14.78	0.00
R102.25	7	547.1	1	0.04 ^{KD}	1	0.14	0.00
R102.50	11	909.0	1	0.26 ^{KD}	1	1.13	0.00
R102.100	18	1466.6	1	7.20 ^{KL}	1	27.49	0.00
R103.25	5	454.6	1	0.06 ^{KD}	1	0.34	0.00
R103.50	9	772.9	41	4.04 ^{KD}	11	11.37	0.00
R103.100	14	1208.7	39	118.66 ^{CR}	33	256.60	0.00
R104.25	4	416.9	1	0.09 ^{KD}	1	0.43	0.00
R104.50	6	625.4	74	102.86 ^{CR}	13	187.59	0.00
R104.100	11	971.5	5396	88474.98 ^{IV}	33		0.97
R105.25	6	530.5	1	0.02 ^{KD}	1	0.16	0.00
R105.50	9	899.3	13	0.46 ^{KD}	5	2.43	0.00
R105.100	11	1355.3	78	27.03 ^{KD}	35	49.17	0.00
R106.25	6	465.4	1	0.09 ^{KD}	1	0.40	0.00
R106.50	9	793.0	1	0.61 ^{KD}	1	3.70	0.00
R106.100	15	1234.6	760	2236.03 ^{CR}	217	555.03	0.00
R107.25	4	424.3	1	0.10 ^{KD}	1	0.38	0.00
R107.50	7	711.1	65	6.72 ^{KD}	21	26.14	0.00
R107.100	11	1064.6			283		0.23
R108.25	4	397.3	3	1.30 ^{KD}	1	1.23	0.00
R108.50	6	617.7			41		1.29
R108.100	9	(960.88)			11		19.57
R109.25	5	444.3	1	0.03 ^{KD}	1	0.20	0.00
R109.50	8	786.8	189	10.19 ^{KD}	7	25.43	0.00
R109.100	13	1146.9	4005	53009.90 ^{KD}	3305		0.03
R110.25	4	444.1	25	0.41 ^{KD}	11	2.13	0.00
R110.50	7	697.0	5	1.48 ^{KD}	13	9.28	0.00
R110.100	12	1068.0			2307		0.78
R111.25	4	428.8	3	0.14 ^{KD}	1	0.88	0.00
R111.50	6	707.2	199	76.38 ^{CR}	99	94.79	0.00
R111.100		1048.7			1154		1.48
R112.25	4	393.0	9	1.02 ^{KD}	9	8.25	0.00
R112.50	6	630.2	5263	4749.60 ^{KL}	711		0.32
R112.100	**9**	**(973.6)**			19		4.71

Table 8.6: Computational results - VRPTW class R1

Instance	Solution		Relaxed Pricing		Exact Pricing		
	K	IP	$Nodes$	$T(s)$	$Nodes$	$T(s)$	$G(\%)$
RC101.25	4	461.1	1	0.05 ^{KD}	1	0.38	0.00
RC101.50	8	944.0	3	0.78 ^{KD}	1	2.87	0.00
RC101.100	15	1619.8	11	9.70 ^{KD}	239	145.45	0.00
RC102.25	3	351.8	1	0.04 ^{KD}	1	0.64	0.00
RC102.50	7	822.5	507	46.98 ^{KD}	159	41.39	0.00
RC102.100	14	1457.4			5815		1.67
RC103.25	3	332.8	3	0.28 ^{KD}	1	0.73	0.00
RC103.50	6	710.9	3	1.77 ^{KD}	1	17.32	0.00
RC103.100	11	1258.0			1397		0.72
RC104.25	3	306.6	7	0.48 ^{KD}	1	0.46	0.00
RC104.50	5	545.8	17	9.42 ^{CR}	1	38.79	0.00
RC104.100	10	1132.3	6757	325646.97 ^{IV}	7		5.13
RC105.25	4	411.3	3	0.22 ^{KD}	1	0.32	0.00
RC105.50	8	855.3	16	2.87 ^{KD}	3	6.82	0.00
RC105.100	15	1513.7	37	19.46 ^{KD}	9	69.26	0.00
RC106.25	3	345.5	15	0.54 ^{KD}	1	0.43	0.00
RC106.50	6	855.3	21	3.78 ^{KD}	7	11.72	0.00
RC106.100	**13**	**1401.2**			**14451**	**107194.07**	0.00
RC107.25	3	298.3	1	0.05 ^{KD}	1	0.27	0.00
RC107.50	6	723.2	71	17.58 ^{KD}	11	47.45	0.00
RC107.100	12	1207.8	1493	14114.33 ^{IV}	1319		0.01
RC108.25	3	294.5	1	0.30 ^{KD}	1	0.80	0.00
RC108.50	6	598.1	8	9.08 ^{CR}	3	829.24	0.00
RC108.100	11	1114.2	707	23516.79 ^{IV}	17		2.74

Table 8.7: Computational results - VRPTW class RC1

Instance	Solution		Relaxed Pricing		Exact Pricing		
	K	IP	$Nodes$	$T(s)$	$Nodes$	$T(s)$	$G(\%)$
R201.25	4	463.3	3	0.30 ^{KL}	3	1.13	0.00
R201.50	6	791.9	1	1.2 ^{KL}	1	16.25	0.00
R201.100	8	1143.2	183	380.10 ^{KL}	23	837.31	0.00
R202.25	4	410.5	5	1.2 ^{KL}	1	1.32	0.00
R202.50	5	698.5	11	17.1 ^{KL}	7	19.55	0.00
R203.25	3	391.4	37	8.1 ^{KL}	1	2.24	0.00
R203.50	5	605.3	19	71.64 ^{IV}	75	1602.00	0.00
R204.25	2	355.0	35	40.62 ^{IV}	19	48.74	0.00
R204.50	2	506.4	132	7837.34 ^{IV}	23		0.51
R205.25	3	393.0	15	1.65 ^{KL}	5	4.32	0.00
R205.50	4	690.1	133	193.28 ^{IV}	123	519.72	0.00
R206.25	3	374.4	85	21.3 ^{KL}	5	3.74	0.00
R206.50	4	632.4	1615	7410.25 ^{IV}	59	4068.00	0.00
R207.25	3	361.6	125	45.3 ^{KL}	9	15.64	0.00
R207.50	4	(584.6)			13		3.63
R208.25	1	328.2	13	106.12 ^{IV}	1	121.50	0.00
R209.25	2	370.7	75	87.3 ^{KL}	9	12.70	0.00
R209.50	4	600.6	6	47.0 ^{IV}	5	733.55	0.00
R210.25	3	404.6	65	9.75 ^{KL}	1	4.52	0.00
R210.50	4	645.6	525	795.6 ^{KL}	99		2.14
R211.25	2	350.9	1513	772.95 ^{KL}	73	5188.00	0.00
R211.50	3	535.5	1972	7036.6 ^{IV}	5		19.51

Table 8.8: Computational results - VRPTW class R2

Instance	Solution		Relaxed Pricing		Exact Pricing		
	K	IP	$Nodes$	$T(s)$	$Nodes$	$T(s)$	$G(\%)$
RC201.25	3	360.2	1	0.30 ^{KL}	1	0.40	0.00
RC201.50	5	684.8	15	4.50 ^{KL}	1	10.03	0.00
RC201.100	9	1261.8	679	1808.85 ^{KL}	65	3518.9	0.00
RC202.25	3	338.0	665	333.45 ^{KL}	1	1.70	0.00
RC202.50	5	613.6	29	79.73 ^{IV}	1	6.67	0.00
RC202.100	8	1092.3	239	40925.94 ^{IV}	33	5983.63	0.00
RC203.25	3	326.9	379	619.38 ^{IV}	1	2.59	0.00
RC203.50	4	555.3	38	17895.64 ^{IV}	1	856.56	0.00
RC204.25	**3**	**299.7**			**1**	**2.48***	0.00
RC205.25	3	338.0	23	10.35 ^{KL}	1	0.64	0.00
RC205.50	5	630.2	5	17.36 ^{IV}	1	5.03	0.00
RC205.100	7	1154.0	65	4387.65 ^{IV}	73	2770.48	0.00
RC206.25	3	324.0	503	293.25 ^{KL}	1	1.61	0.00
RC206.50	5	610.0	62	154.80 ^{IV}	1	50.80	0.00
RC207.25	**3**	**298.3**			**1**	**3.39***	0.00
RC207.50	**4**	**558.6**			**1**	**223.67***	0.00
RC208.25	**2**	**269.1**			**1**	**123.65***	0.00
RC208.50	**3**	**476.7**			**21**	**26412.50**	0.00

Table 8.9: Computational results - VRPTW class RC2

$k \leq 6$ have been used (see Cook and Rich [90]). Best lower bounds have been reported by Kallehauge et al. [5] for intstances rc104.100 and r 108.100 It can be seen that the use of elementary paths and 2-cuts allows to solve to optimality eleven instances at the root node of the search tree. 100 customers instances rc104, rc107 and rc108 have been solved by Irnich and Villeneuve [83] but the authors did not report the lower bounds at the root node for those instances. Instead I could solve rc106 (removing the time limitation) in 107194.07 seconds for the first time.

Table 8.3 reports only instances for which the relaxed pricing algorithm fails to compute the optimal solution at the root node of the search tree, other instances are not reported since the comparison is not significant. The table shows that the use of elementary paths allows to solve all instances at the root node of the search tree except instance c204.100 that was not solved within the time limit.

Tables 8.4 and 8.5 show that the lower bound improvement is greater for instances of set 2. It varies from 0% to 17.57% for *r*-instances and from 1% to 65.09% for *rc*-instances. 5 *r*-instances and 14 *rc*-instances have been solved at the root node. For unreported instances no valid lower bounds can be computed within the time limit by the exact pricing algorithm and in some cases, by both exact pricing and relaxed pricing algorithms. Thus the comparison cannot be performed.

8.2.2 Search tree and time comparison

For the experimental comparison of the overall Branch-and-Price algorithms I report the search tree size and the overall computing time multiplied by a factor 0.07 for Kohl et al. [69], by 0.16 for Cook and Rich [90], by 0.33 for Irnich and Villeneuve [83] and the duality gap at the end of the computation for the exact pricing. The multiplication factors have been obtained comparing different computers performance using the Linpack benchmark, available at http://performance.netlib.org. Since the comparison between the Sun Fire 15K system and a standard Pentium IV is not available the computing times of the results presented by Kallehauge et al. [5] are roughly multiplied by 1.5 that is a lower bound on the ratio of the performances of the Sun Fire 15K against those of a Pentium IV. The ratio 1.5 has been obtained comparing the MFLOPS of a UltraSPARC III (the core of the Sun Fire 15K system) and the MFLOPS of a Pentium IV. It is clear that the overall performaces of a system not only depend on the processor speed.

Tables from 8.6 to 8.9 show that the search tree was always smaller when elementary paths were used except for instances r110.50, rc101.100 and r203.50 for which Kohl et al. [69] and Irnich and Villeneuve [83] were able to perform very effective branching decisions.

Tables 8.6 and 8.7 show that for set 1 instances the algorithm based on elementary paths is not competitive with the algorithms based on relaxed pricing except for a few cases. Instance R106.100 has been solved in a less than a quarter of time used by Cook and Rich [90]. Instance R107.100 were solved in 8120.34 seconds, instance R109.100 were solved in 7635.13 seconds and instance R110.100 in 17574.84 seconds by removing the time limit. The value 973.6 for instance R112.100 is the new best known obtained with the exact pricing algorithm. The exact pricing provided a new optimal solution for instance RC106.100.

Table 8.8 does not show any clear domination between the two algorithms. The algorithm based on exact pricing outperforms the one based on relaxed pricing when the lower bound improvement shown in previous tables is significant. By contrast some instances that seem to be quite easy for the relaxed pricing algorithm were not solved by the exact pricing one.

Table 8.9 reports on computational results that are favourable to exact pricing. Five new instances have been solved by exact pricing. Instances marked with an asterisk have been solved also by Chabrier [1] who proposed an algorithm based on the computation of elementary paths similar to the one proposed in this thesis. Instance rc208.50 has been solved for the first time. It should be noticed that several instances have been solved at the root node of the search tree.

Conclusions The computational experience shows that column generation coupled with the exact solution of the pricing problem allows to compute better lower bounds and reduce the size of the search tree. The overall performances of algorithms based on relaxed or exact pricing are comparable. It can be noticed (and it is well known in literature, see Cordeau et al. [44]) that branching decisions do not help to increase significantly the lower bounds (see the search tree size comparison) and then it can be stated that the most part of the computation should be used to increase as much as possible the lower bound in the early nodes of the search tree by the use of elementary paths and valid cuts. Future research should be devoted in this direction following the approach of Fukasawa et al. [79] for the CVRP.

Chapter 9

Conclusions

In this thesis I have presented an algorithmic approach based on the Branch-and-Price framework for the solution of vehicle routing problems with additional constraints. Such approach has been applied to the capacitated VRP and to two of its variations: the VRP with Delivery and Collection and the VRP with Time Windows.

The Branch-and-Price algorithm presented is based on the linear relaxation of a Set Covering reformulation of the VRP, solved via column generation, where the pricing problem is the Resource Constrained Elementary Shortest Path Problem. The most successful approaches for the RCESPP are based on dynamic programming. In this thesis I have proposed some new algorithmic ideas to improve the dynamic programming algorithms for the solution of the pricing problem: in particular I have presented exact algorithms for the RCESPP based on bi-directional and bounded dynamic programming. I have also proposed a new dynamic programming algorithm for the solution of the RCESPP, called *Decremental State Space Relaxation*, which is a development of the state space relaxation concept proposed by Christofides et al. [67].

After presenting these algorithms I investigated the advantages and drawbacks of two possible approaches: to solve the pricing problem to optimality or to solve a relaxation, namely the RCSPP, where cycles are allowed.

I performed an exhaustive computational experience on both methods for the pricing problem and on the resulting branch-and-price algorithms for the solution of the CVRP, the VRPDC and the CVPTW with and without the use of cutting planes to strengthen the formulation .

Several authors used both the relaxed pricing approach (see for instance Irnich and Villeneuve [83]) and the exact pricing approach (see for instance Chabrier [1]) but, at the best of my knowledge, this is the first attempt to compare the two approaches in terms of lower bounds and overall performances on a variety of

VRP problems. The algorithmic ideas presented in this work are useful to improve dynamic programming algorithms for both the RCESPP and the RCSPP. It can be seen in chapter 4 that the bidirectional bounded algorithm outperforms the mono-directional one. The average computing time reduction is about the 30% but in some cases the computing time has been reduced by one order of magnitude.

The main conclusion that arises from this research is that better results can be obtained through the improvement of the lower bounds in the former levels of the search tree.

I focused my attention on the lower bound increase obtained by the generation of columns corresponding to elementary paths. Other approaches in the literature are based on the computation of cutting planes (see Fukasawa et al. [79]). The reported experiments show that both approaches are useful. Experiments on the VRPDC (see chapter 7) and on the CVRTW (see chapter 8) showed that the use of elementary paths gives better results while experiments on the CVRP (see chapter 6) showed the exact pricing algorithm is useful to increase the lower bounds but the use of cutting planes is faster. The main reason for that behaviour is that for more constrained problems (like VRPDC and CVPTW) the branching decisions allow to obtain children problems that are easier than the parent problem and then the computation of elementary paths is easier. Thus, since the lower bound increase given by the exact pricing is significant, the overall branch-and-price is faster. On the contrary for less constrained problems, like the CVRP, the former nodes of the search tree are more difficult for the exact pricing algorithm and thus the overall branch-and-price is slower.

This also indicates new directions for future research. It has been shown that the computation of elementary paths is useful but time consuming. A new compromise between relaxed pricing and exact pricing should be explored, not only in the direction of forbidding cycles of order k as typically done in the literature but also avoiding the cycling over a subset of critical nodes. It should be pointed out that the decremental space relaxation algorithm presented in chapter 4 can be used in this direction. The balance between the speed of the algorithm versus the accuracy can be tuned stopping the decrement of the state space relaxation to a fixed number of critical nodes. The lower is the number of critical nodes the faster and less accurate is the algorithm.

Bibliography

[1] Chabrier A. Vehicle routing problem with elementary shortesp path based column generation. *Computers and Opeerations Research*, to appear, 2005.

[2] Mingozzi A., Giorgi S., and Baldacci R. An exact method for the vehicle routing problem with backhauls. *Transportation Science*, 33(3):315 – 329, 1999.

[3] Farley A.A. A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research*, 38, 1990.

[4] Gendreau M .and Laporte G. and Potvin J.Y. Metaheuristics for the capacitated vrp. In Toth P. and Vigo D., editors, *The vehicle routing problem*, chapter 6, pages 129–149. SIAM monograph on discrete mathematics and applications, 2002.

[5] Kallehague B., Larsen J., and Madsen O.B.G. Lagrangean duality applied on vehicle routing with time windows. *Computers and Opeerations Research*, to appear.

[6] Ribeiro C. and Soumis F. A column gneration approach to the multiple-depot vehicle scheduling problem. *Operations Research*, 142:41–52, 1994.

[7] Hoong C.L. and Liang Z. Pickup and delivery with time windows: Algorithms and test case generation. In *Proceedings of the 13th International Conference on Tools with Artificial Intelligence*, pages 333 – 340. IEEE, 2001.

[8] Feillet D., Dejax P., Gendreau M., and Gueguen C. An exact algorithm for the elementary shortest path with resource constraints: application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

[9] Naddef D. and Rinaldi G. Branch-and-cut algorithms for the capacitated vrp. In Toth P. and Vigo D., editors, *The vehicle routing problem*, chapter 3, pages 53–81. SIAM monograph on discrete mathematics and applications, 2002.

[10] Jan Dethloff. Vehicle routing and reverse logistics: the vehicle routing problem with simultaneous delivery and pick-up. *OR Spektrum*, 23:79 – 96, 2001.

[11] Ryan D.M. and Foster B.A. An integer programming approach to scheduling. In Wren A., editor, *Computer Scheduling of Public Transportation Urban Passenger and Crew Scheduling*, pages 269–280. North-Holland, 1981.

[12] du Merle O., Villeneuve D., Desrosiers J., and Hansen P. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.

[13] Angelelli E. and Mansini R. The vehicle routing problem with time windows and simultaneous pick-up and delivery. In Klose A., Speranza M. G., and Van Wassenhove L. N., editors, *Quantitative Approaches to Distribution Logistics and Supply Chain Management*. Springer-Verlag, to appear.

[14] Hadjicostantinou E., Christofides N., and Mingozzi A. A new exact algorithm for the vehicle routing problem based on q-paths and k-shortest paths relaxations. *Annals of Operations Research*, 61:21–43, 1995.

[15] Lübbecke M. E. and Desrosiers J. Selected topics in column generation. *Operations Research*, revised 2004, to appear.

[16] Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., and Shmoys D.B., editors. *The Traveling Salesman Problem*. Wiley, Chichester, UK, 1985.

[17] Vanderbeck F. *Decomposition and Column Generation for Integer Programs*. PhD thesis, Université catholique de Louvain, 1994.

[18] Vanderbeck F. and Wolsey L.A. An exact algorithm for ip column generation. *Operations Research Letters*, 19:151–159, 1996.

[19] Carpaneto G., Dell'Amico, Fischetti M., and Toth P. A branch-and-bound algorithm for the multiple depot vehicle routing problem. *Networks*, 19:531–548, 1989.

[20] Clarke G. and Wright J. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[21] Cornuejols G. and Harche F. Polyhedral study of the vehicle routing problem. *Mathematical Programming*, 60:21–52, 19993.

[22] Crainic T. G. and Laporte G., editors. *Fleet management and logistics*. Kluwer, Boston, Ma, 1998.

[23] Desaulniers G., Desrosiers J., Ioachim I., Solomon M., Soumis F., and Villeneuve D. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In Crainic T. and Laporte G, editors, *Fleet Management and Logistics*, pages 57–93. Kluwer Academic, 1998.

[24] Desaulniers G., Lavigne J., and Soumis F. Multi-depot vehicle scheduling problems with time windows and waiting costs. *European Journal of Operational Research*, 111(3):479 – 494, 1998.

[25] Desrochers G. An algorithm for the shortest path problem with resource constraints. *Les cahiers du GERAD*, G-88-27, 1988.

[26] Laporte G. The vehicle routing problem: an overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[27] Laporte G. and Semet F. Classical heuristics for the capacitated vrp. In Toth P. and Vigo D., editors, *The vehicle routing problem*, chapter 5, pages 109–126. SIAM monograph on discrete mathematics and applications, 2002.

[28] Laporte G., Mercure H., and Norbert Y. A branch-and-bound algorithm for a class of asymmetrical vehicle routing problem. *Journal of Operational Research Society*, 43:469–481, 1992.

[29] Laporte G. and Norbert Y. Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, 31:147–184, 1987.

[30] Laporte G., Norbert Y., and Desrochers M. Optimal routing under capacity and distance restrictions. *Operations Research*, 33:1050–1073, 1985.

[31] Dantzig G.B. and Ramser J.H. The truck dispathcing problem. *Management Science*, 6:80–91, 1959.

[32] Dantzig G.B. and Wolfe P. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[33] Nemhauser G.L. and Wolsey L.A., editors. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, N.Y., 1988.

[34] Handler G.Y. and Zang I. A dual algorithm for the constrained shortest path problem. *Networks*, 10:293–310, 1980.

[35] Ben Amor H. *Stabilization de l-Algorithme de Génération de Colonnes*. PhD thesis, École Polytechnique de Montréal, 2002.

[36] Bramel J. and Simchi-Levi D. Set-covering based algorithms for the capacitated vrp. In Toth P. and Vigo D., editors, *The vehicle routing problem*, chapter 4, pages 85–106. SIAM monograph on discrete mathematics and applications, 2002.

[37] Desrosiers J., Soumis F., and Desrochers M. Routing with time-windows by column generation. *Networks*, 14:545–565, 1984.

[38] Desrosiers J., Dumas Y., Solomon M., and Soumis F. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Routing*, volume 8 of *Hanbooks in Operations Research and Management Science*, chapter 1, pages 1–33. Elsevier, 1995.

[39] Desrosiers J., Dumas Y., Solomon M., and Soumis F. Daily aircraft routing and scheduling. *Management Science*, 43:841–855, 1997.

[40] Lysgaard J. Cvrpsep: A package of separation routines for the capacitated vehicle routing problem. *Available at www.asb.dk/ lys*, 2003.

[41] Lysgaard J., Letchford A., and Eglese R. A new branch and cut algorithm for the vehicle routing problem. *Mathematical Programming*, to appear.

[42] Lysgaard J., Letchford A.N., , and Eglese R.W. A new branch-and-cut algorithm for capacitated vehicle routing problems. *Mathematical Programming A*, 100:423–445, 2004.

[43] Beasley J.E. and Christofides N. An algorithm for the resource constrained shortest path problem. *Networks*, 19:379–394, 1989.

[44] Cordeau J.F., Desaulniers G., Desrosiers J., Solomon M.M., and Soumis F. Vrp with time windows. In Toth P. and Vigo D., editors, *The vehicle routing problem*, chapter 7, pages 157–193. SIAM monograph on discrete mathematics and applications, 2002.

[45] Rousseau L.M., Gendreau M., and Feillet D. Interior point stabilization for column generation. Technical Report PO2003-39-X, CRT - Centre de recherche sur les transports, 2003.

[46] Dell'Amico M., Righini G., and Salani M. Exact solution of the vehicle routing problem with simultaneous pick-up and delivery. *Transportation Science*, (to appear).

[47] Desrochers M. and Soumis F. A reoptimization algorithm for the shortest path problem with time windows. *European Journal of Operational Research*, 35:242–254, 1988.

[48] Desrochers M. and Soumis F. A column generation approach to the transit crew scheduling problem. *Transportation Science*, 23:1–13, 1989.

[49] Desrochers M., Desrosiers J., and Solomon M. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342–354, 1992.

[50] Desrochers M., Lenstra J.K., and Savelsbergh M.W.P. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research*, 46:322–332, 1990.

[51] Dror M. Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42:977–978, 1994.

[52] Dror M., Laporte G., and Trudeau P. Vehicle routing with split deliveries. *Discrete and Applied Mathematics*, 50:239–254, 1994.

[53] Fischetti M., Toth P., and Vigo D. A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graph. *Operations Research*, 42:846–859, 1994.

[54] Fisher M. Optimal solution of the vehicle routing problems using minimum k-trees. *Operations Research*, 42:626–642, 1994.

[55] Fisher M. Vehicle routing. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Routing*, volume 8 of *Hanbooks in Operations Research and Management Science*, chapter 1, pages 1–33. Elsevier, 1995.

[56] Gamache M., Soumis F., Marquis G., and Desrosiers J. A column generation approach for large-scale aircrew rostering problems. *Operations Research*, 48(2):247–263, 1992.

[57] Goetschalckx M. and Jacobs-Blecha C. The vehicle routing problem with backhauls. *European Journal of Operational Research*, 42(1):39 – 51, 1989.

[58] Jünger M., Reinelt G., and Rinaldi G. The travelling salesman problem. In Ball M. O., Magnante T.L., Monma C.L., and Nemhauser G. L., editors, *Handbooks in Operations Research and Management Science 7: Network Models*, pages 225–330. North-Holland, 1995.

[59] Sigurd M., Pisinger D., and Sig M. The pickup and delivery problem with time windows and precedences. *Transportation Science*, 38:197–209, 2004.

[60] Sol M. *Column Generation Techniques for Pickup and delivery Problems*. PhD thesis, Technical University of Eindhoven, 1994.

[61] Solomon M.M. *Vehicle Routing and Scheduling with Time Windows Constraints: Models and Algorithms*. PhD thesis, University of Pennsylvania, 1983.

[62] Solomon M.M. and Desrosiers J. Time window constrained routing and scheduling problems. *Transportation Science*, 22:1–13, 1988.

[63] Ball M.O., Magnanti T.L., Monma C.L., and Nemhauser G.L., editors. *Network Routing*. Elsevier, 1995.

[64] Savelsbergh M.W.P. and Sol M. The general pickup and delivery problem. *Transportation Science*, 29:17–29, 1995.

[65] Bianchessi N. and Righini G. Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. *Computers and Opeerations Research*, to appear.

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

[67] Christofides N., Mingozzi A., and Toth P. State-space relaxation of bounds to routing problems. *Networks*, 11:145–164, 1981.

[68] Christofides N. and Elion S. An algorithms for the vehicle dispatching problem. *Operational Research Quarterly*, 20:309–318, 1969.

[69] Kohl N., Desrosiers J., Madsen O.B.G., Solomon M.M., and Soumis F. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116, 1999.

[70] Augerat P., Belenguer J.M., Benavent E., Coberán A., Naddef D., and Rinaldi G. Computational results with a branch and cut code for the capacitated vehicle routing problem. Technical Report Tecnical Report RR 949-M, Université Joseph Fourier, Grenoble, 1995.

[71] Toth P. and Vigo D. An exact algorithm for the vehicle routing problem with backhauls. *Transportation Science*, 31:372–385, 1997.

[72] Toth P. and Vigo D. A heuristic algorithm for the symmetric and asymmetric vehicle routing problem with backhauls. *European Journal of Operational Research*, 113:528–543, 1999.

[73] Toth P. and Vigo D. Branch-and-bound algorithms for the capacitated vrp. In Toth P. and Vigo D., editors, *The vehicle routing problem*, chapter 2, pages 29–49. SIAM monograph on discrete mathematics and applications, 2002.

[74] Toth P. and Vigo D. An overview of vehicle routing problems. In Toth P. and Vigo D., editors, *The vehicle routing problem*, chapter 1, pages 1–26. SIAM monograph on discrete mathematics and applications, 2002.

[75] Toth P. and Vigo D., editors. *The Vehicle Routing Problem.* SIAM monograph on discrete mathematics and applications, 2002.

[76] Gilmore P.C. and Gomory R.E. A linear programming approach to the cutting stock problem. *Operations Research*, 11:849–859, 1961.

[77] Vance P.H., Barnhart E.L., Johnson E.L., and Nemhauser G.L. Solving binary cutting stock problem by conlumn generation and branch-and-bound. *Computational Optimization and Applications*, 3:111–130, 1994.

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

[79] Fukasawa R., Lysgaard J., de Aragao M.P., Reis M., Uchoa E., and Werneck R.F. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. In D. Bienstock and G. Nemhauser, editors, *IPCO 2004*, pages 1–15. Springer-Verlag, 2004.

[80] Martsen R.E. The use of boxstep method in discrete optimization. *Math. programming Stud.*, 3:127–144, 1975.

[81] Ahuja R.K., Magnanti T.L., and Orlin J.B. *Network flows.* Prentice Hall, 1993.

[82] Gélinas S., Desrochers M., Desrosiers J., and Solomon M.M. A new branching strategy for time constrained routing problems with application to backhauling. *Annals of Operations Research*, 61:91–109, 1995.

[83] Irnich S. and Villeneuve D. The shortest path problem wiht resource constraints and k-cycle elimination for $k \geq 3$. Technical Report G-2003-55, Les cahiers du GERAD, 2003.

[84] Irnich S. and Desaulniers G. Shortest path problems with resource constraints. Technical Report G-2004-11, Les cahiers du GERAD, 2004.

[85] Kim S., Chang K.N., and Lee J.Y. A descent method with linear programming subproblems for nondifferentiable convex optimization. *Mathematical Programming*, 71:17–28, 1995.

[86] Martello S. and Toth P., editors. *Knapsack problems: Algorithms and Computer Implementations*. Wiley, Chichester, UK, 1990.

[87] Thangiah S.R., Potvin J., and Sun T. Heuristic approaches to vehicle routing with backhauls and time windows. *Computers and Opeerations Research*, 23(11):1043 – 1057, 1996.

[88] Ralphs T.K., Kopman L., Pulleyblank W.R., and Trotter L.E. On the capacitated vehicle routing problem. *Mathematical Programming B*, 94, 2003.

[89] Blasum U. and Hochstattler. Application of the branch-and-cut framework to vehicle routing problem. Technical Report ZPR2000-386, ZPR, University of Cologne, 2000.

[90] Cook W. and Rich J. A parallel cutting-plane algorithm for the vehicle routing problem with time windows. Technical Report TR99-04, Computational and Applied Mathematics - Rice University, 1999.

[91] Nanry W.P. and Barnes J.W. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B*, 34:107 – 121, 2000.

[92] Dumas Y., Desrosiers J., Gélinas E., and Solomon M. M. An optimal algorithm for the travelling salesman problem with time windows. *Operations Research*, 42:626–642, 1994.