Università degli Studi di Milano

Facoltà di Scienze Matematiche, Fisiche e Naturali

Dipartimento di Matematica

Dottorato di ricerca in Matematica e Statistica per le Scienze Computazionali, ciclo XXIII

Settore scientifico/disciplinare MAT-09

Mathematical Programming Algorithms for Transportation Problems

PhD Thesis of:

Andrea Bettinelli

Tutor: Prof. Giovanni Righini

Co-tutor: Prof. Alberto Ceselli

Coordinator: Prof. Vincenzo Capasso

A.A. 2009/2010

To my tutor Giovanni Righini and my co-tutor Alberto Ceselli,
whose support and guidance have been fundamental in this work.

To my parents Osvaldo and Luisa and my sister Livia,
who have always provided unwavering love and encouragement.

To Valentina
for everything, from technical to emotional support,
for her understanding, endless patience and encouragement
when it was most required.

Thank you.

Contents

Preface

The thesis deals with the study of transportation problems, and in particular focuses on developing new exact and heuristic algorithms for obtaining effective solutions to interesting variants of the very well known Vehicle Routing Problem and Pickup and Delivery Problem. The studied problems consider additional real-world requirements, often neglected in the literature. They lead to more involved problems but on the other hand more realistic ones, that call for powerful optimization methods in order to tackle such difficult applications. The proposed algorithms are based on mathematical programming techniques, such as branch-and-price, column generation and dynamic programming. The performance of the algorithms is analyzed with extensive computational experiments and compared with the most effective algorithms from the literature, showing the usefulness of the proposed methods.

Chapter 1

Introduction

1.1 Transportation Problems

It is needless to underline the importance of developing efficient algorithmic techniques for optimization in transportation. Indeed transportation is undoubtedly one of the most critical elements in managing the global supply chain, because it represents a relevant factor in the final cost of goods. The study of transportation problem is a central topic in operations research and management science, and it represents one of the most successful applications of operations research solution methods. The complexity of the problems arising in this context and the possibility to achieve substantial economic savings calls for powerful optimization and mathematical programming techniques. In this thesis, we study transportation problems and propose advanced optimization algorithms in order to find optimal and heuristic solutions. The Vehicle Routing Problem can be considered as the basis of transportation problems. Many works from the literature study how to solve different applications of the Vehicle Routing Problem (see e.g. the survey by Golden et al. [46]). This problem requires to find a set of routes of minimum cost (including travel cost, travel time and fixed cost for the use of the vehicles) for a fleet of identical vehicles with limited service capacity, with the aim of serving a given set of customers. Routes start and end at a unique

depot and each customer has a given demand which must be served by a single vehicle of the fleet. However real-world applications often require to take into account several additional constraints [24]: for instance, visiting certain locations is allowed only in certain time windows; furthermore, large companies usually have more than one depot and they often manage a fleet made of vehicles with different capacities and operating costs. This motivates the study of optimization techniques for transportation problems that are variants of the classical Vehicle Routing Problem, and which consider additional real-world characteristics.

Another classical problem that we extend with additional constraints is the Pickup and Delivery Problem. In this problem a set of pickup locations and a set of delivery locations are given, and each customer specifies a demand, which must be picked up at a given location and delivered at a given location. Thus, besides the described constraints of the Vehicle Routing Problem, pairing constraints must be satisfied, i.e. for each customer pickup and delivery must be performed in the same route, and precedence constraints are imposed, i.e. for each customer pickup must precede delivery. Similarly as in the Vehicle Routing Problem, the goal is to find a set of minimum cost routes for a fleet of identical vehicles with limited service capacity, which start and end at the unique depot and that serve the given set of customers, satisfying pairing and precedence constraints. For comprehensive reviews on routing problems involving pickups and deliveries, the reader is referred to the works of Savelsbergh and Sol [81], Cordeau et al. [18] and Parragh et al. [64]. We extend the classical Pickup and Delivery Problem by considering additional real-world constraints. In particular, we deal with several depots, consider a heterogeneous fleet of vehicles and take into account for each customer a soft time window: usually only hard time windows are considered, i.e. service of each customer must happen within the given hard time window. However, sometimes this is too strict, and a more realistic behavior is to allow violation of the hard time window, and apply a (linear) penalization cost when the hard time windows is violated.

In the remainder of the chapter, we briefly describe the two problems that

we studied: the first one is an extension of the Vehicle Routing Problem and the second one is an extension of the Pickup and Delivery Problem. For both problems we propose exact and heuristic approaches and perform computation experiments on benchmark instances from the literature.

1.2 Multi-Depot Heterogeneous Vehicle Routing Problem with Time Windows

The first problem that we consider is the Multi-Depot Heterogeneous Vehicle Routing Problem with Time Windows (MDHVRPTW). It is a variation of the vehicle routing problem with time windows in which the transportation fleet is made by vehicles with different capacities and fixed costs, based at different depots. To the best of our knowledge, no exact algorithms have been proposed for the MDHVRPTW so far.

We introduce a set covering formulation for the problem, with a binary variable for each feasible route, taking value 1 if the route is selected in the solution. Since the model presents an exponential number of variables, we adopt column generation techniques for the solution of its Linear Programming (LP) relaxation. Given a depot and a vehicle type, the problem of finding the most negative reduced cost column (corresponding to a route for this vehicle type using the considered depot) turns out to be a Resource Constrained Elementary Shortest Path Problem (RCESPP). This problem is NP-hard and we solve it using three pricing algorithms: a greedy one, a heuristic dynamic programming one and an exact dynamic programming one. They are called in sequence, only if the previous pricing algorithm cannot find any column with negative reduced cost.

We present a branch-and-cut-and-price algorithm, based on the described LP-relaxation, for obtaining the exact solution to the problem. The algorithm extends some recent ideas, such as bi-directional dynamic programming and decremental state space relaxation. In particular, for the exact solution of the RCESPP we use the technique proposed in [71], which consists of a

bi-directional extension of node labels. In addition, we use decremental state space relaxation: the idea is to project the state space of the problem to a smaller one, by removing some elementarity constraints. One of the main difficulties with the heterogeneous fleet and multi-depot version of the Vehicle Routing Problem is that the reduced cost of the routes depends on the depot chosen and the vehicle type. Hence in principle it would be necessary to execute the pricing algorithm for each combination of vehicle type and depot. One of the main features of our algorithm is that multiple execution of the pricing algorithm is avoided. This is done by an aggregated pricing algorithm. We also consider different cutting techniques: in particular, we add violated 2-path inequalities, i.e. we search for a set of customers that are served by less than 2 vehicles in the fractional solution but require at least 2 vehicles in any integer solution. We embed the dynamic generation of these inequalities in the column generation scheme. Two branching policies are considered in the branch-and-cut-and-price algorithm: one consists in branching on the number of vehicles, and the other one in branching on arcs.

Extensive computational results are reported on the use of the algorithm both for exact optimization and as a heuristic method. The algorithm is tested on benchmark instances from the literature and compared with state-of-the-art algorithms. It is also tested on new instances, showing the behavior when all the real-life constraints are considered (multi depot and heterogeneous fleet). The results of the study are presented in [10].

1.3 Multi-Depot Heterogeneous-Fleet Pickup and Delivery Problem with Soft Time Windows

The second problem that we consider is the Multi-Depot Heterogeneous-Fleet Pickup and Delivery Problem with Soft Time Windows (MDHPDPSTW). It is an extension of the classical Pickup and Delivery Problem with Hard Time Windows. The problem requires to find a minimum cost routing for a fleet

vehicles with different capacities and based at different depots, satisfying a given set of customers. A customer request is associated with two locations: a source where a certain demand must be picked up and a destination where this demand must be delivered. Each route must satisfy pairing constraints (pickup and delivery of a customer must both be visited in the same route) and precedence constraints (the delivery location must be visited after the corresponding pickup location). Further, each pickup and delivery location has a time windows for the service that can be violated at the cost of a linear penalty. This problem has application in various scenarios, such as urban courier services, less-than-truckload transportation, door to door transportation services.

We propose a branch-and-price algorithm for the MDHPDPSTW which combines ideas proposed in [74] with bidirectional label extension and decremental state space relaxation and use a modified version of the algorithm developed by Liberatore et al. [57] to handle soft time windows. Bidirectional label extension is adapted to deal with the Pickup and Delivery Problem, taking into account pairing constraints and precedence constraints. In addition, since we are dealing with soft time windows, we have the possibility of trading cost for time: we are allowed to violate given hard time windows at a penalty cost. This information is stored in a node label as a cost function and we adapt label extension to deal with this new feature.

We report computational experiments on several benchmark instances both to measure the performance of the proposed algorithm and to investigate the impact of soft time windows on the structure of the optimal solutions. A preliminary discussion on the study has been presented in [11].

1.4 Original Contributions

The contributions of this thesis follow both theoretical and practical directions. From a theoretical point of view, new modifications to an existing method, introduced by Righini and Salani [71, 78], are proposed in order to deal with very general routing problems. The method was designed for

solving by branch-and-price the Capacitated Vehicle Routing Problem, the Vehicle Routing Problem with Distribution and Collection, and the Capacitated Vehicle Routing Problem with Time Windows. Starting from the basic version of the method, we introduced new changes that allowed us to tackle routing problems characterized by the following additional features:

- multiple depots instead of just one depot where all the vehicles are based;
- a heterogeneous fleet of vehicles (i.e. vehicles with different capacities) instead of a single vehicle type;
- soft time windows (i.e. time windows where a linear penalty is applied if the vehicle visits the customer outside the specified time window) instead of the more common hard time windows
- pickup and delivery to the customers instead of just pickup.

The changes that we introduced in order to take into account these additional features require relevant modifications in the structure of the algorithms used for the problem solution. In addition, the changes in the methodology implied substantial changes in the implementation and in particular the use of clever data structures that could keep the efficiency of the method even in a more difficult environment. Moreover, the two different studied problems required very different changes.

Last but not least, to the best of our knowledge, no exact method was designed for routing problems including all these additional features.

From a practical point of view, the additional considered features lead to more realistic problems, and reduce the gap between theory and practice. Of course, additional constraints might be considered in a real-world setting. However, the insertion of the described characteristics of the problems helps in exploring a more realistic setting and in understanding how far the designed algorithms can go and fulfill the real company requirements. Extensive computational experiments have been performed in order to compare

the proposed algorithms with state-of-the-art algorithms that however do not include these additional features. In addition, computational testing has been done on more realistic instances, characterized by multiple depots, heterogeneous fleet of vehicles, pickup and delivery to the customers, soft time windows. The effectiveness of the algorithms presented in this work is also encouraging in the study of exact and heuristic methods for real-world involved problems.

1.5 Outline

The thesis is organized as follows: in Chapter 2 we introduce the very well-known Vehicle Routing Problem and Pickup and Delivery Problem, and we present the main results in the literature for these problems. In Chapter 3 we describe the main ideas of the methodology that is the starting point for the solution methods used in this thesis. In particular, we will describe the branch-and-price technique applied to the Vehicle Routing Problem, with details on column generation and on the corresponding arising subproblem. In Chapter 4 and 5 we will present the study on two variants of the Vehicle Routing Problem and Pickup and Delivery Problem, respectively. We will formally describe the problems and we will present new approaches for obtaining exact and heuristic solutions to these problems. Finally, in Chapter 6 we will draw some conclusions and we will present possible topics for future research.

Chapter 2

Vehicle routing and pickup and delivery problems

In this chapter the basic versions of vehicle routing and pickup and delivery problems are introduced. They will be useful to describe the main ideas of the solution method used in this work, without the complications introduced by the additional constraints present in the variants of the problems studied in this thesis.

2.1 The Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a very well-known combinatorial optimization problem. It concerns the effective management of the transportation of goods and services in distribution systems. In this section, we will describe the problem, we will present the main Integer Linear Programming models used for it, and finally we will describe the most effective methods developed for its solution.

2.1.1 Problem Description

The VRP (also known as Capacitated VRP) can be defined as follows: given a set of customers, each one with a demand to be served, given a fleet of vehicles, each one with a capacity, and given a depot at which the vehicles are based, it calls for determining an optimal set of routes (to be performed by the fleet of vehicles), starting and ending at the depot, so that all the customers' demands are satisfied. This problem is defined on a graph $G = (V, A)$ that represents a road network. The set of vertices $V = \{0, \ldots, n\}$ contains the depot (represented by the vertex 0) and one vertex for each customer. Each arc (i, j) in the set A corresponds to a road section and is assigned a nonnegative travel cost c_{ij} (generally representing its length) and a nonnegative travel time t_{ij}. Each customer $i \in \{1, \ldots, n\}$ has a given nonnegative demand q_i to be served. The fleet is composed of K identical vehicles, each one with a given capacity Q. The goal is to determine an optimal set of minimum cost routes (including arc cost and travel cost), starting and ending at the given depot, with the constraints that the demand of each customer is served by exactly one route, and that the capacity of the vehicles is respected.

2.1.2 Mathematical formulation

In this section we recall two basic mathematical programming formulations for the VRP: a flow model and a set partitioning model. An exhaustive discussion of mathematical formulations can be found in [88] and [54].

2.1.2.1 A flow model

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

$$\text{minimize} \sum_{i \in V} \sum_{j \in V} c_{ij} \sum_{k=1}^{K} x_{ijk} \tag{2.1}$$

$$\text{s.t.} \sum_{k=1}^{K} y_{ik} = 1 \qquad \forall i \in V \setminus \{0\} \tag{2.2}$$

$$\sum_{k=1}^{K} y_{0k} = K \tag{2.3}$$

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} = y_{ik} \qquad \forall i \in V, k = 1, \ldots, K \tag{2.4}$$

$$\sum_{i \in V} q_i y_{ik} \leq Q \qquad \forall k = 1, \ldots, K \tag{2.5}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \qquad \forall S \subseteq V, |S| \geq 2, k = 1, \ldots, K \tag{2.6}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i \in V, j \in V, k = 1, \ldots, K \tag{2.7}$$

$$y_{ik} \in \{0, 1\} \qquad \forall i \in V, k = 1, \ldots, K \tag{2.8}$$

where (2.6) is one of the known formulations for the subtour elimination constraints. The vehicle flow model has been successfully applied in branch-and-bound and branch-and-cut algorithms for the VRP (see Toth and Vigo [87] and Naddef and Rinaldi [62]). This model can be extended to represent constraints on the type of vehicle serving each customer or vehicle-related constraints, such as different capacities or tour lengths. Details on this topic can be found in Toth and Vigo [89].

2.1.2.2 A set partitioning model

The set partitioning model presents an exponential number of binary variables: one for each feasible route. Let Ω be the set of all feasible routes and let x_r for each $r \in \Omega$ be a binary variable assuming value 1 if the route is selected in the solution and 0 otherwise. In addition, let a_{ir} be a binary coefficient assuming value 1 if and only if customer $i \in V \setminus \{0\}$ is visited by route r. We indicate with c_r the cost of route $r \in \Omega$. The model reads as

follows:

$$\text{minimize} \sum_{r\in\Omega} c_r x_r \tag{2.9}$$

$$\text{s.t.} \sum_{r\in\Omega} a_{ir} x_r = 1 \qquad \forall i \in V\backslash\{0\} \tag{2.10}$$

$$\sum_{r\in\Omega} x_r = K \tag{2.11}$$

$$x_r \in \{0,1\} \qquad \forall r \in \Omega. \tag{2.12}$$

The objective (2.9) aims at minimizing the cost of the selected routes. Constraints (2.10) require that each customer is visited by exactly one route and constraints (2.11) impose to select exactly one route for each vehicle. The main advantage of this model is that it can handle many different constraints, since they are hidden in the definition of the routes. In addition, the linear programming relaxation of this model is typically very tight. If the cost matrix satisfies the triangle inequality, then the set partitioning formulation may be transformed into an equivalent Set Covering model, by replacing constraints (2.10) with the following constraints:

$$\sum_{r\in\Omega} a_{ir} x_r \geq 1 \quad \forall i \in V\backslash\{0\} \tag{2.13}$$

The main advantage of the latter formulation is that only inclusion-maximal feasible routes, among those with the same cost, need to be considered in the definition of Ω, and this reduces the number of variables. In addition, when using the set covering formulation, the dual space is reduced since dual variables are restricted to nonnegative values only. One of the main drawbacks of both model is in the huge number of variables. Column generation techniques help in this direction for solving the linear programming relaxation of such models.

2.1.3 Literature Review

Research in the field of vehicle routing started in 1959 with the *truck dispatching problem* by Dantzig and Ramser [20]. Since then a huge number of works have been proposed in the literature for this problem. For a comprehensive survey on solution techniques for the vehicle routing problem we refer the reader to [19, 16, 89]. We recall here some recent contributions.

A robust branch-and-cut-and-price (BCP) was presented in Fukasawa et al. [43]. It combined column generation over q-routes with cuts over the edge CVRP formulation. A q-route is a walk that starts at the depot vertex, traverses a sequence of clients vertices with total demand smaller or equal to the capacity Q of the vehicle, and returns to the depot (i.e. the elementarity constraint is dropped). While pricing real CVRP routes would lead to a strongly $\mathcal{NP}$-hard problem, q-routes can be found in pseudo-polynomial time by dynamic programming. Since the cuts on the edge variables can be translated in arc costs in the dynamic programming, the pricing remains tractable. This algorithm was able to solve instances from the literature with up to 135 customers.

Another successful algorithm has been proposed by Baldacci et al. [4]. It obtain a good estimate of the optimal dual variables values through cheap lower bounding procedures. Then a BCP is called with the dual variables bounded to be above and close to the estimates. Convergence is obtained with very few calls to the dynamic programming pricing. This algorithm was able to solve almost all the instances solved by Fukasawa et al., often in much less time.

Several heuristic algorithms have also been proposed. Toth and Vigo [90] developed a granular tabu search (GTS) algorithm that a priori removes from the graph edges that are unlikely to appear in an optimal VRP solution. Li et al. [56] put forward an algorithm that combines the record-to-record principle with a variable-length neighbor list whose principle is similar to GTS. A very large neighborhood search has been proposed by Ergun et al. [41]. Neighbor solutions are defined by means of 2-opt moves, vertex swaps

between routes, and vertex insertions in different routes. In order to determine the best sequence of moves at a given iteration, a shortest path problem is solved on an auxiliary graph, called improvement graph. Prins [68], and Berger and Barkaoui [8] proposed memetic algorithms that combines features of evolutionary strategies with local search procedures. Tarantilis and Kiranoudis [85] have developed a rather effective adaptive memory procedure. In a first phase a solution is obtained by means of a constructive procedure, followed by a tabu search procedure. The adaptive memory procedure initiates new solutions by combining route segments, called bones, extracted from good quality routes. Mester and Bräysy [61] proposed an iterative two-stage procedure that combines guided local search with evolution strategies. Pisinger and Ropke [65] developed an adaptive large neighborhood search framework to solve five different variants of VRPs. Toth and Tramontani [86] proposed a local search algorithm based on the exploration of an exponential neighborhood by solving an integer linear programming problem.

2.2 The Pickup and Delivery Problem

2.2.1 Problem Description

The Pickup and Delivery Problem (PDP) is a generalization of the VRP, in which each customer i is associated with two locations: a source where a certain demand q_i must be picked up and a destination where this demand must be delivered. Each route must satisfy pairing constraints (pickup and delivery of a customer must both be visited in the same route) and precedence constraints (the delivery location must be visited after the corresponding pickup location). As in the VRP case, the problem is defined on a graph $G = (V, A)$ that represents a road network. The set of vertices $V = \{0, \ldots, n\}$ contains the depot (represented by the vertex 0) and one vertex for each location. The set of arcs represents the set of road sections, and each arc is assigned a nonnegative travel cost c_{ij} and a nonnegative travel time t_{ij}. The fleet is again composed of K identical vehicles, each one with a given

capacity Q. The PDP calls for determining a set of minimum cost routes, starting and ending at the given depot, with the constraints that the demand of each customer is picked up at the customer source and delivered at the customer destination in the same route, and that the capacity of the vehicles is respected.

A Set Partitioning Integer Linear Programming Model

The model (2.9)-(2.12) is valid also for this variant of the VRP, indeed the additional constraints arising in PDP can be dealt with through the definition of the feasible routes. The considerations on the transformation of the model to a Set Covering formulation remain valid.

2.2.2 Literature Review

A column generation approach is elaborated by Dumas et al. [40]. They consider heterogeneous vehicles, time windows as well as multiple depots. The constrained shortest path problems are solved by means of a forward dynamic programming algorithm.

Ropke and Cordeau [74] introduce a branch-and-price algorithm with additional valid inequalities. The elementary constrained shortest path problem with time windows, capacity, and pickup and delivery restrictions is the natural pricing problem of the PDPTW. It is solved by means of a label setting shortest path algorithm. Valid inequalities are added in a branch-and-cut fashion; their impact on the structure of the pricing problem is discussed.

A branch-and-cut algorithm departing from two different two-index PDPTW formulations is studied by Ropke et al. [75]. Formulation one makes use of time variables. In formulation two time related constraints are modelled by means of infeasible path inequalities. The latter formulation proves more efficient when used as a basis for the branch and cut algorithm. New valid inequalities to strengthen the proposed formulations are also discussed.

Baldacci et al. [3] proposed an exact algorithm based on a set partitioning-like formulation. They describe a bounding procedure to find near optimal dual solution of the LP-relaxation of the formulation. The dual solution is used to generate a restricted problem which is solved by an integer programming solver.

Xu et al. [92] propose a column generation based heuristic algorithm. They consider several additional constraints, such as multiple time windows at pickup and delivery locations, loading restrictions, compatibility of goods and vehicles as well as driver working hours. The column generation master problem can be solved using a commercial LP solver. The resulting subproblems are solved by means of two heuristics, called merge and two-phase, i.e. merging trips and greedy deletion and insertion of requests. Lu and Dessouky [59] elaborate a construction heuristic. It does not only incorporate distance increase into the evaluation criterion but also time window slack reduction as well as visual attractiveness (referred to as crossing length percentage).

Pankratz [63] proposes a grouping genetic algorithm for the PDP. It differs from traditional genetic algorithms in that a group-oriented genetic encoding is used. The encoding used by [63] corresponds to the cluster of requests forming a route. The routing aspect, not comprised in the encoding, is added while decoding the chromosome.

Ropke and Pisinger [76] present an adaptive large neighborhood search algorithm for the PDPTW. Multiple depots as well as the existence of service times can be handled by the approach at hand. A two-stage hybrid algorithm for the static PDPTW is presented by Bent and van Hentenryck [7]. The first phase uses simulated annealing to decrease the number of vehicles needed. The second phase consists of a large neighborhood search algorithm in order to reduce total travel cost. Derigs and Döhmer [23] discuss an indirect (evolutionary) local search heuristic for the same problem. In indirect search solutions are encoded such that the problem of securing feasibility is separated from the metaheuristic search process. Here a greedy decoding is used.

Chapter 3

Branch-and-Price for the VRP

Branch-and-price algorithms belongs to the family of branch-and-bound algorithms. Their peculiarity is the use of *Column Generation* techniques to compute a dual bound. They have been successfully applied to a wide variety of optimization problems [31]. In this chapter we describe how the method works for the capacitated VRP and we refer, in particular, to the method used by Righini and Salani [78, 72]. It has been also applied to the VRP with time windows [78] and the VRP with simultaneous pick-up and delivery [22]. In Chapters 4 and 5, we will present a new adaptation of the methodology to deal with variants of the VRP and PDP, dealing with additional real-world constraints.

3.1 Column Generation

Let us consider the LP-relaxation of model (2.9)-(2.12), i.e. constraints (2.12) are replaced by

$$0 \leq x_r \leq 1 \quad \forall r \in \Omega.$$

This problem is called *master problem* (MP). 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 dual variables λ_i $i \in V \backslash \{0\}$ of constraints (2.10) and

the dual variable μ of constraint (2.11), we want to find:

$$\min_{r\in\Omega}\left\{\bar{c}_r = c_r - \sum_{i\in V\setminus\{0\}} a_{ir}\lambda_i - \mu\right\}$$

The complexity of this step depends on $|\Omega|$ and can be time consuming when $|\Omega|$ is huge. The idea is to consider a small subset of columns $\Omega' \subseteq \Omega$: 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^*, μ^*) be respectively the primal and dual optimal solutions of the RMP. Then the problem

$$c_r^* = \min_{r\in\Omega}\left\{\bar{c}_r = c_r - \sum_{i\in V\setminus\{0\}} a_{ir}\lambda_i^* - \mu^*\right\}$$

is an oracle for pricing. If the solution is non-negative then no reduced cost coefficient $\bar{c}_r$ has a negative value, the restricted master problem cannot be further improved and $\mathbf{x}^*$ is the optimal solution also for the original master problem. Otherwise the column which has the reduced cost equal to c_r^* is a candidate to enter the basis and it is added to the RMP. The process is repeated until no negative reduced cost columns are found. The method described is known as *Column Generation*. If the set of negative reduced cost solutions is finite then the column generation algorithm converges and is exact. The critical part of the column generation procedure is, of course, the pricing step. Let ξ_a $(a = (i, j) \in A)$ be a binary variable assuming value 1 if arc a is selected in the solution and z_i $(i \in V)$ be a binary variable assuming value 1 if vertex i is visited. Let $\delta^+(i)$ and $\delta^-(i)$ represent the outgoing arcs and the ingoing arcs of vertex i, respectively. Let $n+1$ indicate an additional vertex representing a copy of the depot 0, added for convenience. Recall that λ_i^* $(i \in V)$ and μ^* are the optimal dual variables of the RMP, associated with constraints (2.10) and (2.11) respectively. A new column with minimum

reduced cost is obtained by solving the following pricing problem:

$$\text{minimize} \sum_{a \in A} c_a \xi_a - \sum_{i \in V \setminus \{0\}} a_{ir} \lambda_i^* - \mu^* \tag{3.1}$$

$$\text{s.t.} \sum_{a \in \delta^+(0)} \xi_a = 1 \tag{3.2}$$

$$\sum_{a \in \delta^-(n+1)} \xi_a = 1 \tag{3.3}$$

$$\sum_{a \in \delta^+(i)} \xi_a = z_i \qquad i \in \{1 \dots n\} \tag{3.4}$$

$$\sum_{a \in \delta^-(i)} \xi_a = z_i \qquad i \in \{1 \dots n\} \tag{3.5}$$

$$\sum_{a \in \delta^+(S)} \xi_a \geq z_i \qquad S \subseteq V \setminus \{0\}, i \in S \tag{3.6}$$

$$\sum_{i=1}^{n} q_i z_i \leq Q \tag{3.7}$$

$$\xi_a \in \{0, 1\} \qquad a \in A \tag{3.8}$$

$$z_i \in \{0, 1\} \qquad i \in \{1 \dots n\} \tag{3.9}$$

The objective (3.1) is to minimize the arc costs, while taking into account a *prize* given by the dual variables λ_i for each visited customer. Since μ does not depend on the route choice, it is not necessary to consider it in the subproblem model. Constraint (3.2) (resp. (3.3)) imposes to have one outgoing (resp. ingoing) arc from the depot. Constraints (3.4) and (3.5) ensure that if customer i is visited, thus we choose an ingoing and an outgoing arc for it. Constraints (3.6) avoid to have sub-tours which do not contain the depot. Constraint (3.7) allows to use at most the available capacity of the vehicle for satisfying the customers' demands. This problem is known as *Resource Constrained Elementary Shortest Path Problem* (RCESPP) and the method used for solving it will be described in details in Section 3.3.

3.2 Branch-and-price

When we have to solve the integer linear program (2.9)-(2.12), we need to embed the column generation process, described above, into a branch-and-bound algorithm. The first known attempt dates back to 1961 when Gilmore and Gomory [45] developed a column generation approach to the cutting stock problem. Several applications of column generation techniques appeared in the last decades. Desrosiers et al. [32], Desrocher and Soumis [28], Ribeiro et al [70], Vance et al. [91] and Gamache et al. [44] are but a few example of successful application of this technique to hard Integer Programming (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 specific algorithms can be devised to price out new columns. Indeed, 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. A branch-and-price algorithm works as follows. The search tree is properly initialized with the root node. At each node of the search tree the RMP is initialized with a subset of feasible columns for the active node. Then the RMP 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. As in standard branch-and-bound, the dual bound is used to avoid exploring nodes of the search tree that are not promising and cannot lead to an improved solution. At the end of the search process the best integer solution is the optimal solution for the original integer linear problem.

A valid branching scheme should cut off the current fractional solution, produce a balanced search tree and 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 the structure of the pricing problem. While fixing a variable to 1 does not create troubles 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.

In the approach of Salani [78], two different branching scheme are applied. The first one consists in branching on the arc choice. A vertex i, belonging to several fractional routes in which the arcs entering it or leaving it are different, is selected. Half of its outgoing arcs are then forbidden in one branch and the other half are forbidden in the other branch. The second branching scheme consists in branching on the number of vehicles. We will see an example of these two schemes applied to the variants of VRP and PDP studied in the next two chapters.

3.3 The Resource Constrained Elementary Shortest Path Problem

In this section, we describe the subproblem that arises in the column generation process and the methodology used for solving it. This method was introduced by Righini and Salani [78, 72] who modified and improved a previous approach by Feillet et al. [42]. The solution approach is based on dynamic programming techniques.

The resource constrained elementary shortest path problem (RCESPP) is defined as follows: a graph $G = (V, A)$ is given, where the vertex set V is made by a set of vertices N representing n customers and two vertices s and t representing the depot. A non-negative prize λ_i is associated with each vertex $i \in N$, a non-negative cost λ_0 is associated with the depot and a non-negative cost c_{ij} is associated with each arc $(i, j) \in A$. 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. Additional constraints must be taken into account, depending on the kind of vehicle routing problem at hand. All these additional constraints are modelled as resource constraints. In the case of the classical VRP, we have only one resource (i.e. the capacity of the vehicles) and a possible ILP model is (3.1) - (3.9), where $\lambda_0 = -\mu$, $s = 0$, and $t = n + 1$.

If the underlying graph may have negative cost cycles (e.g. in a column generation approach for the VRP), the RCESPP is strongly $\mathcal{NP}$-hard [36]. It is possible to address the pricing problem by optimizing its relaxation, obtained by dropping the elementarity constraints. 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. [42] and Desrochers et al. [29] to solve the vehicle routing problem with time windows (VRPTW) through column generation.

In the dynamic programming approach for RCSPP (see Desrochers [27]), a state is associated to each vertex i: it 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 state 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 amount of the corresponding resource are fathomed. The cost is initialized

at 0 at vertex s and it is updated according to the formula

$$C' := C + c_{ij} - \lambda_i/2 - \lambda_j/2 \tag{3.10}$$

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. For the case of the classical VRP, where the resource consists of the capacity Q of each vehicle (i.e. the sum of the demands of the vertices visited by the same vehicle cannot exceed Q), the constraint is modelled in the state by one resource, representing the amount of capacity still available along a path. Let χ be the amount of resource consumed. Each state is represented by a label (χ,C,i). Every time a node i is visited the corresponding amount of load q_i is stored on board, therefore χ is increased by q_i. A state (χ,C,i) is feasible only if $\chi \leq Q$.

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 [49] proposed to add to the state an additional binary resource for each vertex $i \in N$. There is only one unit available for each dummy resource and it is consumed when the corresponding vertex is visited.

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

- (a) $S' \leq S''$
- (b) $R' \leq R''$
- (c) $C' \leq C''$

and at least one of the inequalities is strict. Feillet et al. [42] 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. This enhancement allows the dynamic programming algorithm to fathom a larger number of states and to reduce the computation time. 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. Rghini and Salani [71] proposed to use a bounded bi-directional dynamic programming method to mitigate this phenomenon.

Bounded bi-directional dynamic programming 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. More precisely, 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 are associated forward and backward states. A path from s to t is detected each time a forward state and a backward state can be

feasibly joined. In particular a feasibility test imposes that the 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.

As mentioned above, 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 (capacity in the case of classical VRP), 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.

Decremental state space relaxation Christofides et al. introduced the idea of state space relaxation in order to reduce complexity and computing time. It consists of mapping each state (S,R,C,i) onto a new state (σ,R,C,i), where σ represents the length of the path, that is the number of vertices visited (excluding s). A dynamic programming algorithm based on state space relaxation must explore only a pseudo-polynomial number of 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 dominance rule is modified as follows: (σ',R',C',i) dominates (σ'',R'',C'',i) if

- (a) $\sigma' \leq \sigma''$

- (b) $R' \leq R''$
- (c) $C' \leq C''$

and at least one of the inequalities is strict. This state space relaxation can be tightened by eliminating all cycles of length two.

As shown in [72], this idea can be integrated into an exact dynamic programming algorithm for the RCESPP as follows: some vertices are identified as critical, according to the structure of the optimal RCSPP solution obtained with state space relaxation. Let $\bar{S}$ indicate the set of critical vertices at the current iteration. In the subsequent iteration the dynamic programming algorithm prevents multiple visits to the vertices in $\bar{S}$, still allowing multiple visits to the others. This is easily accomplished by extending the state space relaxation labels with a binary vector restricted only to the critical vertices. This algorithm is known as *decremental state space relaxation*. 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.

Chapter 4

Multi-Depot Heterogeneous-fleet Vehicle Routing Problem with Time Windows

4.1 Introduction

There is a considerable economic interest around algorithmic techniques for optimization in logistics, since transportation represents a relevant factor in the final cost of goods.

Many real applications require to find a set of routes of minimum length for a fleet of vehicles with limited resources, in order to serve a given set of customers. When routes start and end at a unique depot, each customer has a given demand which must be served by a single vehicle, and the fleet is composed by identical vehicles with limited service capacity, the problem turns out to be a classical Vehicle Routing Problem (VRP) [46].

However these applications often require to take into account several additional constraints [79] [67]. For instance, in several applications involving interaction with human personnel or customers, visiting certain locations is allowed only in certain time windows; furthermore, large companies or consortia of small transportation companies usually have more than one depot

and they often manage a fleet made of vehicles with different capacities and operating costs [12].

Therefore, in this work we consider three major extensions of the basic VRP model, namely time windows, heterogeneous vehicles and multiple depots. The resulting problem is called Multi-Depot Heterogeneous Vehicle Routing Problem with Time Windows (MDHVRPTW): we want to simultaneously determine the optimal composition, placement and routing of a heterogeneous fleet of capacitated vehicles in order to satisfy the delivery demands of a set of customers under time constraints.

We refer the reader to the recent book by Golden et al. [46] for an up-to-date survey of VRP models and methods. Several variants of the VRP involving subsets of the features of the MDHVRPTW have been studied in the literature.

State-of-the-art methods for the VRP with Time Windows (VRPTW) are able to solve instances with up to some hundreds customers [26] [46].

The Heterogeneous Vehicle Routing Problem with Time Windows (HVRPTW) in the variant without any limit on the number of available vehicles was first addressed by Liu and Shen [58] under the denomination of Fleet Size and Mix VRP with Time Windows; the authors proposed a number of insertion-based parallel savings heuristics, performing tests on instances with up to 100 customers. Later, Dullaert et al. [38] developed three insertion-based heuristics, that are an extension of Solomon's [83] sequential insertion heuristic combined with ideas of Golden et al.'s [47]. The authors tested their algorithms using a slightly different objective function. More recently Belfiore and Fvero [6] and Dell'Amico et al [21] respectively proposed scatter search and "ruin-and-recreate" approaches to this problem. State-of-the-art heuristics are able to provide good quality approximations for problems with 100 customers in some minutes.

Instead, Polacek et al. [66] studied a Variable Neighborhood Search for the Multi-Depot Vehicle Routing Problem with Time Windows (MDVRPTW). A cluster-based optimization approach to the MDHVRPTW has been proposed

by Dondo et Cerd [33], who also developed a large-scale neighborhood method to improve the solutions found by their previous heuristic [34]. In this way, instances with 100 customers can be heuristically solved in a time ranging from some seconds to one hour.

Baldacci et al. [5] proposed a unified exact method, which is able to solve different classes of vehicle routing problems, including heterogeneous fleets and multiple depots (MDHVRP), without time windows; this is based on the exact solution of an integer linear programming problem and on dual heuristics. Instances involving up to 100 customers were solved to optimality in some hours of computing time.

Finally, in [12] a rich vehicle routing problem was considered: it involves multiple depots, heterogeneous vehicles, time windows and other features. The authors proposed a column generation heuristic yielding good results on instances with up to few hundred customers in some hours.

To the best of our knowledge, no exact algorithms have been proposed for the MDHVRPTW so far. In this paper we present a branch-and-cut-and-price algorithm for the exact optimization of the MDHVRPTW. The algorithm extends some recent ideas, such as bi-directional dynamic programming and decremental state space relaxation already applied to the VRPTW and other versions of the VRP. In particular, in Section 4.2 we introduce an extended formulation for the problem; in Section 5.3.1 we illustrate our column generation routines and show how the repeated execution of the dynamic programming-based pricing algorithm can be made more efficient; in the same section we also discuss the addition of strengthening inequalities and branching strategies. In Section 4.4 we report the results of an extensive experimental campaign, evaluating the effectiveness of our algorithm both as an exact and as a heuristic optimization method. Section 4.5 ends the paper with some concluding remarks.

4.2 Problem formulation

The MDHVRPTW can be formulated as follows. Let $\mathcal{G} = (\mathcal{N} \cup \mathcal{H}, \mathcal{A})$ be a directed graph whose node set is the union of a set $\mathcal{N}$ of customers and a set $\mathcal{H}$ of depots.

Non-negative costs d_{ij} and t_{ij} are associated with each arc $(i, j) \in \mathcal{A}$, representing respectively the travel cost and the travel time to reach $j \in \mathcal{N}$ starting from $i \in \mathcal{N}$. Each customer $i \in \mathcal{N}$ has a delivery demand q_i, a service time s_i and a delivery time window $[a_i, b_i]$.

We consider a set $\mathcal{K}$ of vehicle types, with known capacities w_k and fixed costs $f_k \; \forall k \in \mathcal{K}$. We assume that the number u_k of available vehicles of each type $k \in \mathcal{K}$ is limited. The number of routes associated with each depot $h \in \mathcal{H}$ is limited by a given value g_h. This represents the number of drivers living close to each depot, who are able to operate routes from it; we assume drivers to be able to use any type of vehicle.

In the version of the problem we study, we assume that all vehicles can be freely associated with any depot: this corresponds to the situation in which the location of the vehicles at the depots is a decision, not a datum.

Finally, vehicles do not necessarily leave their depots at time 0: due to time windows, it may be convenient to delay the departure time to reduce waiting time at customer locations. Following [58] we indicate as *duration* the difference between the arrival and the starting time of the route, that is the actual time spent in delivery operations. This cannot exceed a given limit D, representing the length of driver shifts.

The objective is to minimize the sum of vehicles fixed costs and routing costs, satisfying the following conditions:

I all customers must be served;

II each customer must be visited by only one vehicle

III the service at each customer must start within the customer time window;

IV each route begins at a depot and ends at the same depot;

V the sum of the demands of the customers served in each route must not exceed the capacity of the associated vehicle;

VI the duration of each route must not be greater than D.

VII the number of available vehicles for each type must not be exceeded;

VIII the number of allowed routes for each depot must not be exceeded;

Our framework allows to easily consider also the version in which the location of vehicles is fixed, i.e. in each depot a given number of vehicles of each type are pre-positioned; in this case a bound u_{hk} is imposed on the number of routes for each depot $h \in \mathcal{H}$ and for each vehicle type $k \in \mathcal{K}$, but the algorithm does not change. Nevertheless, in the remainder we discuss only the former version, because it is the only one having terms of comparison in the literature.

We introduce a set covering formulation of the MDHVRPTW as follows. We indicate as feasible any route satisfying conditions II, III, IV, V and VI. We indicate as Ω_{hk} the set of all feasible routes using a vehicle of type $k \in \mathcal{K}$ from depot $h \in \mathcal{H}$.

We associate a binary variable x_r with each feasible route: x_r takes value 1 if and only if route r is selected. We also use binary coefficients a_{ir} with value 1 if and only if customer $i \in \mathcal{N}$ is visited by route r. We indicate by c_r the cost of route r; it is equal to the sum of the vehicle fixed cost f_k and the routing costs, i.e. the optimal value of the Traveling Salesman Problem with Time Windows (TSPTW) on the customer set $\{i \in \mathcal{N} | a_{ir} = 1\}$ starting and ending at a particular depot. With these definitions we obtain the following

integer linear programming model:

$$\text{minimize} \sum_{h\in\mathcal{H}}\sum_{k\in\mathcal{K}}\sum_{r\in\Omega_{hk}} c_r x_r \tag{4.1}$$

$$\text{s.t.} \sum_{h\in\mathcal{H}}\sum_{k\in\mathcal{K}}\sum_{r\in\Omega_{hk}} a_{ir} x_r \geq 1 \qquad \forall i \in \mathcal{N} \tag{4.2}$$

$$\sum_{h\in\mathcal{H}}\sum_{r\in\Omega_{hk}} x_r \leq u_k \qquad \forall k \in \mathcal{K} \tag{4.3}$$

$$\sum_{k\in\mathcal{K}}\sum_{r\in\Omega_{hk}} x_r \leq g_h \qquad \forall h \in \mathcal{H} \tag{4.4}$$

$$x_r \in \{0,1\} \qquad \forall r \in \bigcup_{k\in\mathcal{K},h\in\mathcal{H}} \Omega_{hk}. \tag{4.5}$$

Constraints (4.2) are standard set covering constraints, modeling condition I, while (4.3) and (4.4) impose limits on the maximum number of available vehicles of each type and the maximum number of allowed routes for each depot, modeling conditions VII and VIII. The objective is to minimize the overall cost of the selected routes. In the remainder we indicate this formulation as *Master Problem* (MP).

4.3 Branch-and-cut-and-price

We solve the linear relaxation of the MP to obtain a lower bound which is used in a tree search algorithm. The number of variables is exponential in the cardinality of the customer set $\mathcal{N}$, thus we use a column generation approach [26]. In this section we describe the main components of the resulting branch-and-cut-and-price algorithm, namely pricing, cut generation and branching. We also discuss some implementation issues.

4.3.1 Column Generation

We initially consider only a small subset of the variables in the MP. Such initial Restricted Master Problem (RMP) includes

- a set of $|\mathcal{N}|$ columns, one for each customer, representing the optimal paths serving one customer at a time,
- a set of columns representing a feasible MDHVRPTW solution, found using a straightforward greedy policy,
- an additional *dummy column* $\bar{r}$ of very high cost, having $a_{i\bar{r}} = 1 \forall i \in \mathcal{N}$ and every other coefficient set to 0.

The aim of the dummy column is to ensure feasibility at each node of the search tree.

We solve the linear relaxation of the RMP, and we search for columns which are not in the RMP, but have negative reduced cost. If no such column exists, the solution is optimal for the MP linear relaxation as well, and thus yields a valid lower bound to the problem. On the opposite, if any negative reduced cost column is found, it is added to the RMP, and the process is iterated.

Let λ, μ and σ be the non negative dual vectors corresponding respectively to constraints (4.2), and to constraints (4.3) and (4.4) rewritten as $\geq$ inequalities.

The reduced cost of a column encoding route r has the following form:

$$c_r - \sum_{i \in \mathcal{N}} \lambda_i a_{ir} + \mu_k + \sigma_h.$$

As stressed before, the routes generated must comply with conditions II, III, IV, V and VI: it is important to note that the same sequence of customers may correspond to a feasible or to an infeasible route according to the type of vehicle and the depot it is associated with. For instance, different types of vehicles imply different capacities. Moreover, cost c_r and dual variables μ_k and σ_h depend on both k and h. Therefore, at each column generation iteration we have to solve $|\mathcal{K}| \cdot |\mathcal{H}|$ pricing problems. We also remark that it is never convenient to perform cycles in a feasible solution, although the prize structure given by dual variables can make it appealing; therefore, a search must be performed for elementary routes only.

Hence, given a particular depot h and vehicle type k, the problem of finding the most negative reduced cost column encoding a route for vehicle k using depot h turns out to be a Resource Constrained Elementary Shortest Path Problem (RCESPP).

The RCESPP is NP-hard [36]; following [78] we solve it using three pricing algorithms: a greedy one, a heuristic dynamic programming one and an exact dynamic programming one. They are called in sequence, only if the previous pricing algorithm cannot find any column with negative reduced cost. In the remainder we give a description of the pricing problem and of the three algorithms. For the purpose of better illustrating them, we follow the reverse order with respect to their execution.

4.3.1.1 Exact Dynamic Programming

Let us consider first the case of a single depot $h \in \mathcal{H}$ and a single vehicle type $k \in \mathcal{K}$: let s and v be the two distinct copies of depot h, representing the departure and arrival node and let $w = w_k$ be the capacity of vehicle k.

For the exact solution of the RCESPP we use the technique proposed in [71], which consists of a bi-directional extension of node labels. It associates labels with each node $i \in \mathcal{N}$ of the graph: forward labels represent paths from s to i and backward labels represent paths from i to v. Each label is iteratively considered and the corresponding path is extended to adjacent nodes.

Label structure. Each label has the form $(S, \chi, \tau, \delta, \rho, C, i)$. This tuple represents the state of the path associated with the label; C is the cost of the path, i is the last reached node, S, χ, τ, δ and ρ are either resources or resource consumptions with the following meaning: S is a set indicating which nodes have already been visited, χ indicates the amount of capacity consumed, τ indicates the elapsed time from time 0 up to the earliest point in time at which service at i can start, δ indicates the minimum duration of the path from the departure time up to the point in time at which service at i starts.

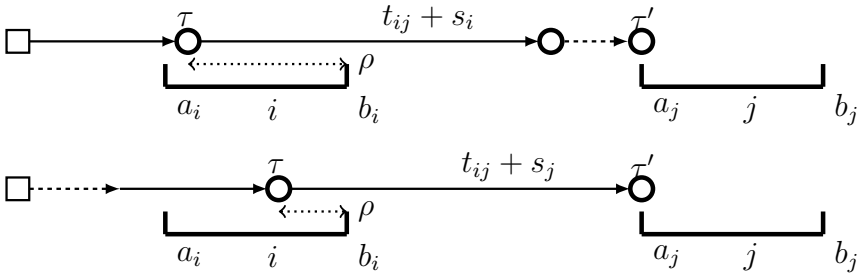

Figure 4.1: Label resources in the dynamic programming RCESPP algorithm.

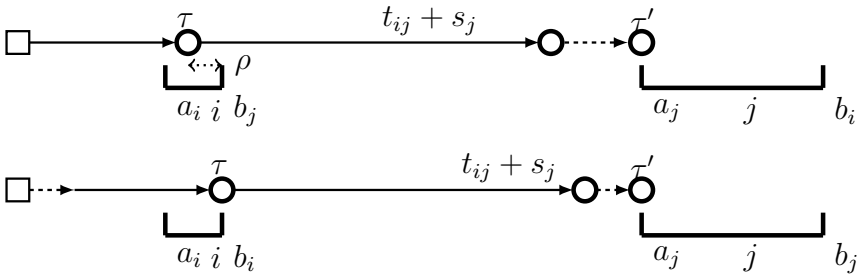

Figure 4.2: A time window limiting the time buffer ρ.

The value ρ is needed to update δ at each extension; it represents a time buffer, indicating the maximum amount of time by which the path defined so far can be delayed still remaining feasible with respect to the time windows of the visited nodes.

Figure 4.1 represents a sample instance of two nodes i and j on a time-line. The bold lines correspond to time spent in travel and service operations, the dashed lines correspond to waiting time either at an intermediate node or at the initial depot; the dotted line represents the time buffer ρ. By leaving the depot at time 0 and visiting node i (top figure), the vehicle has to wait until the time window of node j allows the service to start. Instead, such waiting can be avoided by delaying the departure from the depot (bottom figure). The time buffer ρ is spent to reduce the overall duration δ of the path.

Extension. In forward extensions the set S is initialized at $\emptyset$, all resource consumptions are initialized at 0 at the depot nodes, while ρ is initialized at

a very large value.

The search is restricted to elementary paths by discarding extensions to any node $j \in S$. When a label $(S, \chi, \tau, \delta, \rho, C, i)$ is extended forward to a node j, a new label $(S', \chi', \tau', \delta', \rho', C', j)$ is computed using the following rules.

First, resources S, χ, τ and C are updated as follows:

$$S' = S \cup \{j\} \tag{4.6}$$

$$\chi' = \chi + q_j \tag{4.7}$$

$$\tau' = \max\{\tau + s_i + t_{ij}, a_j\} \tag{4.8}$$

$$C' = C - \frac{1}{2}\lambda_i + d_{ij} - \frac{1}{2}\lambda_j; \tag{4.9}$$

in the updating formula for cost, we consider $\lambda_s = \lambda_v = 0$.

In order to update duration δ and time buffer ρ we have to consider a special case: if the vehicle arrives at j early, it should wait until a_j. Let us define the waiting time at j as $\omega = \max\{a_j - (\tau + s_i + t_{ij}), 0\}$. Hence the earliest starting time of service at j is equal to the earliest arrival time τ plus ω. Also the duration of the route is increased by the service time at i and the travel time from i to j, but the time buffer can be spent by delaying the departure from the depot to reduce the waiting time, as in the sample instance reported in Figure 4.1. It might not be possible to fully avoid waiting: the departure delay is limited by the amount ρ, which takes into account feasibility with respect to the time windows of the customers visited up to i, and by the waiting time, that is $\min\{\rho, \omega\}$; for example, in Figure 4.2 the time window of node i is tight (top figure), and the time buffer ρ is not enough to compensate the waiting time at node j (bottom figure).

Besides traveling and service time, the actual increase in duration is therefore $\omega - \min\{\rho, \omega\}$, that is $-\min\{\rho - \omega, 0\}$. By substitution with the expression of ω, we obtain the following update formula for δ':

$$\delta' = \delta + (s_i + t_{ij}) - \min\{\rho - \max\{a_j - (\tau + s_i + t_{ij}), 0\}, 0\}$$

The remaining amount of possible delay to absorb future waiting times is ρ', which is limited from above both by the remaining buffer due to the time windows of the customers in S, that is $\rho - \min\{\rho, \omega\}$, and by the constraint imposed by the time window of customer j, that is $b_j - \tau'$. The update formula for ρ' is therefore

$$\rho' = \min\{\rho - \min\{\rho, \max\{a_j - (\tau + s_i + t_{ij}), 0\}\}, b_j - \tau'\}.$$

The new state is feasible if and only if

$$\chi' \leq w \text{ (capacity constraint)} \tag{4.10}$$
$$\tau' \leq b_j \text{ (time window constraint)} \tag{4.11}$$
$$\delta' \leq D \text{ (duration constraint),} \tag{4.12}$$

otherwise it is fathomed.

The updating rules and feasibility tests for backward extension are symmetrical. Backward extensions start from time $T = \max_{i \in \mathcal{N}}\{b_i + s_i + t_{iv}\}$ that is the latest possible arrival time at the final depot. In particular the value $T - \tau$ in a backward label at a generic node j indicates the latest point in time at which service at the node j can end.

The algorithm iteratively extends each forward and backward label to all possible successors or predecessors respectively.

In order to limit the extension of forward and backward labels and to reduce useless duplication of paths, we impose that each partial path can use at most half of a critical resource whose consumption is monotone along the paths. The best critical resource is the tightest one. In our case we have two meaningful choices: either time or duration. Since in the instances from the literature it is often $D = +\infty$, we chose time as the critical resource.

Join. Forward and backward paths must be joined together to produce complete $s - v$ paths. The result of the join is a feasible solution if all the resource constraints are satisfied. When a forward path $(S^{fw}, \chi^{fw}, \tau^{fw}, \delta^{fw}, \rho^{fw},$

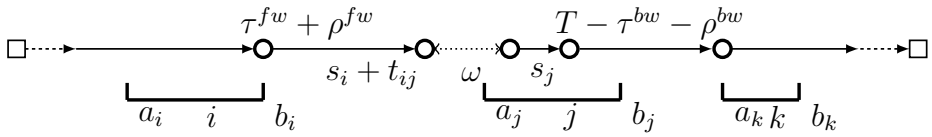

Figure 4.3: Computation of the waiting time during the join of two labels.

$C^{fw}, i)$ is joined with a backward path $(S^{bw}, \chi^{bw}, \tau^{bw}, \delta^{bw}, \rho^{bw}, C^{bw}, j)$, the feasibility conditions for a vehicle type k are:

$$S^{fw} \bigcap S^{bw} = \emptyset$$
$$\chi^{fw} + \chi^{bw} \leq w$$
$$\tau^{fw} + s_i + t_{ij} + s_j + \tau^{bw} \leq T$$

Furthermore, the constraint on the maximum duration of the whole path has to be satisfied. We remark that in backward labels, paths are supposed to end at time T, but can be shifted up to ρ^{bw} time units without violating time windows of the visited customers. Therefore $T - \tau^{bw} - \rho^{bw} - s_j$ represents the earliest point in time at which service at the node j can start in the backward path. The way in which backward paths are built already ensures $T - \tau^{bw} - \rho^{bw} - s_j \geq a_j$.

Let us consider the sample three-nodes instance depicted in Figure 4.3. The structure of the figure is similar to the previous ones: the bolded lines correspond to time spent in travel and service operations, the dashed lines correspond to late departure and early arrival at the depot, the dotted line represents waiting time. During the join on arc (i, j), the time window of node k prevents the backward path to be shifted by more than ρ^{bw} units in time. In the same way, the time window of node i prevents the forward path to be shifted by more than ρ^{fw} units in time.

The waiting time needed to join the forward and backward paths is therefore $\omega = T - \tau^{bw} - \rho^{bw} - s_j - (\tau^{fw} + \rho^{fw} + s_i + t_{ij})$, the actual waiting time at node j is $\max\{\omega, 0\}$ and the duration of the overall path is

$$\delta^{fw} + s_i + t_{ij} + \max\{\omega, 0\} + s_j + \delta^{bw}.$$

The feasibility conditions on the overall duration of the path is thus the following:

$$\delta^{fw} + s_i + t_{ij} + \max\{T - \tau^{bw} - \rho^{bw} - s_j - (\tau^{fw} + \rho^{fw} + s_i + t_{ij}), 0\} + s_j + \delta^{bw} \leq D$$

The reduced cost of the resulting $s - v$ path is

$$C^{fw} - \frac{\lambda_i}{2} + d_{ij} - \frac{\lambda_j}{2} + C^{bw} + f_k + \mu_k + \sigma_h.$$

Dominance test. During the extension of labels, a dominance test is done to fathom labels that cannot lead to an optimal solution. Let $l' = (S', \chi', \tau', \delta', \rho', C', i)$ and $l'' = (S'', \chi'', \tau'', \delta'', \rho'', C'', i)$ be two labels associated with node i. Then the former dominates the latter if the following conditions are satisfied

$$\begin{aligned}
&(a) \quad S' \subseteq S'' \\
&(b) \quad \chi' \leq \chi'' \\
&(c) \quad \tau' \leq \tau'' \\
&(d) \quad \delta' \leq \delta'' \\
&(e) \quad \rho' \geq \rho'' \\
&(f) \quad C' \leq C''
\end{aligned}$$

Condition (e) is too restrictive and can be relaxed into the surrogate inequality

$$(g) \quad \delta' - \rho' \leq \delta'' - \rho''$$

without losing the optimality guarantee. It is not hard to show by counter examples that inequalities (a), (b), (c), (d), (f) and (g) represent a set of necessary dominance conditions: dropping any of them optimal labels might be discarded.

Further, as shown in Feillet et al. [42], it is sometimes possible to identify some node $u \in \mathcal{N}$ that cannot be reached by any feasible extension of a given label, because of resource limitations. In this case it is useful to insert u in the set S of that label: it is easy to check that enlarging set S'' helps satisfying condition (a); at the same time, if a node cannot be reached by extending

label l' due to resource limitations, it cannot be reached by extending label l'' either, since resource consumption in l'' is not lower. Therefore, enlarging each set S allows dynamic programming fathoming a larger number of labels and hence reducing the computation time.

Decremental state space relaxation. The dynamic programming algorithm is executed iteratively applying decremental state space relaxation [72]. The idea is to project the state space of the problem to a smaller one, by removing some elementarity constraints. From an algorithmic point of view, this amounts to identify a set of *critical nodes* $\tilde{\mathcal{N}}$, and to replace extension rule (4.6) as

$$S' = (S \cup \{j\}) \cap \tilde{\mathcal{N}}$$

This relaxed problem can be solved more efficiently, since more labels can be compared in the dominance test.

In order to identify a good critical node set, we initialize $\tilde{\mathcal{N}} = \emptyset$; then we iteratively solve the state space relaxation of the pricing problem and insert in the set $\tilde{\mathcal{N}}$ all the nodes visited more than once in the optimal path, until we find an elementary one.

It may happen that even if the minimum reduced cost route contains a cycle, other elementary routes with negative reduced cost are found during the join phase. In this case we interrupt the pricing algorithm and we add these columns to the master problem. The effect is similar to that of partial pricing strategies [25]: the cost of the single column generation iteration is reduced at the expense of an increased total number of iterations.

Bounding. Using ideas similar to [14] and [4], we compute lower bounds in a preprocessing phase; these are checked during the dynamic programming iterations to detect suboptimal labels that can be fathomed.

Consider forward labels first: we disregard elementarity constraints, time windows and duration constraints, and for each node $i \in \mathcal{N}$ we compute the

least cost path from i to v that uses exactly p capacity units, for all values of p in the range $[q_i, \ldots, w]$.

Such computation requires the search for a capacity constrained shortest path; this is a weakly $\mathcal{NP}$-hard problem, and can be solved effectively in a single run for all vertices i and for all values $p \in [0, \ldots, w]$ as follows. We build an auxiliary graph with $w + 1$ layers, one for each value of p in the range, each containing a copy of the original graph. Then, each copy of node i in layer p' is connected to a copy of node j in layer p'' if and only if $p'' = p' + q_j$, and the cost of the arc connecting them is $d_{ij} - \lambda_i/2 - \lambda_j/2$; each copy of i in layer p is also connected to the copy of the same node in layer $p + 1$ with a zero-cost arc. Since this auxiliary graph is acyclic, one can use standard shortest path algorithms, and solve the problem of finding the optimal path starting from node i by finding the optimal shortest path tree rooted at node i in layer 0: the optimal path using at most p capacity units is the one connecting node i in layer 0 with node v in layer p. We note that the same computation can be done by reversing each arc and finding an optimal shortest path tree rooted in node v for each layer p. The overall complexity of this procedure is therefore $O(\min\{|\mathcal{N}|, w\} \cdot (|\mathcal{N}| \cdot w)^2)$.

Let us indicate as F_{ip} the value of the optimal path from node i in layer p to node v in layer w; since elementarity, time windows and duration constraints have been relaxed, this is a valid lower bound to the cost of any backward label associated with node i with $\chi \leq p$. When the exact dynamic programming algorithm is executed, if a forward label $(S, \chi, \tau, \delta, \rho, C, i)$ is such that

$$C + F_{i\chi} + f_k + \mu_k + \sigma_h \geq 0, \tag{4.13}$$

the label cannot lead to a negative reduced cost path and, therefore, it can be discarded.

If the instances are asymmetric, we pre-compute an analogous lower bound to possibly fathom backward labels applying the same idea in the other direction. This pre-computation is done once for each column generation iteration, once the dual values have been obtained from the linear relaxation of the restricted master problem.

Preliminary computational results showed that this technique considerably reduces the overall number of labels explored, with a negligible additional computational effort.

Aggregated pricing algorithm. One of the main difficulties with the heterogeneous fleet and multi-depot version of the VRP is that the reduced cost of the routes depends on the depot chosen and the vehicle type. Hence in principle it would be necessary to execute the pricing algorithm for each combination of $k \in \mathcal{K}$ and $h \in \mathcal{H}$. One of the main features of our algorithm is that multiple execution of the pricing algorithm is avoided.

First, in order to optimize all vehicle types at once, we extend the labels only once at each column generation iteration, using the capacity of the largest vehicle. Indeed, feasibility rules for time and duration are satisfied for each vehicle type, since travel time does not change from one type to another. Hence, during the join phase a column is generated for each vehicle type $k \in \mathcal{K}$ such that $w_k \geq \chi^{fw} + \chi^{bw}$. A similar approach can be applied to speed-up the bounding technique described above: the values F can be computed just once, considering only the largest capacity value. The drawback of this approach is that it weakens the bounding condition (4.13); in fact, in order to perform a valid check, we have to consider at the same time the capacity of the largest vehicle, and the smallest fixed costs $\min_{k\in\mathcal{K}}\{f_k + \mu_k\}$.

Furthermore it is possible to deal with all the depots in a single run as follows. We enlarge the state space by introducing different reduced cost (C_h), time consumption (τ_h), duration (δ_h) and time buffer (ρ_h) resources for each depot $h \in \mathcal{H}$. When a forward label is extended from a node i to a node j and $\tau_{\overline{h}} + t_{ij} + s_i > b_j$ for a certain $\overline{h} \in \mathcal{H}$, the path is not feasible for depot $\overline{h}$: thus we set $\tau'_{\overline{h}} = +\infty$. A similar argument applies to δ_h resources.

For what concerns domination rules, whenever two labels l' and l'' satisfy conditions $(a), (b), (c), (d), (f)$ and (g) for a particular depot $\overline{h}$, resources $\tau''_{\overline{h}}$, $\delta''_{\overline{h}}$ and $C''_{\overline{h}}$ are set to $+\infty$, as label l'' can never yield an optimal path for depot $\overline{h}$.

A forward label is fathomed when, due either to extensions, feasibility checks or dominance, it has $\tau_h > T/2$ and $\delta_h > D$ for each $h \in \mathcal{H}$.

Finally, we observe that in each label reduced costs C_h and resources τ_h, δ_h and ρ_h for each $h \in \mathcal{H}$ only differ for an offset given by the first and last arc in the path. We exploit this observation by avoiding to store a value for each $h \in \mathcal{H}$, and saving memory by computing such offset at each check.

4.3.1.2 Heuristic Dynamic Programming.

The heuristic dynamic programming pricing algorithm uses basically the same bi-directional label extension described above, but dominance condition (a) on the set S of visited nodes is dropped.

Since in this way several labels representing promising paths are fathomed, an additional dominance condition is enforced, as explained hereafter.

For each label, we define a *potential* $R(S, \chi)$ which is an upper bound on the amount of dual prizes λ that can be collected by completing the path.

In order to get a better evaluation of this potential, we reduce each coefficient λ_i by $\min_{j \in \mathcal{N} \setminus S}\{d_{ij}\}$, since it is always needed to select an outgoing arc from each visited node, and cycles are not allowed.

Then we consider the problem of selecting a subset of nodes which are not yet visited, whose overall demand still fits in the residual capacity of the vehicle and whose potential is maximum. For a given vehicle type k, such problem can be formulated as follows:

$$R(S, \chi) = \max \sum_{i \in \mathcal{N} \setminus S} \lambda_i z_i \tag{4.14}$$

$$s.t. \sum_{i \in \mathcal{N} \setminus S} q_i z_i \leq w_k - \chi \tag{4.15}$$

$$z_i \in \{0, 1\} \qquad \forall i \in \mathcal{N} \setminus S \tag{4.16}$$

where S represents the set of visited customers and χ is the sum of their demands; binary variables z_i are 1 if node i is selected, 0 otherwise; constraint (4.15) is a capacity constraint. The objective (4.14) is to maximize the sum of the potential corresponding to the selected nodes.

Computing each $R(S, \chi)$ value requires to solve a 0–1 Knapsack problem [60] [51]; therefore, to speed-up the computation we approximate the potential by computing its continuous relaxation $\bar{R}(S, \chi)$: this can be done in $O(|\mathcal{N}| \log |\mathcal{N}|)$ time; furthermore, the computation of $\bar{R}(S, \chi)$ after each label extension can be done in linear time through very efficient re-optimization techniques.

Therefore, in the heuristic pricing algorithm a label $l' = (S', \chi', \tau', \delta', \rho', C', i)$ dominates a label $l'' = (S'', \chi'', \tau'', \delta'', \rho'', C'', i)$ if the following conditions are met:

$$
\begin{aligned}
&(\tilde{a}) \quad \bar{R}(S', \chi') \geq \bar{R}(S'', \chi'') \\
&(\tilde{b}) \quad \chi' \leq \chi'' \\
&(\tilde{c}) \quad \tau' \leq \tau'' \\
&(\tilde{d}) \quad \delta' \leq \delta'' \\
&(\tilde{f}) \quad C' \leq C'' \\
&(\tilde{g}) \quad \delta' - \rho' \leq \delta'' - \rho''
\end{aligned}
$$

and at least one of these inequalities is strict. This results in a much larger number of dominations and reduces computational time; on the other hand, these conditions are necessary but not sufficient to allow dominance, and therefore we loose optimality guarantees on the best path found.

Finally, by changing in sign each value $\bar{R}(S, \chi)$ we also obtain a lower bound on the improvement one can get by extending each label. Therefore, given the reduced cost ϕ^* of the best $s - v$ path found so far, any label having

$$
C - \bar{R}(S, \chi) + \min_{k \in \mathcal{K}} \{\mu_k\} + \min_{h \in \mathcal{H}} \{\sigma_h\} - \frac{\lambda_i}{2} \geq \phi^*
$$

is fathomed, because it cannot provide better solutions.

4.3.1.3 Greedy.

In the greedy pricing algorithm a single label $(S, \chi, \tau, \delta, \rho, C, i)$ is considered and iteratively extended to a single node with a nearest neighbor policy. Given a vehicle type k, we first compute the set $\mathcal{R}$ of reachable customers, that is the set of nodes $j \in \mathcal{N}$ satisfying the following conditions

$$
\begin{aligned}
& j \notin S \\
& \chi + q_j \leq w_k \\
& \max\{\tau + s_i + t_{ij}, a_j\} \leq T \\
& \delta + (s_i + t_{ij}) - \min\{\rho - \max\{a_j - (\tau + s_i + t_{ij}), 0\}, 0\} \leq D.
\end{aligned}
$$

If $\mathcal{R} = \emptyset$ we close the path going back to the depot and we stop the search. Otherwise, among the customers in $\mathcal{R}$, we select the one that minimizes the path cost, that is

$$\bar{j} \in \operatorname{argmin}_{j \in \mathcal{R}} \{d_{ij} - \lambda_j\},$$

we extend the label to node $\bar{j}$ using the same update rules described for the exact pricing algorithm, and we iterate.

It is still possible to simultaneously find routes for the different vehicle types: if the value $\chi + q_{\bar{j}}$ exceeds the capacity w_k of some vehicle types, we generate a valid route for these type by closing the path with an arc from i to v. Then, if the capacity of the largest vehicle is not reached, we add $\bar{j}$ to the path and iterate.

This greedy algorithm is repeated for each depot $h \in \mathcal{H}$.

We tried different policies for selecting the nearest neighbor at each iteration, without observing substantial differences.

4.3.2 2-path inequalities

To strengthen the lower bound we add violated 2-path inequalities. These cuts have been introduced by Kohl et al. [52] for the CVRPTW, extending an

idea of Laporte et al. [55]. In this paper we use the same heuristic separation algorithm described in [52]. We search for a subset $\mathcal{S} \subset \mathcal{N}$ such that the customers in $\mathcal{S}$ are served by less than 2 vehicles in the fractional solution of the master problem, but require at least 2 vehicles in any integer solution. In particular, this property holds for a set $\mathcal{S}$ in the following two cases: (a) if the capacity of the largest vehicle is not sufficient to satisfy the demand of all the customers in $\mathcal{S}$, (b) if there is no tour including all customers in $\mathcal{S}$ that respects their time windows, and therefore does not exceed the maximum length T. While property (a) can simply be checked by inspection, property (b) requires to compute a feasible Hamiltonian tour of customers in $\mathcal{S}$, that is to solve an instance of TSP with Time Windows. However, when $|\mathcal{S}|$ is small the TSPTW feasibility problem can be easily solved by dynamic programming (see Dumas et al. [39]). We remark that the property holds in other cases as well, but these two are the only easily computable ones.

A 2-path inequality can take two forms. The first one is the following:

$$\sum_{h\in\mathcal{H}}\sum_{k\in\mathcal{K}}\sum_{r\in\Omega_{hk}} \alpha_{r\mathcal{S}} x_r \geq 2 \tag{4.17}$$

where the coefficient $\alpha_{r\mathcal{S}}$ is equal to the number of arcs (i, j) used by the route r such that $i \in \mathcal{S}$ and $j \notin \mathcal{S}$. Coefficients $\alpha_{r\mathcal{S}}$ might be larger than 1, since it is not needed to the nodes in $\mathcal{S}$ to be visited consecutively.

The second form is:

$$\sum_{h\in\mathcal{H}}\sum_{k\in\mathcal{K}}\sum_{r\in\Omega_{hk}|\sum_{i\in\mathcal{S}} a_{ir}\geq 1} x_r \geq 2. \tag{4.18}$$

Since $\sum_{i\in\mathcal{S}} a_{ir} \geq 1$ for each column encoding a path visiting a customer in $\mathcal{S}$, the inequality forces to select at least two such paths. One can show that any fractional solution satisfying inequalities (4.18) also satisfies inequalities (4.17); in fact, each column r whose variable does not appear in inequality (4.17) has no incoming arcs in $\mathcal{S}$, hence no node in $\mathcal{S}$ is visited, and the variable does not appear in inequality (4.18) either; at the same time whenever a variable appears in (4.18) it also appears in (4.17) with a coefficient not

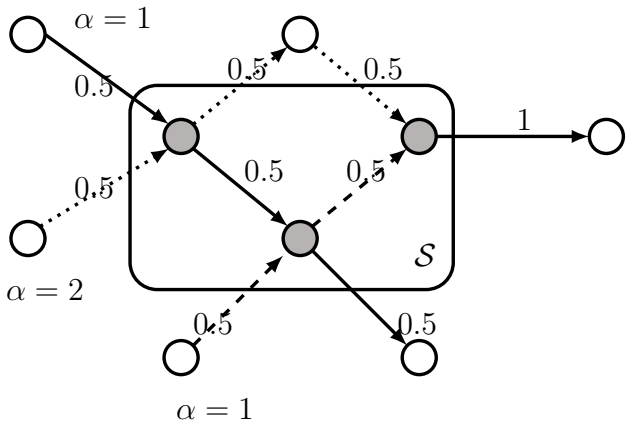

Figure 4.4: Partial fractional solution covering eight nodes of a sample instance.

smaller than (4.18), because it is always needed to select at least an incoming arc in $\mathcal{S}$ to visit its nodes. In the same way, it is easy to show that there exist fractional solutions satisfying inequalities (4.17) but violating inequalities (4.18). Therefore, the bound obtained by introducing (4.18) can be stronger than that found after introducing (4.17).

For example, let us consider Figure 4.4, where three routes (bolded, dotted and dashed arcs), visit a subset $\mathcal{S}$ made by gray nodes. Bolded and dashed routes enter $\mathcal{S}$ once, and have therefore coefficient $\alpha = 1$, while dotted route interleaves nodes inside and outside of $\mathcal{S}$, entering $\mathcal{S}$ twice and having thus $\alpha = 2$. Each route is fractionally selected with value 0.5: the corresponding amount of flow is reported next to each arc. The total incoming flow in $\mathcal{S}$ is 2.0, but the fractional number of routes serving $\mathcal{S}$ is 1.5. Hence an inequality in the form (4.17) is respected, while the corresponding inequality in form (4.18) is violated.

We embed the dynamic generation of these inequalities in a column generation scheme separating them at the end of the column generation iterations. Whenever violated cuts are found, column generation is executed again. This price-and-cut loop is repeated until neither negative reduced cost columns nor violated cuts are found.

When we add 2-path inequalities to the master problem, they induce a new set of dual variables, which must be taken into account in the solution of the pricing problem. Let $\pi \geq 0$ be the dual vector associated with these inequalities. When the weak form (4.17) is used, the expression of the reduced costs of the routes must be modified by subtracting a term $\alpha_{r\mathcal{S}}\pi(\mathcal{S})$ for each 2-path inequality. This can simply be obtained considering modified arc costs

$$\hat{d}_{ij} = d_{ij} - \sum_{\mathcal{S}: i \in \mathcal{S}, j \in \mathcal{N} \setminus \mathcal{S}} \pi(\mathcal{S}).$$

This does not change the structure of the pricing problem. On the contrary, when the strong form (4.18) is used each inequality (4.18) requires to introduce a new resource in the state of the dynamic programming algorithm. Such resource is initialized at 1; after the first visit to a customer in $\mathcal{S}$, the resource is set to 0, and the reduced cost of the label is decreased by the value of the corresponding dual variable. No label having one of these resources at 0 can dominate a label having the corresponding resource at 1. Hence, the pricing problem becomes more and more time consuming as these inequalities are introduced.

In Section 4.4 we compare different ways of handling 2-path inequalities in our algorithm.

4.3.3 Branching strategy

We use two branching policies, described hereafter; the tree search policy is discussed in Section 4.4.

Branching on the number of vehicles. Let $\bar{x}_r$ be the (possibly fractional) value of each variable x_r in the optimal MP fractional solution and let $y_k = \sum_{r \in \Omega_k} \bar{x}_r$ be the (possibly fractional) number of vehicles of type k in such solution. We select the vehicle type $\bar{k}$ whose corresponding $y_{\bar{k}}$ variable has its fractional part closest to 0.5; then we perform binary branching, im-

posing to use at least $\lceil y_{\bar{k}} \rceil$ vehicles of type $\bar{k}$ in one child node and no more than $\lfloor y_{\bar{k}} \rfloor$ in the other.

These branching decisions are handled as follows: first, we modify constraints (4.3) as follows

$$l_k \leq \sum_{r \in \Omega_{hk}} x_r \leq u_k \quad \forall k \in \mathcal{K} \tag{4.19}$$

then we set $l_{\bar{k}} = \lceil y_{\bar{k}} \rceil$ in one child node, $u_{\bar{k}} = \lfloor y_{\bar{k}} \rfloor$ in the other.

The advantage of this branching technique is to leave the pricing subproblem unchanged: dual variables μ_k still appear as constants in the objective function of the pricing problem, but they are now unrestricted in sign. On the other hand constraints (4.19) belong to the MP, which is solved as a linear programming problem, and tightening them usually results in rather weak improvements in the lower bound.

Branching on arcs. When y_k is integer for all vehicle types $k \in \mathcal{K}$, we apply a different branching rule which forbids the use of some arcs. We choose the customer $i \in \mathcal{N}$ that is split among the largest number of routes in the optimal fractional solution of MP and we forbid half of its outgoing arcs to be used in the first child node, and the other half to be used in the second child node. To handle these branching decisions in the pricing problem it is enough to set travel time of forbidden arcs to $+\infty$.

4.4 Experimental analysis

Implementation and hardware. Our algorithms have been implemented in C++, using SCIP 1.1 [1] as a branch-and-cut-and-price framework. Our version of SCIP embeds the CPLEX 11 implementation of the simplex algorithm to solve LP subproblems, and automatically switches between primal and dual simplex, depending on the characteristics of each instance. SCIP performs advanced management of the column and row pools, including their removal and re-insertion. We turned off preprocessing and automatic cut

generation features, but kept all other parameters at their default values, including the decision tree search policy.

We also experimented on introducing stabilization techniques and using the barrier algorithm to solve the LP subproblems.

In both cases we observed a reduction on the number of column generation iterations needed to converge, but the overall performance improvement was negligible. This is because stabilization and the barrier methods help to reduce the number of initial and useless column generation iterations, but in our case most of the CPU time was spent during few final column generation iterations, when the exact dynamic programming algorithm is called. The use of our pricing heuristics can be considered itself as a useful tool to overcome stability problems.

The results reported in this section are obtained using a single core of an Intel Core 2 Duo 3GHz workstation, equipped with 2GB of RAM, running Linux OpenSuse 11.

Benchmark instances. To test our algorithm we used three datasets.

- Dataset 1 is composed by instances for the HVRPTW proposed by Liu and Shen [58], derived from the VRPTW instances of Solomon [83]. The original dataset contains 56 problems. Each problem has 100 customers, whose position are generated in the Euclidean plane, and the travel time between nodes is equal to their Euclidean distance. These instances are divided in 6 classes, depending on the geographical distribution of the customers: random (classes R1 and R2), clustered (C1 and C2) or semi-clustered (RC1 and RC2). Sets R1, C1 and RC1 have a short scheduling horizon and allow for a small number of customers per route. Sets R2, C2 and RC2 have a long scheduling horizon and allow for a large number of customers per route.

  Liu and Shen adapted Solomon's problems to the heterogeneous fleet version by considering vehicles with different capacities. For each of the 6 classes, 3 sets of fixed costs for the vehicles were considered,

obtaining a total of 168 instances. The details of these instances are reported in Table 4.1: we indicate a vehicle type in each row and a customer distribution and demand type scenario in each column; each entry of the table represents the fixed cost of a particular vehicle type in each customer scenario. The table also includes the capacity of each vehicle type in each different scenario.

Since several of these instances are too hard to be solved to proven optimality, we created two new sets of instances by restricting them respectively to the first 50 and 25 customers in the data files. All these instances have a single depot, no constraints on maximum route duration and no limits on the number of vehicles.

Therefore, our Dataset 1 is composed by a total of 504 instances.

- Dataset 2 includes 504 more instances, obtained by simply adding a second depot to each instance in Dataset 1. The coordinates of this depot are randomly generated with uniform probability distribution in the smallest rectangle containing all the customers.

- Dataset 3 contains four instances for the MDVRPTW from Cordeau et al. [17]. These are the smallest in their dataset and include only one type of vehicle. In Table 4.2 we report for each instance in the dataset the number of customers, the number of depots, the maximum allowed duration of the route and the capacity of the vehicle.

All instances considered are symmetric.

Furthermore, we considered the MDHVRPTW instance discussed in [34]. As presented in [9], this instance turned out to be very easy to solve using our algorithms, and therefore we omit further detail.

In the remainder we present the outcome of three different sets of experiments. In Subsection 4.4.1 we studied the trade-off between the computing time and the tightness of the lower bounds obtained at the root nodes; in Subsection 4.4.2 we present the performance of the overall branch-and-cut-and-price algorithm for the exact optimization of the three datasets; in Sub-

section 4.4.3 we discuss the effectiveness of the same algorithm when it is used as a column generation-based heuristic technique.

4.4.1 Lower Bounds

It is known that by dropping elementary conditions on the path, that is running the exact pricing algorithm with decremental state space relaxation and fixing the core $\tilde{\mathcal{N}} = \emptyset$, the RCESPP becomes more tractable [72]. At the same time, paths containing cycles are super-optimal, and by introducing the corresponding infeasible columns in the RMP the bound is weakened. Nevertheless, the generation of valid inequalities can help in filling the same duality gap, and there is a trade-off between the hardness of the pricing problem, the feasibility of the columns which populate the RMPs and the strength of the cutting planes introduced.

Therefore, in the first set of experiments we compared the dual bounds that can be obtained at the root node of the branch-and-bound tree with different combinations of pricing and cutting strategies.

We considered two pricing strategies: (StrongPrice) to generate columns using the greedy pricing algorithm, the heuristic dynamic programming pricing algorithm and the exact dynamic programming pricing algorithm in sequence, as described in Section 4.3.1; (WeakPrice) to generate columns by relaxing the elementary path conditions, and using the relaxed pricing procedure described above.

At the same time, we considered three cut generation strategies: (NoCut) without 2-path inequalities; (WeakCut) cuts in the weak form (4.17); (StrongCut) cuts in the strong form (4.18). In strategies WeakCut and StrongCut we used the heuristic described in [52] to perform cut separation.

A time limit of one hour was imposed to each test.

Tables 4.3 - 4.5, 4.6 - 4.8 and 4.9 - 4.11 show the results we obtained on the instances of Dataset 1 using respectively NoCut, WeakCut or StrongCut. Each table has one row for each class of instances and their columns indicate

the class name, the size and the total number of instances for which the time limit was not exceeded, the average number of column generation iterations needed to converge, the average number of generated cuts, the average duality gap with respect to the best known integer solution and the average time spent during column generation for the two pricing strategies.

As expected, the CPU time increases as the size of the instance increases; the duality gap is not always small: since it is computed using best known primal solutions, this might also depend on poor primal bounds.

We notice that even finding a valid bound within the time limit is sometimes a difficult task; this becomes more evident as more aggressive cut handling strategies are applied. For instance, a valid bound is obtained on 13 C2 instances with $|\mathcal{M}| = 100$ using StrongPrice, but the price-and-cut loop terminated only 12 times using WeakCut and only 10 times using StrongCut (see Tables 4.5, 4.8 and 4.11).

No relevant difference can be observed by varying the fixed costs of the vehicles (instances a, b and c in each class). The most noticeable performance gap on Dataset 1 is between Type 1 (R1, C1, RC1) and Type 2 (R2, C2, RC2) instances: the computation successfully terminates on almost all Type 1 instances, with both pricing strategies and with all cut handling strategies, while timeout expires more often on Type 2 instances as the size of the instances increases. In fact, Type 2 instances have loose capacity constraints, long time horizons and large time windows; this means longer feasible routes and far more labels to manage for the pricing algorithm.

Instances in Class C2 show a particular characteristic: they require more column generation iterations and less computing time, and the final duality gap is larger. This can be explained because fractional solutions can exploit both the cluster structure and the capacity of small vehicles, while integer solutions in general cannot.

Comparing the results reported on the same rows in each table (different pricing strategies, same cutting strategy) one can see that WeakPrice is much faster: on average more column generation iterations are needed to converge,

but the required CPU time is a small fraction of that required by StrongPrice. In fact, more columns are feasible using WeakPrice: a larger number of columns has to be explored before reaching convergence, even though each of them is easier to generate. However, when a valid bound is found using StrongPrice, the duality gap is significantly smaller. This behavior is observed for all cut strategies.

Comparing the results reported in the same block in different tables (same pricing strategy, different cutting strategy), one can see that few cuts are generated with StrongPrice; however, as expected, many more are found with WeakPrice. Cuts are especially effective for WeakPrice in class RC; StrongCut is useful in particular in small R2 and C2 instances, when it is combined with StrongPrice.

For some instances the use of cuts improves the bound and reduces the time as well. A preprocessing rule was responsible for this counter intuitive result: using NoCut when a dual variable λ_i is 0, the corresponding customer i is removed from the pricing problem ; when cuts are introduced, π dual variables can make it appealing to visit such customers: this helps heuristic pricing in finding "good" columns earlier, and therefore in speeding up the computation.

We chose to use StrongPrice with WeakCut as the best trade-off. In this way we were able to complete the price-and-cut loop at the root node within the time limit for all instances but one, still obtaining tight dual bounds and allowing the heuristic generation of good columns.

4.4.2 Branch-and-cut-and-price

In the second set of experiments we let our algorithm perform an exact optimization of each instance in the three Datasets. For these tests we used StrongPrice, heuristic separation of inequalities (4.17), WeakCut and the branching policy described in Subsection 4.3.3. We turned off all SCIP general purpose heuristics, but *fracdiving* and *rens* which proved effective in preliminary tests.

A time limit of one hour was imposed to each test.

Tables 4.12, 4.13, 4.14 show the results we obtained on the instances of Dataset 1, Dataset 2 and Dataset 3 respectively. As before, each table has one row for each class of instances. In the columns of each table we report in turn the class name, the number of instances solved to proven optimality within the time limit, the average optimality gap on the instances for which optimality was not proved and the average solution time for the solved instances. We remark that for some instances the time limit did not allow even to find an initial dual bound, as reported in Tables from 4.3 to 4.11. These have been excluded from this test.

Type 2 classes confirm to be much harder than Type 1. The exact algorithm is fully effective only on small instances: almost all Type 1 classes with 25 customers, 1/3 of those with 50 customers and only 5 of the largest ones are solved to proven optimality. Much less Type 2 instances are solved. On small instances the algorithm is also fast: few seconds are enough to complete each computation. An unexpected bad behavior is observed in class RC1a. By comparing Table 4.12 with Tables 4.6 - 4.8, and Table 4.15a we notice that the difficulty consist in finding a good primal bound.

Comparing Tables 4.14, 4.12 and 4.13, we conclude that handling multiple depots is not an issue in our algorithm. In fact, even the instance with 72 customers and 6 depots of Dataset 3, involving identical vehicles, can be easily solved. Indeed, our feeling is that managing a heterogeneous fleet is what really complicates the problem: multiple depots naturally favor geometric partitions of customers, and therefore the generation of less overlapping routes, without complicating the pricing algorithm. This is not the case for heterogeneous vehicles: good routes for vehicles with different capacity can be substantially different, and there are no geometric properties to exploit.

4.4.3 Column generation-based heuristics

While, to the best of our knowledge, no exact algorithm for the MDHVRP has been proposed so far, in the literature a few algorithms showed to perform

very well in heuristically solving the problem.

Hence, in the third set of experiments we tuned our algorithm to produce good integer solutions faster: we truncated the optimization process at the root node of the branching tree; then, we tried to combine the columns in the final RMP in a good integer solution. In particular, we considered two adaptations.

- In Algorithm iRMP, the column generation process is stopped after a given time limit α, or when no negative reduced cost column is found; then, integrality conditions on the variables of the RMP are restored, and CPLEX 11 ILP solver is invoked to optimize the ILP problem obtained in this way. CPLEX optimization is stopped after a time limit β, or when optimality is proved.

- In Algorithm Fix&Gen, column generation is performed for up to a given time limit γ; then, each variable taking value 1 in the final RMP fractional solution, and the remaining variable with highest fractional value, are fixed to 1. The reduced problem obtained in this way is re-optimized by performing again column generation for up to a time limit γ. This column generation and fixing process is iterated, until a full integer solution is found.

We compared the results of Algorithms iRMP and Fix&Gen with those obtained by the algorithms of Liu and Shen (LS) [58], Belfiore and Fávero (BF) [6], Dell'Amico et al (DAMPV) [21]. We restricted our test to Dataset 1, since algorithms LS, BF and DAMPV have been tested on these instances only.

We remark that, although no a-priori optimality guarantee is given by this procedure, the dual bound obtained by column generation allows us to control the quality of the final solution. This cannot be done in any of the LS, BF and DAMPV algorithms.

In Table 4.15a we report the results of our experiments. The table contains a row for each class of instances in the dataset, and is composed by different

blocks, one for each algorithm, as indicated in the leading row. The columns in the first block report the average value of the best solution found by LS and the average CPU time spent; each subsequent block includes the percentage improvement with respect to the value of the LS solution and the average CPU time spent. The CPU time values for algorithms LS, BF and DAMPV are taken from the original papers: according to [35] the CPU time reported in LS, BF, and DAMPV should be scaled respectively by a factor 20, 2 and 10 in order to be equivalent to CPU time in our computer.

First, in order to get roughly the same computing time of DAMPV, we tried to use iRMP algorithm by fixing α to 10 minutes and β to 5 minutes. In Type 1 instances iRMP is consistently the best, especially in terms of average solutions cost. In classes R2a and C2a, however, iRMP performs poorly.

We found out that limit α affects the performance of iRMP more than limit β: since columns in iRMP are generated to be good for fractional solutions, their quality for integer solutions is not guaranteed, especially during the first column generation iterations. Therefore, we also tested iRMP by increasing both α and β to 60 minutes. No substantial difference was observed in Type 1 classes, since the time limit was almost never exceeded; iRMP improved solutions on Type 2 classes, but still performed worse than other methods in classes R2a and C2a.

Then, we tested Fix&Gen setting time limit γ to 1 minute. Indeed, this heuristic is both faster and more accurate, especially in Type 2 classes. We also tried to increase the time limit γ up to 3 minutes, and we verified that the quality of the solutions keeps improving.

Finally, we tested Fix&Gen by fixing γ to 60 minutes for the first iteration and to 3 minutes for the subsequent iterations. On the average the average quality of solutions improved, especially in difficult classes (i.e. R2 and C2), even though in some classes (i.e. R1b, C1b, C1c and RC1c) this is not the case. This result is obtained at the expense of a significant increase in computing time.

We also tested our algorithms on Dataset 3, which is difficult because all vehicles have to be used in any feasible integer solution. This makes, in fact,

local search more appropriate than iterative route generation. Furthermore, the fixing of a column is likely to affect the overall solution, or even to make it impossible to obtain a feasible one; this effect is more evident when time limit γ is small, and is therefore needed to fix one of the columns produced in early column generation iterations. Nevertheless, we still obtain good results, matching the best known solution in 2 instances and obtaining solutions whose value is within 4% of the best known.

The choice of the right heuristic is obviously instance-dependent. However, Fix&Gen seems to give the best trade-off between quality of the solution and computational effort, especially setting γ between 1 and 3 minutes.

4.5 Conclusions

We have exploited the intrinsic flexibility of a known mathematical programming framework to solve an important optimization problem in distribution logistics, namely the MDHVRPTW.

Besides being flexible the method allows for different combinations of cutting and pricing strategies. We have presented an experimental evaluation of these strategies in terms of lower bound tightness and computing time. This suggests to investigate mixed strategies: for instance it may be appealing to initialize the RMP using WeakPrice, repair the columns in the RMP by removing cycles, then switch to StrongPrice, including WeakCut in the price-and-cut loop. A switch to StrongCut has to be considered only at the end of this process, if the duality gap is still large. Since the number of column generation iterations using WeakPrice is sometimes large, the introduction of a stabilization technique seems appropriate [77]. Since a large amount of computing time is spent in the few calls to the exact dynamic programming algorithm, the use of acceleration techniques like early branching [25] may be beneficial.

When the size of the instance prevents the algorithm to find provably optimal solutions, it is still possible to use the algorithm as a column generation-based

heuristic, which is competitive and often better than constructive or local search-based heuristics from the literature, with the additional advantage of providing also dual bounds.

Finally our experimental results reveal that what makes the MDHVRPTW difficult is the number of different vehicle types rather than the presence of multiple depots. The tightness of the time windows also plays an important role as already highlighted in the literature.

Vehicle	Capacity	R1a	R1b	R1c
A	30	50	10	5
B	50	80	16	8
C	80	140	28	14
D	120	250	50	25
E	200	500	100	50
Vehicle	Capacity	C1a	C1b	C1c
A	100	300	60	30
B	200	800	160	80
C	300	1350	270	135
Vehicle	Capacity	RC1a	RC1b	RC1c
A	40	60	12	6
B	80	150	30	15
C	150	300	60	30
D	200	450	90	45
Vehicle	Capacity	R2a	R2b	R2c
A	300	450	90	45
B	400	700	140	70
C	600	1200	240	120
D	1000	2500	500	250
Vehicle	Capacity	C2a	C2b	C2c
A	400	1000	200	100
B	500	1400	280	140
C	600	2000	400	200
D	700	2700	540	270
Vehicle	Capacity	RC2a	RC2b	RC2c
A	100	150	30	15
B	200	350	70	35
C	300	550	110	55
D	400	800	160	80
E	500	1100	220	110
F	1000	2500	500	250

Table 4.1: Details of instances in Dataset 1 and Dataset 2

instance	customers	depots	duration	capacity
pr01	48	4	500	200
pr02	96	4	480	195
pr07	72	6	500	200
pr08	144	6	475	190

Table 4.2: Details of instances in Dataset 3

Set	Size	StrongPrice				WeakPrice			
		solved	iterations	LB	time	solved	iterations	LB	time
R1a	25	12	34.00	0.98%	0.11	12	63.92	1.58%	0.12
R1b	25	12	43.25	0.78%	0.11	12	80.50	2.25%	0.17
R1c	25	12	43.75	0.65%	0.11	12	84.83	2.29%	0.19
R1	25	36	40.33	0.80%	0.11	36	76.42	2.04%	0.16
C1a	25	9	38.56	7.87%	0.20	9	60.44	8.01%	0.11
C1b	25	9	50.00	6.30%	0.47	9	72.00	6.77%	0.11
C1c	25	9	53.22	4.20%	0.26	9	82.78	5.74%	0.16
C1	25	27	47.26	6.12%	0.31	27	71.74	6.84%	0.13
RC1a	25	8	29.88	7.29%	0.24	8	60.25	9.16%	0.12
RC1b	25	8	46.50	2.77%	0.45	8	78.25	8.64%	0.18
RC1c	25	8	52.25	1.52%	0.47	8	82.13	7.76%	0.18
RC1	25	24	42.88	3.86%	0.38	24	73.54	8.52%	0.16
R2a	25	5	143.60	24.46%	563.69	11	219.00	39.83%	39.41
R2b	25	7	124.71	4.94%	41.22	11	119.86	24.69%	3.24
R2c	25	10	113.70	2.70%	200.37	11	116.30	21.89%	3.25
R2	25	22	124.00	8.36%	232.30	33	140.77	26.86%	11.47
C2a	25	6	181.33	26.95%	554.43	8	136.83	29.93%	2.99
C2b	25	7	194.14	21.34%	164.78	8	142.86	29.01%	6.57
C2c	25	7	186.71	11.59%	30.01	8	130.00	21.34%	2.20
C2	25	20	187.70	19.61%	234.50	24	136.55	26.60%	3.96
RC2a	25	6	36.83	7.48%	8.59	8	91.83	12.76%	0.35
RC2b	25	6	54.00	0.00%	87.59	8	109.17	19.73%	1.07
RC2c	25	7	61.29	0.00%	355.52	8	138.14	27.75%	3.36
RC2	25	19	51.26	2.36%	161.35	24	114.37	20.48%	1.69

Table 4.3: Quality of the bound without cut generation, size 25.

Set	Size	StrongPrice				WeakPrice			
		solved	iterations	LB	time	solved	iterations	LB	time
R1a	50	12	45.17	2.24%	1.17	12	143.08	2.82%	0.97
R1b	50	12	77.00	2.43%	2.30	12	193.00	3.96%	2.02
R1c	50	12	91.92	2.41%	2.76	12	203.00	4.37%	2.15
R1	50	36	71.36	2.36%	2.08	36	179.69	3.72%	1.71
C1a	50	9	74.67	5.04%	2.29	9	119.89	5.14%	0.88
C1b	50	9	88.67	6.74%	2.43	9	149.89	7.03%	0.68
C1c	50	9	95.11	6.14%	3.87	9	178.67	7.59%	1.12
C1	50	27	86.15	5.97%	2.86	27	149.48	6.59%	0.89
RC1a	50	8	65.13	12.36%	2.28	8	133.00	13.58%	0.71
RC1b	50	8	99.63	8.11%	10.72	8	167.50	11.32%	1.24
RC1c	50	8	105.38	8.56%	9.12	8	167.63	12.08%	1.14
RC1	50	24	90.04	9.68%	7.37	24	156.04	12.33%	1.03
R2a	50	1	453.00	10.83%	137.93	2	674.00	16.62%	61.89
R2b	50	3	262.67	13.41%	550.47	11	411.33	27.34%	43.77
R2c	50	3	149.33	6.65%	105.91	11	254.33	20.17%	12.39
R2	50	7	241.29	10.15%	301.01	24	381.57	22.73%	32.91
C2a	50	4	498.75	26.54%	441.47	6	376.25	28.72%	35.94
C2b	50	5	246.60	17.17%	128.49	7	262.00	22.38%	14.63
C2c	50	6	250.00	3.77%	591.72	8	274.67	8.52%	26.28
C2	50	15	315.20	14.31%	397.24	21	297.53	18.53%	24.97
RC2a	50	4	72.50	1.62%	47.38	8	208.50	5.86%	3.25
RC2b	50	4	94.00	0.00%	89.39	8	305.75	16.13%	10.62
RC2c	50	5	125.60	0.00%	470.13	6	448.80	24.68%	238.43
RC2	50	13	99.54	0.50%	222.90	22	330.85	16.26%	95.97

Table 4.4: Quality of the bound without cut generation, size 50.

Set	Size	StrongPrice				WeakPrice			
		solved	iterations	LB	time	solved	iterations	LB	time
R1a	100	12	75.50	3.47%	77.41	12	311.00	3.89%	10.70
R1b	100	11	163.36	4.92%	365.26	12	473.73	6.07%	49.75
R1c	100	12	200.33	5.69%	427.59	12	543.58	7.30%	66.08
R1	100	35	145.91	4.69%	287.94	36	441.89	5.74%	41.96
C1a	100	9	147.22	6.68%	15.88	9	221.44	6.77%	4.47
C1b	100	9	200.33	7.31%	406.34	9	309.44	7.83%	7.80
C1c	100	8	157.13	5.98%	69.12	9	359.88	6.98%	5.71
C1	100	26	168.65	6.68%	167.42	27	294.50	7.20%	6.01
RC1a	100	8	147.75	4.69%	768.52	8	332.75	5.61%	24.97
RC1b	100	8	177.50	8.68%	297.20	8	453.38	11.15%	43.40
RC1c	100	8	179.88	8.69%	217.53	8	451.50	11.81%	37.14
RC1	100	24	168.38	7.35%	427.75	24	412.54	9.52%	35.17
R2a	100	0	-	-	-	0	-	-	-
R2b	100	1	676.00	9.95%	1917.91	1	2969.00	15.81%	979.73
R2c	100	1	366.00	6.60%	578.42	2	1584.00	12.89%	291.04
R2	100	2	521.00	8.28%	1248.17	3	2276.50	14.35%	635.39
C2a	100	2	1240.50	10.56%	497.32	6	1239.50	10.95%	298.23
C2b	100	5	972.60	9.67%	1274.22	7	1318.40	11.70%	281.59
C2c	100	6	996.83	9.99%	1163.41	7	1085.83	12.55%	168.31
C2	100	13	1025.00	9.96%	1103.55	20	1198.92	11.98%	231.87
RC2a	100	3	214.67	4.81%	1586.53	7	450.33	6.96%	97.59
RC2b	100	1	168.00	3.51%	331.60	4	710.00	11.97%	93.92
RC2c	100	1	228.00	5.58%	402.83	4	705.00	16.64%	102.35
RC2	100	5	208.00	4.70%	1098.81	15	553.20	9.90%	97.81

Table 4.5: Quality of the bound without cut generation, size 100..

Set	Size	StrongPrice					WeakPrice				
		solved	iterations	cuts	LB	time	solved	iterations	cuts	LB	time
R1a	25	12	35.17	1.75	0.73%	0.10	12	66.67	2.25	1.32%	0.11
R1b	25	12	44.25	1.25	0.62%	0.12	12	83.17	2.08	2.06%	0.15
R1c	25	12	43.58	0.50	0.47%	0.12	12	87.83	1.33	2.06%	0.16
R1	25	36	41.00	1.17	0.61%	0.11	36	79.22	1.89	1.81%	0.14
C1a	25	9	37.33	0.00	7.87%	0.11	9	59.22	0.11	8.01%	0.10
C1b	25	9	48.67	0.00	6.30%	0.46	9	74.00	0.56	6.77%	0.10
C1c	25	9	51.22	0.22	4.19%	0.58	9	90.89	1.56	5.72%	0.14
C1	25	27	45.74	0.07	6.12%	0.38	27	74.70	0.74	6.83%	0.11
RC1a	25	8	32.25	0.38	6.95%	0.20	8	69.50	6.13	8.64%	0.11
RC1b	25	8	47.00	0.38	1.62%	0.58	8	91.88	13.25	6.83%	0.20
RC1c	25	8	49.50	0.38	0.13%	0.72	8	97.75	13.63	5.48%	0.22
RC1	25	24	42.92	0.38	2.90%	0.50	24	86.38	11.00	6.98%	0.18
R2a	25	5	150.60	0.00	24.46%	403.66	11	245.60	0.60	41.58%	61.28
R2b	25	7	116.43	0.14	4.92%	47.06	11	123.29	0.00	25.42%	4.29
R2c	25	10	102.70	0.00	2.70%	147.20	11	112.60	0.00	22.10%	3.73
R2	25	22	117.95	0.05	8.35%	173.62	33	146.23	0.14	27.58%	16.99
C2a	25	6	185.50	0.17	22.84%	487.55	8	125.50	0.17	25.81%	3.40
C2b	25	7	164.43	0.14	20.38%	229.06	8	174.71	0.14	28.04%	6.33
C2c	25	7	160.57	0.00	11.59%	17.20	8	124.86	0.00	21.43%	2.95
C2	25	20	169.40	0.10	18.04%	232.46	24	142.50	0.10	25.06%	4.27
RC2a	25	6	33.33	0.00	7.48%	7.86	8	95.17	2.67	12.73%	0.49
RC2b	25	6	61.83	0.00	0.00%	96.09	8	117.00	1.43	16.96%	1.62
RC2c	25	7	52.29	0.00	14.29%	90.98	8	150.43	3.14	28.14%	4.53
RC2	25	19	49.32	0.00	7.62%	66.35	24	122.15	2.40	19.60%	2.30

Table 4.6: Quality of the bound with WeakCut, size 25.

Set	Size	StrongPrice					WeakPrice				
		solved	iterations	cuts	LB	time	solved	iterations	cuts	LB	time
R1a	50	12	46.75	0.50	2.22%	0.56	12	144.67	2.33	2.78%	0.94
R1b	50	12	80.08	1.00	2.35%	1.89	12	201.92	4.17	3.69%	2.17
R1c	50	12	92.83	0.92	2.31%	2.28	12	214.42	6.17	4.00%	2.32
R1	50	36	73.22	0.81	2.29%	1.58	36	187.00	4.22	3.49%	1.81
C1a	50	9	69.22	0.11	5.04%	1.03	9	120.89	0.44	5.14%	0.81
C1b	50	9	87.33	0.00	6.74%	1.44	9	148.56	2.22	7.03%	0.63
C1c	50	9	93.78	0.00	6.14%	2.44	9	185.33	5.67	7.57%	1.08
C1	50	27	83.44	0.04	5.97%	1.64	27	151.59	2.78	6.58%	0.84
RC1a	50	8	66.88	3.25	11.88%	2.62	8	159.13	30.75	12.77%	0.94
RC1b	50	8	103.38	3.00	6.60%	12.00	8	199.38	33.00	9.21%	1.71
RC1c	50	8	98.50	2.50	6.58%	11.30	8	204.38	29.75	9.10%	1.73
RC1	50	24	89.58	2.92	8.35%	8.64	24	187.63	31.17	10.36%	1.46
R2a	50	1	453.00	0.00	10.83%	137.93	2	1125.00	0.00	16.62%	103.43
R2b	50	3	262.67	0.00	13.41%	550.48	11	391.67	0.00	27.34%	46.46
R2c	50	3	149.33	0.00	6.65%	105.91	11	263.00	0.33	20.19%	16.49
R2	50	7	241.29	0.00	10.15%	301.01	24	441.29	0.14	22.74%	41.76
C2a	50	4	499.00	2.00	26.45%	441.49	6	408.75	3.00	28.61%	30.96
C2b	50	5	246.80	1.60	16.87%	128.50	8	277.00	2.40	22.05%	14.43
C2c	50	6	250.17	1.00	3.23%	591.73	8	282.33	1.67	7.96%	23.15
C2	50	15	315.40	1.47	13.97%	397.25	22	314.27	2.27	18.17%	22.33
RC2a	50	4	72.50	0.00	1.62%	47.38	8	240.25	13.75	5.66%	5.60
RC2b	50	4	94.00	0.00	0.00%	89.39	8	330.50	14.00	15.63%	18.78
RC2c	50	5	125.60	0.00	0.00%	470.13	6	493.20	14.80	33.17%	380.38
RC2	50	13	99.54	0.00	0.50%	222.90	22	365.31	14.23	19.31%	153.80

Table 4.7: Quality of the bound with WeakCut, size 50.

Set	Size	StrongPrice					WeakPrice				
		solved	iterations	cuts	LB	time	solved	iterations	cuts	LB	time
R1a	100	12	72.67	1.08	3.47%	4.71	12	310.25	2.67	3.87%	12.99
R1b	100	11	161.00	0.73	4.91%	292.98	12	487.91	5.36	5.98%	66.74
R1c	100	12	192.00	1.08	5.67%	288.37	12	550.58	5.75	7.19%	81.44
R1	100	35	141.34	0.97	4.68%	192.56	36	448.49	4.57	5.67%	53.35
C1a	100	9	144.33	0.11	6.68%	5.77	9	229.89	5.22	6.77%	4.25
C1b	100	9	195.67	0.00	7.31%	251.48	9	313.33	27.22	7.81%	9.89
C1c	100	8	146.75	0.00	5.98%	30.76	9	364.00	65.00	6.90%	6.93
C1	100	26	162.85	0.04	6.68%	98.51	27	300.04	31.23	7.17%	7.02
RC1a	100	8	136.63	4.88	4.60%	632.35	8	378.75	56.00	5.42%	48.14
RC1b	100	8	177.88	5.88	8.39%	385.19	8	520.00	64.13	10.55%	80.55
RC1c	100	8	183.63	5.63	8.36%	267.88	8	516.25	64.00	10.98%	66.34
RC1	100	24	166.04	5.46	7.12%	428.47	24	471.67	61.38	8.98%	65.01
R2a	100	0	-	-	-	-	0	-	-	-	-
R2b	100	1	655.00	0.00	9.95%	2242.91	1	2905.00	14.00	15.63%	1507.09
R2c	100	1	263.00	0.00	6.60%	513.52	2	1545.00	18.00	12.65%	402.84
R2	100	2	459.00	0.00	8.28%	1378.22	3	2225.00	16.00	14.14%	954.97
C2a	100	2	2084.50	0.00	10.56%	1460.67	6	1326.50	17.00	10.93%	317.27
C2b	100	5	1017.40	0.00	9.67%	1636.91	6	1354.00	8.40	11.67%	287.32
C2c	100	5	1017.80	0.00	10.79%	1240.19	7	1075.80	6.20	13.68%	154.60
C2	100	12	1195.42	0.00	10.28%	1442.24	19	1233.50	8.92	12.39%	237.01
RC2a	100	3	208.00	0.00	4.81%	1434.43	7	496.67	13.67	6.89%	143.65
RC2b	100	1	182.00	0.00	3.51%	571.57	3	825.00	34.00	11.52%	192.23
RC2c	100	1	212.00	0.00	5.58%	403.18	3	894.00	39.00	15.90%	214.64
RC2	100	5	203.60	0.00	4.70%	1055.61	13	641.80	22.80	9.62%	167.56

Table 4.8: Quality of the bound with WeakCut, size 100.

Set	Size	StrongPrice					WeakPrice				
		solved	iterations	cuts	LB	time	solved	iterations	cuts	LB	time
R1a	25	12	35.17	1.75	0.73%	0.11	12	66.92	2.17	1.31%	0.11
R1b	25	12	44.17	1.25	0.62%	0.12	12	83.00	1.92	2.03%	0.15
R1c	25	12	43.67	0.50	0.47%	0.12	12	86.92	1.17	1.99%	0.16
R1	25	36	41.00	1.17	0.61%	0.12	36	78.94	1.75	1.78%	0.14
C1a	25	9	37.33	0.00	7.87%	0.10	9	59.33	0.11	8.01%	0.10
C1b	25	9	48.67	0.00	6.30%	0.46	9	74.78	0.89	6.76%	0.10
C1c	25	9	51.22	0.22	4.19%	0.57	9	91.33	1.44	5.68%	0.16
C1	25	27	45.74	0.07	6.12%	0.38	27	75.15	0.81	6.81%	0.12
RC1a	25	8	32.25	0.38	6.95%	0.20	8	70.38	5.38	8.56%	0.15
RC1b	25	8	47.00	0.38	1.61%	0.58	8	92.50	8.88	5.79%	0.44
RC1c	25	8	49.50	0.38	0.13%	0.73	8	100.00	10.63	3.55%	0.59
RC1	25	24	42.92	0.38	2.90%	0.50	24	87.63	8.29	5.97%	0.39
R2a	25	5	150.60	0.00	24.46%	386.89	11	239.80	0.40	38.55%	63.15
R2b	25	7	116.29	0.14	4.86%	47.46	11	123.29	0.00	25.42%	4.41
R2c	25	10	102.70	0.00	2.70%	152.23	11	112.60	0.00	22.10%	3.84
R2	25	22	117.91	0.05	8.33%	172.23	33	144.91	0.09	26.90%	17.50
C2a	25	6	203.17	0.17	22.84%	465.43	8	144.83	0.17	25.81%	3.47
C2b	25	7	180.00	0.14	20.38%	240.37	8	196.57	0.14	28.04%	6.63
C2c	25	7	160.57	0.00	11.59%	17.45	8	124.86	0.00	21.43%	3.00
C2	25	20	180.15	0.10	18.04%	229.87	24	155.95	0.10	25.06%	4.41
RC2a	25	6	33.33	0.00	7.48%	7.93	8	94.50	1.50	12.60%	0.61
RC2b	25	6	61.83	0.00	0.00%	96.41	8	120.83	2.83	17.79%	2.67
RC2c	25	7	52.29	0.00	0.00%	92.98	8	156.57	2.14	25.62%	5.64
RC2	25	19	49.32	0.00	2.36%	67.20	24	125.68	2.16	19.04%	3.11

Table 4.9: Quality of the bound with StrongCut, size 25.

Set	Size	StrongPrice					WeakPrice				
		solved	iterations	cuts	LB	time	solved	iterations	cuts	LB	time
R1a	50	12	46.75	0.50	2.22%	0.56	12	143.83	2.08	2.78%	0.92
R1b	50	12	80.08	1.00	2.34%	1.89	12	201.42	4.42	3.61%	2.19
R1c	50	12	92.83	0.92	2.31%	2.28	12	215.75	5.17	3.89%	2.45
R1	50	36	73.22	0.81	2.29%	1.58	36	187.00	3.89	3.43%	1.85
C1a	50	9	69.22	0.11	5.04%	1.03	9	120.89	0.44	5.14%	0.82
C1b	50	9	87.33	0.00	6.74%	1.44	9	150.44	3.44	7.01%	0.71
C1c	50	9	93.78	0.00	6.14%	2.44	9	185.33	6.33	7.50%	1.32
C1	50	27	83.44	0.04	5.97%	1.64	27	152.22	3.41	6.55%	0.95
RC1a	50	8	66.88	3.25	11.45%	2.62	8	157.13	30.00	12.24%	2.45
RC1b	50	8	103.38	3.00	5.55%	12.00	8	204.88	30.13	6.34%	11.74
RC1c	50	8	98.50	2.50	5.14%	11.30	8	210.00	28.63	6.28%	11.92
RC1	50	24	89.58	2.92	7.38%	8.64	24	190.67	29.58	8.28%	8.70
R2a	50	1	453.00	0.00	10.83%	140.91	2	1125.00	0.00	16.62%	104.26
R2b	50	3	262.67	0.00	13.41%	551.52	11	391.67	0.00	27.34%	47.20
R2c	50	3	149.33	0.00	6.65%	105.84	11	274.00	0.33	19.95%	19.43
R2	50	7	241.29	0.00	10.15%	301.86	24	446.00	0.14	22.64%	43.45
C2a	50	4	498.50	2.00	26.45%	444.72	6	415.75	3.50	28.56%	38.71
C2b	50	5	247.20	1.60	16.87%	129.41	8	283.80	2.40	21.91%	16.40
C2c	50	6	250.00	1.00	3.23%	597.60	8	298.33	2.00	7.79%	25.52
C2	50	15	315.33	1.47	13.97%	400.77	22	324.80	2.53	18.03%	26.00
RC2a	50	4	72.50	0.00	1.62%	49.77	8	235.25	10.75	5.35%	46.05
RC2b	50	4	94.00	0.00	0.00%	90.34	8	404.50	13.75	11.48%	554.58
RC2c	50	5	125.60	0.00	0.00%	474.94	6	576.80	13.00	19.88%	835.26
RC2	50	13	99.54	0.00	0.50%	225.78	22	418.69	12.54	12.83%	506.07

Table 4.10: Quality of the bound with StrongCut, size 50.

Set	Size	StrongPrice					WeakPrice				
		solved	iterations	cuts	LB	time	solved	iterations	cuts	LB	time
R1a	100	12	72.58	1.08	3.47%	4.76	12	310.58	2.58	3.86%	13.12
R1b	100	11	161.00	1.09	4.91%	285.13	12	492.36	5.73	5.93%	77.50
R1c	100	12	192.08	0.92	5.67%	297.77	12	555.92	5.25	7.11%	88.05
R1	100	35	141.34	1.03	4.68%	193.34	36	451.83	4.49	5.62%	59.04
C1a	100	9	144.33	0.11	6.68%	5.76	9	229.67	5.44	6.77%	4.26
C1b	100	9	195.67	0.00	7.31%	265.03	9	315.78	24.56	7.78%	17.13
C1c	100	8	146.75	0.00	5.98%	31.25	9	373.75	50.00	6.77%	120.71
C1	100	26	162.85	0.04	6.68%	103.35	27	303.81	25.77	7.12%	44.55
RC1a	100	7	124.43	4.57	4.55%	419.13	8	383.86	39.00	5.16%	102.48
RC1b	100	8	180.00	6.50	8.13%	441.86	8	540.75	48.38	9.63%	247.13
RC1c	100	8	186.50	5.88	8.02%	273.86	8	539.13	49.00	9.84%	138.26
RC1	100	23	165.35	5.70	7.00%	376.51	24	492.43	45.74	8.34%	165.24
R2a	100	0	-	-	-	-	0	-	-	-	-
R2b	100	1	655.00	0.00	9.95%	2320.71	1	2924.00	8.00	15.52%	2059.69
R2c	100	1	263.00	0.00	6.60%	521.91	2	1585.00	11.00	12.54%	708.73
R2	100	2	459.00	0.00	8.28%	1421.31	3	2254.50	9.50	14.03%	1384.21
C2a	100	2	2084.50	0.00	10.56%	1509.65	6	1352.50	10.50	10.92%	363.50
C2b	100	4	901.50	0.00	9.72%	1149.08	6	1252.00	3.00	11.31%	239.79
C2c	100	4	997.75	0.00	11.01%	694.66	7	1104.75	2.25	13.73%	141.79
C2	100	10	1176.60	0.00	10.40%	1039.42	19	1213.20	4.20	12.20%	225.33
RC2a	100	3	205.50	0.00	4.97%	1244.61	5	434.00	6.50	7.00%	154.82
RC2b	100	1	182.00	0.00	3.51%	572.60	3	1017.00	25.00	10.04%	1952.98
RC2c	100	1	212.00	0.00	5.58%	425.94	2	1005.00	33.00	14.04%	662.10
RC2	100	5	201.25	0.00	4.76%	871.94	10	722.50	17.75	9.52%	731.18

Table 4.11: Quality of the bound with StrongCut, size 100.

class	size	solved	gap	time	set	size	solved	gap	time
R1a	25	12	0.00%	3.59	R2a	25	4	40.35%	265.71
	50	7	2.61%	452.34		50	1	0.00%	1201.98
	100	0	5.04%	-		100	0	-	-
R1b	25	11	0.55%	1.42	R2b	25	5	4.21%	569.78
	50	7	2.91%	186.69		50	1	18.99%	182.77
	100	1	8.28%	271.15		100	0	19.88%	
R1c	25	12	0.00%	1.88	R2c	25	9	6.59%	433.17
	50	8	4.11%	631.20		50	2	18.14%	486.85
	100	2	10.50%	394.75		100	0	9.83%	-
R1	25	35	0.55%	2.32	R2	25	18	13.84%	433.90
	50	22	3.15%	432.85		50	4	18.71%	589.61
	100	3	7.76%	353.55		100	0	14.86%	-
C1a	25	9	0.00%	7.91	C2a	25	5	65.18%	24.35
	50	0	0.67%	-		50	3	36.21%	801.53
	100	0	2.22%	-		100	0	4.77%	-
C1b	25	7	0.79%	118.76	C2b	25	7	0.00%	312.46
	50	1	1.83%	0.00		50	4	26.20%	162.30
	100	0	3.69%	-		100	0	16.41%	-
C1c	25	8	3.32%	5.44	C2c	25	7	0.00%	30.37
	50	1	3.32%	647.69		50	4	0.26%	151.48
	100	2	1.27%	1453.95		100	0	12.05%	-
C1	25	24	1.63%	39.42	C2	25	19	65.18%	132.71
	50	2	1.89%	323.85		50	11	15.73%	332.70
	100	2	2.48%	1453.95		100	0	12.65%	-
RC1a	25	1	3.23%	120.25	RC2a	25	3	5.36%	167.13
	50	0	10.35%	-		50	3	3.38%	683.29
	100	0	6.01%	-		100	0	9.30%	-
RC1b	25	8	0.00%	6.27	RC2b	25	6	0.00%	80.47
	50	2	6.86%	78.23		50	4	0.00%	95.47
	100	0	12.37%	-		100	0	9.90%	-
RC1c	25	8	0.00%	0.87	RC2c	25	7	0.00%	356.01
	50	3	6.66%	55.15		50	5	0.00%	339.04
	100	0	10.25%	-		100	0	9.40%	-
RC1	25	17	3.23%	10.43	RC2	25	16	5.36%	217.26
	50	5	8.27%	64.38		50	12	3.38%	343.91
	100	0	9.54%	-		100	0	9.44%	-

Table 4.12: Results of the exact algorithm on instances in Dataset 1.

class	size	solved	gap	time	class	size	solved	gap	time
R1a	100	0	4.90%	-	R2a	100	0	-	-
	50	4	1.92%	351.58		50	0	11.09%	-
	25	12	0.00%	11.71		25	3	31.85%	145.67
R1b	100	1	9.19%	52.55	R2b	100	0	-	-
	50	5	3.06%	455.72		50	0	20.55%	-
	25	10	0.04%	2.87		25	4	11.90%	207.95
R1c	100	1	7.06%	127.23	R2c	100	0	9.67%	-
	50	7	4.68%	599.23		50	1	18.91%	139.44
	25	11	0.00%	1.06		25	6	12.13%	170.72
R1	100	2	6.84%	89.89	R2	100	0	9.67%	-
	50	16	2.97%	492.47		50	1	18.42%	139.44
	25	33	0.02%	5.48		8	13	14.51%	176.39
C1a	100	0	3.45%	-	C2a	100	0	24.37%	-
	50	0	0.92%	-		50	2	15.38%	180.24
	25	8	0.29%	445.40		25	6	0.00%	100.98
C1b	100	0	5.21%	-	C2b	100	0	22.03%	-
	50	0	3.32%	-		50	4	30.71%	818.75
	25	7	2.14%	245.84		25	7	0.00%	134.60
C1c	100	2	1.78%	1948.98	C2c	100	0	14.86%	-
	50	0	4.13%	-		50	3	11.62%	1095.53
	25	7	0.31%	305.59		25	7	0.00%	59.73
C1	100	2	3.69%	1948.98	C2	100	0	19.93%	-
	50	0	2.79%	-		50	9	16.06%	769.12
	25	22	1.04%	337.42		25	20	0.00%	98.31
RC1a	100	0	7.03%	-	RC2a	100	0	9.88%	-
	50	0	11.16%	-		50	1	1.59%	114.18
	25	1	4.50%	998.31		25	4	7.35%	164.88
RC1b	100	0	11.35%	-	RC2b	100	0	9.85%	-
	50	2	9.65%	189.42		50	4	0.00%	592.67
	25	8	0.00%	351.47		25	5	0.78%	39.45
RC1c	100	0	9.86%	-	RC2c	100	0	3.49%	-
	50	4	10.57%	285.75		50	5	0.00%	619.54
	25	8	0.00%	34.08		25	7	0.00%	314.71
RC1	100	0	9.53%	-	RC2	100	0	8.28%	-
	50	6	10.53%	253.64		50	10	1.59%	558.26
	25	17	4.50%	240.16		25	16	5.71%	191.24

Table 4.13: Results of the exact algorithm on instances in Dataset 2.

instance	customers	depots	duration	capacity	UB	LB	gap	time
pr01	48	4	500	200	1074.21	1074.21	0.00%	2.21
pr07	72	6	500	200	1418.22	1418.22	0.00%	834.94
pr02	96	4	480	195	1851.99	1743.53	5.86%	3600.02
pr08	144	6	475	190	-	-	-	-

Table 4.14: Results of the exact algorithm on instances in Dataset 3.

Set	LS		BF		DAMPV		iRMP(10+5min)		iRMP(1h+1h)		Fix&Gen(1min)		Fix&Gen(3min)		Fix&Gen(1h+3min)	
	value	time	Δ	time	Δ	time	Δ	time	Δ	time	Δ	time	Δ	time	Δ	time
R1a	4398		-4.27%	531	-5.14%	921.95	-4.64%	106.04	-4.64%	134.87	-6.35%	139.89	-6.33%	161.75	-6.33%	112.64
R1b	2054		-5.67%	586	-6.23%	921.57	-6.83%	264.49	-6.83%	424.63	-7.75%	179.69	-7.48%	425.02	-7.27%	859.00
R1c	1700		-3.49%	605	-5.65%	921.53	-6.64%	244.65	-6.64%	318.56	-6.95%	190.41	-6.53%	333.61	-6.60%	502.51
R1	2717	531	-4.48%	574	-5.51%	921.68	-6.04%	205.06	-6.04%	292.68	-6.81%	170.00	-6.64%	306.79	-6.61%	491.38
C1a	8007		-7.31%	644	-11.12%	921.67	-9.25%	32.47	-9.25%	25.96	-11.22%	18.99	-11.22%	18.53	-11.22%	20.03
C1b	2485		-3.63%	629	-4.00%	921.27	-2.48%	98.39	-2.48%	314.08	-2.51%	44.06	-2.66%	52.05	-2.57%	394.95
C1c	1705		-2.00%	665	-4.88%	921.28	-2.87%	105.67	-2.87%	629.88	-1.20%	112.50	-1.26%	180.39	-1.23%	632.06
C1	4065.67	435	-4.32%	646	-8.70%	921.40	-4.87%	78.84	-4.87%	323.30	-7.86%	58.52	-7.90%	83.65	-7.88%	349.01
RC1a	5184		-1.03%	619	-1.61%	921.36	-3.25%	370.77	-3.25%	904.39	-1.94%	218.93	-2.58%	519.56	-2.64%	841.51
RC1b	2235		-2.00%	591	-3.04%	920.86	-1.88%	272.02	-1.88%	454.32	0.57%	167.07	0.42%	272.36	0.31%	378.42
RC1c	1849		-3.18%	621	-3.66%	921.09	-2.50%	183.93	-2.50%	194.78	-0.71%	129.49	-0.11%	199.52	-0.03%	251.25
RC1	3089.33	470	-2.07%	610	-2.35%	921.10	-2.54%	275.57	-2.54%	517.83	-1.08%	171.83	-1.34%	330.48	-1.39%	490.39
R2a	3809		-2.63%	672	-5.29%	921.24	45.64%	859.36	32.35%	4659.52	6.85%	300.52	0.44%	836.42	0.06%	4318.98
R2b	1797		-3.89%	806	-4.78%	896.98	-7.69%	901.42	-16.40%	5618.96	-11.82%	306.99	-17.44%	919.37	-22.39%	4371.23
R2c	1513		-4.87%	794	-3.96%	921.03	-16.27%	924.39	-18.21%	5700.76	-18.51%	313.73	-22.89%	854.65	-26.46%	4307.33
R2	2373	529	-3.80%	757	-4.87%	913.08	7.23%	895.06	-0.75%	5326.41	-2.10%	307.08	-8.07%	870.15	-9.93%	4332.51
C2a	6717		-5.39%	672	-10.69%	921.30	34.21%	766.75	12.40%	5118.5	-4.57%	225.50	-7.21%	582.97	-14.48%	3324.05
C2b	1970		-3.26%	686	-5.57%	921.15	0.35%	730.78	-9.44%	4035.4	-4.82%	175.26	-10.31%	584.03	-11.46%	2680.96
C2c	1288		-0.91%	633	-1.82%	683.24	-2.50%	557.16	-4.80%	2913.88	2.24%	176.52	-1.66%	434.03	-1.87%	2945.29
C2	3325	444	-3.19%	664	-8.43%	841.90	10.69%	684.90	-0.61%	4022.59	-3.69%	192.43	-7.05%	533.67	-12.09%	2983.43
RC2a	5273		-8.76%	834	-12.42%	921.18	-18.64%	819.13	-18.99%	3566.34	-21.60%	601.90	-22.16%	1478.85	-22.40%	4703.58
RC2b	2324		-5.78%	870	-6.64%	921.14	-22.89%	778.20	-23.72%	3582.67	-28.14%	372.07	-28.24%	1118.27	-28.84%	4199.70
RC2c	1978		-4.22%	910	-14.09%	920.90	-26.05%	768.77	-28.01%	3642.37	-32.23%	362.44	-37.35%	1052.91	-39.03%	3992.84
RC2	3191.67	533	-6.25%	871	-11.29%	921.07	-22.52%	788.70	-23.57%	3597.13	-25.23%	445.47	-26.51%	1216.67	-27.08%	4298.71

(a) Dataset 1

file	best known	iRMP(10+5min)		iRMP(1h+1h)		Fix&Gen(1min)		Fix&Gen(3min)		Fix&Gen(1h+3min)	
		Δ	time	Δ	time	Δ	time	Δ	time	Δ	time
pr01	1074.12	0.00	2.22	0.00	2.22	0.00	2.70	0.00	2.75	0.00	2.75
pr02	1762.21	0.03	437.82	0.03	437.82	0.02	215.27	0.01	445.19	0.01	696.14
pr07	1418.22	0.00	23.88	0.00	23.88	0.02	40.27	0.02	37.95	0.02	33.63
pr08	2096.73	0.04	640.82	0.03	3600.20	0.13	589.50	0.04	1106.45	0.06	4504.50

(b) Dataset 3

Table 4.15: Evaluation of heuristic algorithms for the MDHVRP.

Chapter 5

Branch-and-price for the multi-depot heterogeneous-fleet pickup and delivery problem with soft time windows

5.1 Introduction

Transportation and distribution are major cost factors in the supply chain. At the same time their complexity is growing due to decentralized production and distribution schemes. Consequently it is important to develop effective computational tools for logistics management.

A classical transportation problem is the capacitated vehicle routing problem (CVRP). It deals with the delivery of goods by a fleet of trucks from a central depot to many customer locations. The main goal is to find a set of minimum cost routes for the vehicle fleet in order to meet customers demands and without violating capacity constraints.

The pickup and delivery problem (PDP) is a generalization of the CVRP in which each customer demands to pickup a certain amount of goods from an origin location and to deliver it to a destination location. Each route has to satisfy pairing constraints (origin and destination must be visited by the

same vehicle, without any transshipment) and precedence constraints (the origin location must precede the destination in the route).

Real world applications often require to take into account several other details. In order to be competitive and benefit from economy scale, small and medium companies tend to aggregate. As a consequence there is the need to manage several depots and a fleet of highly heterogeneous vehicles, with different capacities and operating costs; it is also common that customers impose constraints on the arrival and departure times of the vehicles visiting them for pickup or delivery operations [12]. It is also possible that the customers express preferences on the service time which are not constraints, but they should be taken into account to gain competitiveness on the market if a customer is visited out of his preferred time window, a penalty is incurred. The resulting problem is called multi-depot heterogeneous-fleet pickup and delivery problem with soft time windows (MDHPDPSTW) and has application in various scenarios, such as urban courier services, less-than-truckload transportation, door to door transportation services. The problem has been modeled according to the requirements of a real project whose aim is to develop an integrated intelligent system for transportation companies operating in the urban area of Milan (Italy).

The problem has been widely studied in the version with hard time windows constraints (PDPTW). Several exact approaches based on branch-and-cut [15, 75] and branch-and-price [40, 81, 74, 3] have been proposed. For comprehensive reviews on routing problems involving pickup and delivery, the reader is referred to the works of Savelsbergh and Sol [80], Cordeau et al. [18] and Parragh et al. [64].

The soft time windows case is much less studied. It was addressed in an early work by Sexton and Choi [82]. They developed a heuristic algorithm based on Benders' decomposition. Several heuristic approaches have been proposed for the vehicle routing problem with soft time windows (VRPSTW): Koskosidis et al. [53] developed an optimization-based heuristic; Balakrishnan [2] proposed several constructive heuristics, while Taillard et al. [84] and Chiang and Russell [13] presented tabu search heuristics; Ibaraki et al. [48] studied

acceleration techniques for local search algorithms in the case of multiple soft time windows.

Recently, Liberatore et al. [57] and Qureshi et al. [69] proposed branch-and-price approaches for the vehicle routing problem with soft and semi-soft time windows respectively.

In this work we propose a branch-and-price algorithm for the MDHPDPSTW where the pricing problem is solved by a modified version of the algorithm developed by Liberatore et al. [57] owing to the pickup and delivery constraints.

5.2 Problem formulation

Given a set $\mathcal{K}$ of vehicle types, a set $\mathcal{H}$ of depots and a set $\mathcal{N}$ of customer requests, the problem can be defined on a directed graph $G = (V, A)$, where V contains a vertex for each depot $h \in \mathcal{H}$ and for the pickup and delivery locations of each customer $i \in \mathcal{N}$. Let $\mathcal{P}$ and $\mathcal{D}$ be the sets of vertices corresponding to all pickup and delivery locations respectively. We denote with $\mathcal{P}(i)$ and $\mathcal{D}(i)$ the pickup and delivery vertices of customer $i \in \mathcal{N}$ and with $\mathcal{N}(j)$ the customer to whom vertex $j \in \mathcal{P} \cup \mathcal{D}$ belongs to. Non negative weights d_{ij} and t_{ij} are associated with each arc $(i, j) \in A$: they represent the transportation cost and the traveling time respectively. In this work we assume they are independent from the vehicle type. Since we are focusing on the last mile transportation, the performances of the different vehicles are quite similar. With each vertex $j \in V$ are associated a load variation q_j (positive for pickup and negative for delivery operations), a service time s_j, a hard time window $[A_j, B_j]$ and a soft time window $[a_j,b_j]$; if the service at location j starts inside its time window no penalty is incurred, otherwise a linear penalty, proportional to the anticipation or delay through non-negative coefficients α_j and β_j respectively, has to be paid. Let T_j be the starting time

of service at location j, the penalty term $\pi(T_j)$ is defined as follows:

$$\pi(T_j) = \begin{cases} \alpha_j(a_j - T_j) & \text{if } T_j < a_j \\ 0 & \text{if } a_j \leq T_j \leq b_j \\ \beta_j(T_j - b_j) & \text{if } T_j > b_j. \end{cases}$$

Each vehicle type $k \in \mathcal{K}$ has given capacity w_k and fixed cost f_k. A limited number of vehicles is available: at most u_{hk} vehicles of type $k \in \mathcal{K}$ can be based at depot $h \in \mathcal{H}$. This model the scenario where different companies (each of them with its own fleet and depot) share their resources.

The objective is to minimize the sum of vehicles fixed costs and routing costs (including penalties), satisfying the following conditions:

I all customers are served,

II each customer is visited by only one vehicle,

III each route begins at a depot and ends at the same depot,

IV the capacity of the associated vehicle is not exceeded,

V pickup and delivery operations for each customer are performed in the same route (pairing constraints),

VI the pickup vertices are visited before the corresponding delivery vertices (precedence constraints),

VII the number of available vehicles of each type for each depot is not exceeded.

We consider a set covering formulation of the MDHPDPSTW. We say that a route is feasible if it satisfies conditions II, III, IV, V and VI. Let Ω_{hk} be the set of all feasible routes using a vehicle of type $k \in \mathcal{K}$ from depot $h \in \mathcal{H}$ and let $\Omega = \cup_{h\in\mathcal{H},k\in\mathcal{K}}\Omega_{hk}$ be their union. We associate a binary variable x_r with each feasible route $r \in \Omega_{hk}$, which takes value 1 if and only if route r is selected. Let e_{ir} be a binary coefficient with value 1 if and only if customer

$i \in \mathcal{N}$ is visited by route r. Let c_r be the overall cost of route r, taking into account vehicle fixed cost f_k, routing costs and penalties for soft time windows violations; With these definitions we obtain the following integer linear programming model:

$$\min \sum_{h\in\mathcal{H}} \sum_{k\in\mathcal{K}} \sum_{r\in\Omega_{hk}} c_r z_r \tag{5.1}$$

$$\text{s.t.} \sum_{h\in\mathcal{H}} \sum_{k\in\mathcal{K}} \sum_{r\in\Omega_{hk}} e_{ir} z_r \geq 1 \qquad \forall i \in \mathcal{N} \tag{5.2}$$

$$\sum_{r\in\Omega_{hk}} z_r \leq u_{hk} \qquad \forall h \in \mathcal{H},\ k \in \mathcal{K} \tag{5.3}$$

$$z_r \in \{0,1\} \qquad \forall h \in \mathcal{H},\ k \in \mathcal{K},\ r \in \Omega_{hk} \tag{5.4}$$

Constraints (5.2) are standard set covering constraints, modeling condition I, while (5.3) impose limits on the maximum number of available vehicles of each type at each depot, modeling condition VII. The objective is to minimize the overall cost of the selected routes. In the remainder we indicate this formulation as Master Problem (MP).

5.3 Branch-and-price

We solve the linear relaxation of the MP to obtain a lower bound which is used in a tree search algorithm. The number of variables is exponential in the cardinality of the customer set $\mathcal{N}$, thus we use a column generation approach.

5.3.1 Column Generation

We initially consider only a small subset of the variables in the MP. Such initial Restricted Master Problem (RMP) includes

- a set of $|\mathcal{N}|$ columns, one for each customer, representing the optimal paths serving one customer at a time,

- a set of columns representing a feasible MDHPDPSTW solution, found using a straightforward greedy policy, described in Section 5.3.3,
- an additional *dummy column* $\bar{r}$ of very high cost, having $e_{i\bar{r}} = 1 \forall i \in \mathcal{N}$ and every other coefficient set to 0.

The aim of the dummy column is to ensure feasibility at each node of the search tree.

We solve the linear relaxation of the RMP, and we search for columns which are not in the RMP, but have negative reduced cost. If no such column exists, the solution is optimal for the MP linear relaxation as well, and thus yields a valid lower bound to the problem. On the opposite, if any negative reduced cost column is found, it is added to the RMP, and the process is iterated.

Let λ and μ be the non negative dual vectors corresponding respectively to constraints (5.2), and to constraints (5.3) rewritten as $\geq$ inequalities. The reduced cost of a column encoding route r has the following form:

$$c_r - \sum_{i \in \mathcal{N}} \lambda_i a_{ir} + \mu_{hk}.$$

The routes generated must comply with conditions II, III, IV, V and VI: it is important to note that the same sequence of customers may correspond to a feasible or to an infeasible route according to the type of vehicle and the depot it is associated with. For instance, different types of vehicles imply different capacities. Moreover, cost c_r and dual variables μ_{hk} depend on both k and h. Therefore, at each column generation iteration we have to solve $|\mathcal{K}| \cdot |\mathcal{H}|$ pricing problems. We also remark that it is never convenient to perform cycles in a feasible solution, although the prize structure given by dual variables can make it appealing; therefore, a search must be performed for elementary routes only. Hence, given a particular depot h and vehicle type k, the problem of finding the most negative reduced cost column encoding a route for vehicle k using depot h turns out to be a Resource Constrained Elementary Shortest Path Problem (RCESPP); in particular we have capacity, pickup and delivery, and soft time windows constraints.

The RCESPP is NP-hard [37]; we solve it using a bi-directional dynamic programming method, which is based on the algorithm proposed by Liberatore et al. [57] for the VRPSTW, modified to deal with pickup and delivery constraints.

The main difficulty when dealing with soft time windows in column generation algorithms is that the possibility of trading time versus cost generates an infinite number of feasible solutions of the pricing problem that do not dominate one another. In algorithmic terms this means that the pricing algorithm must take into account an infinite number of Pareto-optimal states. In the algorithm by Liberatore et al. [57] different states are grouped into *labels* by means of a cost function that gives the cost of the corresponding path for each feasible service starting time at the current vertex.

We now describe the modified version of the dynamic programming procedure we use in our algorithm.

5.3.1.1 Exact Dynamic Programming

Let us consider first the case of a single depot and a single vehicle type: let s and v be the two distinct copies of the depot, representing the departure and arrival node and let w be the capacity of the vehicle.

The solving technique consists of a bi-directional extension of node labels. We propagate forward labels describing partial paths from the starting depot vertex s and backward labels corresponding to partial paths ending in t. Then pairs of labels are joined to obtain complete routes.

Modified costs. To compute the cost of the routes in the pricing problem, we consider modified traveling costs $\bar{d}_{ij}$ for the arcs of the graph that takes into account the dual values of the constraints of the master problem. The original cost matrix (d_{ij}) satisfies the triangular inequalities, but $(\bar{d}_{ij})$ usually does not.

As shown by Ropke and Cordeau [74], it is computationally convenient to work with a cost matrix that satisfies the *delivery triangle inequalities*, i.e.

$\bar{d}_{ij} + \bar{d}_{jk} \geq \bar{d}_{ik}$ for all $j \in \mathcal{D}$. since it allows to apply weaker dominance rules. We will see further details on this topic later, when discussing the dominance rules. The authors proposed a method to transform an arbitrary cost matrix into a cost matrix that satisfies the delivery triangle inequality while maintaining the optimal solution of the pricing problem.

Since we use a bi-directional dynamic programming method, for backward search we would like to have the same property on the pickup vertices: $\bar{d}_{ij} + \bar{d}_{jk} \geq \bar{d}_{ik}\ \forall j \in \mathcal{P}$. Thus we follow a different strategy: we consider the following linear program

$$
\begin{aligned}
&\min\ v \\
&\text{s.t. } \bar{d}_{ij} = (d_{ij} - \sigma_{ij}) - (\alpha_i^{out} + \alpha_j^{in}) && \forall i, j \in \mathcal{N} \\
&\quad \bar{d}_{ij} + \bar{d}_{jk} \geq \bar{d}_{ik} && \forall i, j, k \in \mathcal{N} \\
&\quad \rho_l = \lambda_l - (\alpha^{in}_{\mathcal{P}(l)} + \alpha^{out}_{\mathcal{P}(l)} + \alpha^{in}_{\mathcal{D}(l)} + \alpha^{out}_{\mathcal{D}(l)}) && \forall l \in \mathcal{R} \\
&\quad v \geq \rho_l && \forall l \in \mathcal{R} \\
&\quad \rho_l \geq 0 && \forall l \in \mathcal{R}
\end{aligned}
$$

where σ_{ij} are cost coefficients representing the contribution to the modified arc costs due to the values of the dual variables of eventual branching constraints or valid inequalities defined on the arc variables. We obtain a modified cost matrix $(\bar{d}_{ij})$ that satisfies the triangular inequalities and a set of prizes ρ_l associated to the customers. In the dynamic programming procedure, we assign to a path the prize of a customer when the pickup vertex is reached, during forward propagation, when the delivery vertex is reached, during backward propagation. This way we ensure delivery triangle inequalities in the former case and pickup triangle inequalities in the latter. Moreover we are sure that the cost of any s–t path does not change if computed with $(\bar{d}_{ij})$ and ρ instead of (d_{ij}) and λ.

States and labels. In bi-directional dynamic programming we associate *forward states* and *backward states* to the vertices of the graph. A forward state associated with vertex $i \in V$ represents a path from the depot s to

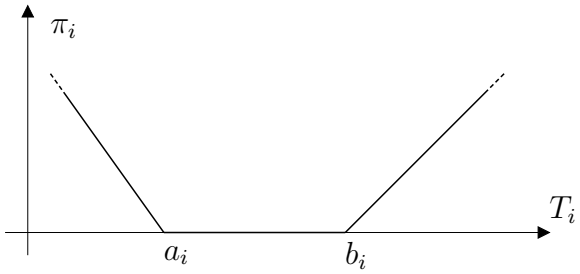

Figure 5.1: A soft time window at a generic vertex $i \in V$: a linear penalty π_i is incurred depending on the service starting time T_i.

i. Different states associated with the same vertex correspond to different feasible paths reaching that vertex. When a vehicle reaches a vertex it can start the service immediately or it can wait and start the service at a later time in order to reduce costs in case of early arrival. Therefore from each feasible state an infinite number of feasible states can be generated. For this reason the dynamic programming algorithm must take into account an infinite number of non-dominated states and this is done by grouping them into labels. Each label corresponds to an infinite number of states associated with the same path. A label associated with vertex $i \in V$ is a tuple $L_i = (S, \chi, \Psi, C(T_i), i)$, where i is the last reached vertex, S is the set of customers visited along the path, χ is the load of the vehicle after visiting vertex i, Ψ is the set of "open" customers (the ones for which the pickup operation has been performed but not yet the delivery), C is the cost of the path, T_i is the time at which the service at vertex i begins. In each label the continuous function $C(T_i)$ describes the trade-off between cost and time. This function is piecewise linear and convex , because it is the sum of piecewise linear and convex functions, like the one shown in Figure 5.1, one for each visited vertex.

In the same way we define *backward states*, corresponding to paths from vertex i to the final depot t represented by labels $(S, \chi, \Psi, C(T_i), i)$, where T_i is the time at which the service at vertex i ends. Of course here Ψ contains the customers for which the delivery vertex is in the path, but the pickup is not.

Extension. The dynamic programming algorithm iteratively extends all feasible forward and backward labels to generate new ones. The extension of a forward label corresponds to appending an additional arc (i, j) to a path from s to i, obtaining a path from s to j, while the extension of a backward label corresponds to appending an additional arc (j, i) to a path from i to t, obtaining a path from j to t.

In forward extensions the sets S and Ψ are initialized at $\emptyset$, the load χ is set to 0 and the cost function is $C(T_s) = 0 \, \forall T_s \geq 0$ at the starting depot vertex s.

The search is restricted to elementary paths by discarding extensions to any vertex j corresponding to the pickup of a customer already in S. To satisfy precedence constraints we also discard extension to any vertex j corresponding to the delivery location of a customer not in Ψ. Further, we cannot exceed the capacity of the vehicle, thus we must have

$$\chi + q_j \leq w.$$

When a label $(S, \chi, \Psi, C(T_i), i)$ is extended forward to a vertex j, a new label $(S', \chi', \Psi', C(T_j)', j)$ is computed using the following rules.
If j is a pickup vertex

$$S' = S \cup \{\mathcal{N}(j)\} \tag{5.5}$$
$$\chi' = \chi + q_j \tag{5.6}$$
$$\Psi' = \Psi \cup \{\mathcal{N}(j)\} \tag{5.7}$$

while if j is a delivery vertex

$$S' = S \tag{5.8}$$
$$\chi' = \chi + q_j \tag{5.9}$$
$$\Psi' = \Psi \setminus \{\mathcal{N}(j)\} \tag{5.10}$$

Finally the cost $C(T_i)$ at each extension depends on both traveling time and

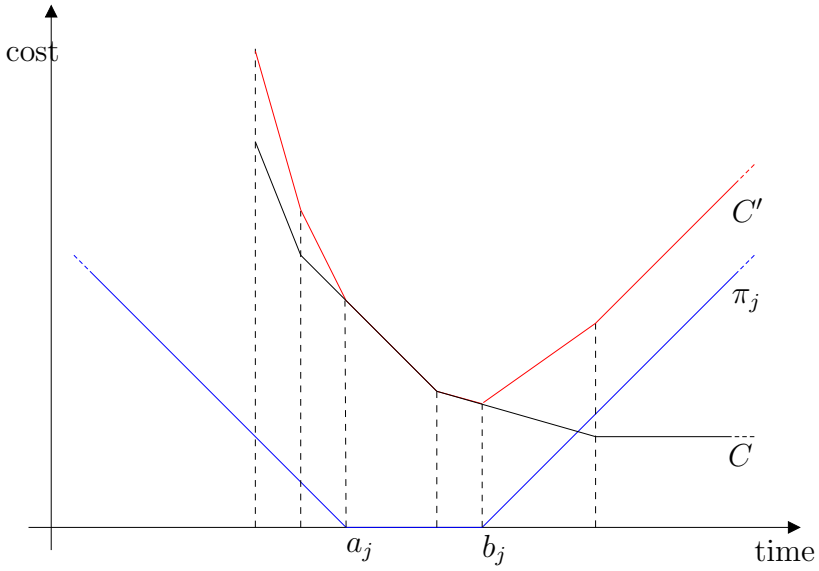

Figure 5.2: Forward extension of a label to vertex j. The C' function resulting from the extension is the sum of the C function of the source label suitably shifted (black line) and the penalty function π_j.

penalties and it is updated according to the formula:

$$C(T_j) = C(T_j - (s_i + t_{ij})) + \bar{d}_{ij} + \pi_j(T_j).$$

In this expression the cost function of the predecessor is evaluated at $T_i = T_j - (s_i + t_{ij})$, which is the latest time instant at which the service at vertex i should begin to allow starting the service at vertex j at time T_j. Figure 5.2 shows an example of forward extension. In graphical terms, the cost function $C(T_i)$ is shifted (black line) to the right by the service time at vertex i, that is s_i, plus the traveling time t_{ij} spent to reach vertex j, and it is shifted vertically by the traveling cost $\bar{d}_{ij}$. Then it is summed to the penalty term π_j, which depends on the arrival time T_j. If $C(T_i)$ is continuous and convex, then the resulting function $C(T_j)$ preserves both these properties. The number of segments in these piecewise linear functions is increased by at most two at every extension.

The extension rules for backward propagation are similar, but the role played by pickup and delivery vertices is switched. When a label $(S, \chi, \Psi, C(T_i), i)$

is extended backward to a vertex j, a new label $(S', \chi', \Psi', C(T_j)', j)$ is computed using the following rules.
If j is a pickup vertex

$$S' = S \tag{5.11}$$
$$\chi' = \chi + q_j \tag{5.12}$$
$$\Psi' = \Psi \setminus \{\mathcal{N}(j)\} \tag{5.13}$$

while if j is a delivery vertex

$$S' = S \cup \{\mathcal{N}(j)\} \tag{5.14}$$
$$\chi' = \chi + q_j \tag{5.15}$$
$$\Psi' = \Psi \cup \{\mathcal{N}(j)\} \tag{5.16}$$

The cost $C(T_j)$ is computed according to the formula

$$C(T_j) = C(T_j + (s_i + t_{ji})) + \bar{d}_{ji} + \hat{\pi}_j(T_j).$$

The cost function of the successor is evaluated at $T_i = T_j + s_i + t_{ji}$ which is the earliest possible time at which the service at vertex i should end in order to end at time T_j the service at vertex j. Since in backward propagation times refer to the end of the service, we have to consider also shifted penalty functions $\hat{\pi}(T_i) = \pi(T_i - s_i)$.

The algorithm iteratively extends each forward and backward labels to all possible successors or predecessors respectively. In order to limit the extension of forward and backward labels and to reduce useless duplication of paths, we impose that each partial path can use at most half of a critical resource whose consumption is monotone along the paths. The most useful critical resource is the tightest one. In our case the only meaningful choice is time.

Cycles of length 2 are automatically forbidden by imposing pickup and delivery constraints, i.e. forward (backward) labels are not extended to pickup (delivery) vertices of customers in Ψ; further, a path containing a cycle must

satisfy a sequence of at least 5 vertices (the pickup and delivery locations of the customer visited twice and a vertex associated with another customer). If the hard time windows are small compared with the travel times, such a cycle is not common and it may be convenient to solve the sub-problem without imposing the elementarity on the paths [30]. In this work we want to model a last-mile delivery application where travel time are usally small; thus, we decided to keep the elementarity constraints following [74] and [3].

Dominance rules. The ability to reduce the number of states generated plays a crucial role in the effectiveness of dynamic programming algorithms. We first remove from each label part of the cost function encoding feasible states that cannot lead to an optimal solution and then we apply suitable dominance rules to fathom labels. Let us consider a generic forward label, like the one in Figure 5.3. Since it is possible to wait at no cost, all the states corresponding to value of service starting time for which the first derivative of the cost function is positive are dominated by states of the same label with smaller cost and time. In graphical terms the rightmost part of the piecewise linear function is always an unbounded horizontal half line, that replace all the segments with positive first derivative. Further, any service starting time T_i that falls outside the hard time window of the last visited customer is infeasible. Hence, the domain of the cost function is restricted to the interval $[A_i, B_i]$. Symmetric arguments can be applied to backward labels: an unbounded horizontal half line replace the leftmost segments, with negative slope, and the domain of the function is bounded by the hard time window of the current vertex.

The second strategy to reduce the number of explored states consists in a test, based on a set of dominance rules, that allows to identify states that cannot lead to an optimal solution of the RCESPP. Those states can of course be discarded without losing the optimality guarantee.

Let $l' = (S', \chi', \Psi', C'(T_i'), i)$ and $l'' = (S'', \chi'', \Psi'', C''(T_i''), i)$ be two forward or backward states associated with node i. Then the former dominates the

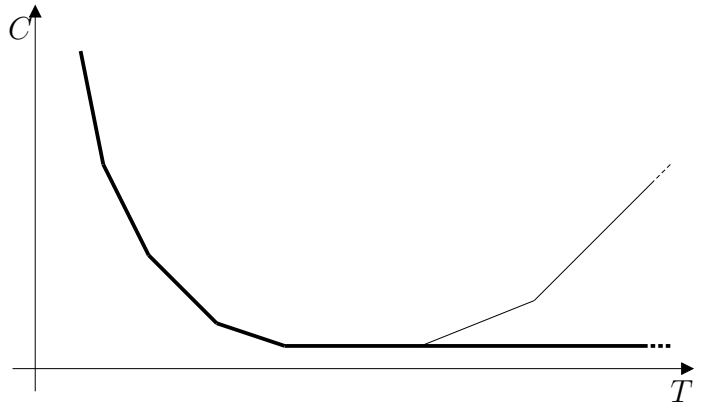

Figure 5.3: A generic forward label cost function. The rightmost part of the function, with positive first derivative, is replaced by an horizontal segment

latter if the following conditions are satisfied

$$
\begin{aligned}
&(a) \quad S' \subseteq S'' \\
&(b) \quad \chi' \leq \chi'' \\
&(c) \quad \Psi' \subseteq \Psi'' \\
&(d) \quad T'_i \leq T''_i \\
&(e) \quad C'(T'_i) \leq C''(T''_i).
\end{aligned}
$$

Since for any label the load of the vehicle is uniquely determined by the open customer set, condition (b) is implied by condition (c) and it is, therefore, redundant. Further, as shown in Feillet et al. [42], it is sometimes possible to identify some vertices $u \in \mathcal{N}$ that cannot be reached by any feasible extension of a given label, because of resource limitations. In this case it is useful to insert u in the set S of that label: it is easy to check that enlarging set S'' helps satisfying condition (a); at the same time, if a node cannot be reached by extending label l' due to resource limitations, it cannot be reached by extending label l'' either, since resource consumption in l'' is not lower. Therefore, enlarging each set S allows dynamic programming fathoming more labels and hence helps to reduce the computational time.

As shown in [74], the dominance condition (c) is valid for forward labels only if the delivery triangle inequalities hold. In presence of this property we are

sure that visiting a delivery node never makes the path cheaper. Without the delivery triangle inequalities, condition (c) would be $\Psi' = \Psi''$, which is more restrictive and therefore allows to fathom a smaller number of states. The same consideration is true for backward labels if we substitute the delivery triangle inequalities with the pickup triangle inequalities. Earlier in this section we have described how it is possible to generate suitable modified cost sets for both forward and backward propagation.

When dominance is applied to single states and the tests succeed, the result is to remove the dominated one. In our case we have labels each representing an infinite number of states. The effect of the dominance test is to delete some parts of the cost functions (see Figure 5.4). The states surviving the test are those of minimum cost for each feasible starting service time. As a consequence the resulting cost function may contain gaps and it is not convex in general.

When a new label is generated, it is tested against all the other labels referred to the same vertex. If $S' = S''$ and $\Psi' = \Psi''$ the two labels are merged into one. This way at most one piecewise linear function for each combination of S, Ψ and i. In this case the new function may contain vertical gaps and concavities, like the one obtained considering the heavy lines (both black and gray) in Figure 5.4. Instead, if $S' \subset S''$ or $\Psi' \subset \Psi''$ only the states in label l'' can be dominated while l' is left unchanged. The new cost function C'' may contain horizontal gaps (black heavy lines in Figure 5.4):

Join. Forward and backward paths must be joined together to produce complete $s - v$ paths. The result of the join is a feasible solution if all the resource constraints are satisfied. When a forward path $(S^{fw}, \chi^{fw}, \tau^{fw}, \delta^{fw}, \rho^{fw}, C^{fw}, i)$ is joined with a backward path $(S^{bw}, \chi^{bw}, \tau^{bw}, \delta^{bw}, \rho^{bw}, C^{bw}, j)$, the feasibility conditions are:

$$\Psi^{fw} = \Psi^{bw}$$
$$S^{fw} \bigcap S^{bw} = \Psi^{fw}$$
$$T_j^{bw} - T_i^{fw} \geq s_i + t_{ij} + s_j$$

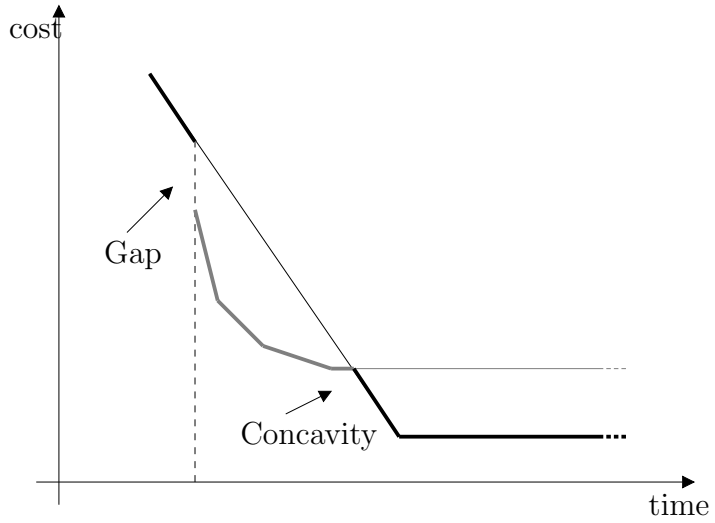

Figure 5.4: As an effect of the dominance test, some parts of the piecewise linear functions are deleted. Only the states drawn with heavy lines are non-dominated and survive the test.

If the join is feasible, then the cost of the resulting path can be obtained as the minimum of the function

$$C(T) = C^{fw}(T - s_i - \frac{t_{ij}}{2}) - \bar{d}_{ij} + \sum_{l \in \Psi^{fw}} \rho_l + C^{bw}(T + s_i + \frac{t_{ij}}{2}).$$

The term $\sum_{l \in \Psi^{fw}} \rho_l$ is needed because the prizes of the customers in $\Psi^{fw} = \Psi^{bw}$ are counted twice, once in the forward and once in the backward label. This function may have several local minima, as shown in Figure 5.5. However the detection of the global minimum takes time linear in the number of discontinuity points of the two piecewise linear functions, since it requires a merge operation between two sorted lists.

Decremental state space relaxation. The dynamic programming algorithm is executed iteratively applying decremental state space relaxation [72]. The idea is to project the state space of the problem to a smaller one, by removing some elementarity constraints. From an algorithmic point of view, this amounts to identify a set of *critical customers* $\tilde{\mathcal{R}}$, and to replace

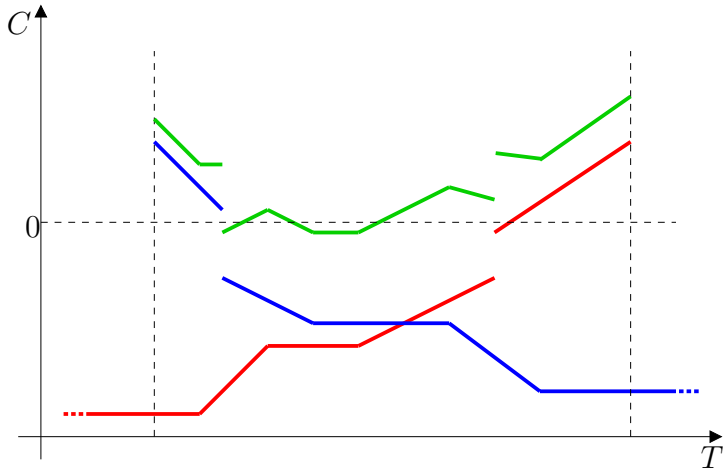

Figure 5.5: When a forward and a backward labels are joined together, the two corresponding cost functions (red and blue) are summed up. The resulting piecewise linear function (green) may have multiple local minima. All its points below 0 correspond to negative reduced cost routes.

extension rule (5.5) as

$$S' = (S \cup \{\mathcal{N}(j)\}) \cap \tilde{\mathcal{R}}$$

This relaxed problem can be solved more efficiently, since more labels can be compared in the dominance test.

In order to identify a good critical node set, we initialize $\tilde{\mathcal{R}} = \emptyset$; then we iteratively solve the state space relaxation of the pricing problem and insert in the set $\tilde{\mathcal{R}}$ all the nodes visited more than once in the optimal path, until we find an elementary one.

Aggregated pricing. In principle the dynamic programming procedure has to be repeated for all depots and vehicle types. We observe that any valid state for a certain vehicle type $k \in \mathcal{K}$ is also feasible for larger vehicles. Let's assume that the vehicle types are sorted by non decreasing capacity. We call the dynamic programming on vehicle type 1. Whenever a label would have been discarded because $\chi > w_1$, we save it in a list L, instead of deleting it. To solve the RCESPP for the second vehicle type, we keep all the labels

generated for $k = 1$, and we extend the labels in L that are feasible for the new vehicle type, i.e. those with $\chi \leq w_2$, and so on.

5.3.1.2 Heuristic pricing

As stated before, the pricing subproblem is difficult to be solved. Thus we experiment on the use of two heuristic methods, both based on the dynamic programming described above, in order to generate negative reduced cost columns.

Heuristic Dynamic Programming. The heuristic dynamic programming pricing algorithm (HDP) uses a slightly modified version of the bi-directional label extension described above: in particular, the dominance condition (a) on the set S of visited nodes is replaced with the surrogate condition

$$|S'| \leq |S''|$$

Restricted Graph Dynamic Programming. Following [73], we construct a graph G_{10} in which each vertex $i \in V$ is incident with at most 10 outgoing arcs reaching a pickup vertex and 10 outgoing arcs reaching a delivery vertex. We choose the arcs that are cheapest with respect to d_{ij}. Then we add the arcs connecting the starting depot vertex s with the pickup vertices and the delivery vertices with the ending depot t. We then call the dynamic programming method on G_{10}. The described algorithm will be called Restricted Graph Dynamic Programming (RGDP).

5.3.2 Valid inequalities

The lower bound obtained from the linear relaxation of (5.1) - (5.4) can be strengthened by means of valid inequalities. We consider a special case of the subset-row inequalities introduced by Jepsen et al. [50]. Let $\mathcal{C} = \{C \subseteq \mathcal{N} : |C| = 3\}$ be the set of all request triplets, where a request triple

is any subset of three customers. For $C \in \mathcal{C}$, let $\Omega(C) \subseteq \Omega$ be the subset of routes servicing at least two requests in C. The following inequalities are valid:

$$\sum_{r \in \Omega(C)} z_r \leq 1 \quad \forall C \in \mathcal{C} \tag{5.17}$$

The separation of these inequalities can be done by complete enumeration since their number is polynomial. A new state variable for each active inequality has to be considered in the dynamic programming procedure to take into account the value of the corresponding dual variables.

5.3.3 Primal heuristic

We use a simple greedy heuristic to obtain an initial primal bound. We start with an empty route and a randomly selected vehicle. We select the customer (among the ones not yet covered) which can be inserted into the route with the smallest cost increase and we add it. We recall that each customer requires to visit two vertices. Thus, we evaluate their insertion in all the feasible positions along the route. If no customer can be feasibly inserted, we start a new route. The process is iterated until all customer has been satisfied. If the limit on the number of available vehicles is tight the heuristic may fail in finding a complete solution. Nevertheless the columns identified by the algorithm can be used to initialize the restricted master problem, as mentioned earlier.

5.3.4 Branching strategy

We use two branching policies, described hereafter; they are similar to those described in [78] and widely used in the literature for the VRP.

Branching on the number of vehicles. Let $\bar{x}_r$ be the (possibly fractional) value of each variable x_r in the optimal MP fractional solution and let $y_{hk} = \sum_{r \in \Omega_{hk}} \bar{x}_r$ be the (possibly fractional) number of vehicles of type k

in such solution. We select the vehicle type $\bar{k}$ and the depot $\bar{h}$ whose corresponding $y_{\bar{h}\bar{k}}$ variable has its fractional part closest to 0.5; then we perform binary branching, imposing to use at least $\lceil y_{\bar{h}\bar{k}} \rceil$ vehicles of type $\bar{k}$ in one child node and no more than $\lfloor y_{\bar{h}\bar{k}} \rfloor$ in the other.

These branching decisions are handled as follows: first, we modify constraints (5.3) as follows

$$l_k \leq \sum_{r \in \Omega_{hk}} x_r \leq u_k \quad \forall h \in \mathcal{H}, k \in \mathcal{K} \tag{5.18}$$

then we set $l_{\bar{h}\bar{k}} = \lceil y_{\bar{h}\bar{k}} \rceil$ in one child node, $u_{\bar{h}\bar{k}} = \lfloor y_{\bar{h}\bar{k}} \rfloor$ in the other.

The advantage of this branching technique is to leave the pricing subproblem unchanged: dual variables μ_k still appear as constants in the objective function of the pricing problem, but they are now unrestricted in sign. On the other hand constraints (5.18) belong to the MP, which is solved as a linear programming problem, and tightening them usually results in rather weak improvements in the lower bound.

Branching on arcs. When y_{hk} is integer for all combinations of depots $h \in \mathcal{H}$ and vehicle types $k \in \mathcal{K}$, we apply a different branching rule which forbids the use of some arcs. We choose the customer $i \in \mathcal{R}$ that is split among the largest number of routes in the optimal fractional solution of MP and we forbid half of its outgoing arcs to be used in the first child node, and the other half to be used in the second child node. Unfortunately we cannot handle these branching decision by removing the corresponding arcs from the graph or by setting their travel time to $+\infty$ because we need to maintain the delivery and pickup triangle inequality properties, as explained above. The branching decision is then enforced again by adding a cut to the master problem.

5.4 Experimental analysis

Implementation and hardware. Our algorithms have been implemented in C++, using SCIP 1.2 [1] as a branch-and-cut-and-price framework. Our

version of SCIP embeds the CPLEX 12 implementation of the simplex algorithm to solve LP subproblems, and automatically switches between primal and dual simplex, depending on the characteristics of each instance. SCIP performs advanced management of the column and row pools, including their removal and re-insertion. We turned off preprocessing and automatic cut generation features, but kept all other parameters at their default values, including the decision tree search policy.

The results reported in this section are obtained using a single core of an Intel Core 2 Duo 3GHz workstation, equipped with 2GB of RAM, running Linux OpenSuse 11. The time limit was set to one hour in all tests.

Benchmark instances. To test our algorithm we considered a set of instances (RCL07) proposed by Ropke and Cordeau [74] for the pickup and delivery problem with time windows. The instances were produced with a generator similar to that initially proposed by Savelsbergh and Sol [81]. In all instances, the coordinates of each pickup and delivery location are chosen randomly, according to a uniform distribution over the $[0, 50] \times [0, 50]$ square. The load q_i of request i is selected randomly from the interval $[5, Q]$, where Q is the vehicle capacity. A planning horizon of length $T = 600$ is considered, and each time window has width W. In all instances, the primary objective consists of minimizing the number of vehicles, and a fixed cost of 10^4 is thus imposed on each vehicle type. The instances are grouped in four classes according to the different values of Q and W. The characteristics of these classes are summarized in Table 5.1. There are 10 instances with $30 \leq n \leq 75$ for each class. The name of each instance (e.g., AA50) indicates the class to which it belongs and the number of requests it contains.

Instances RCL07 have hard time windows. In order to test our algorithm, we introduced three soft time windows patterns:

I $a_i = A_i, b_i = B_i,\ \alpha_i = \beta_i = \infty\ \forall i \in \mathcal{N}$;

II $a_i = A_i + \frac{B_i - A_i}{3}, b_i = B_i - \frac{B_i - A_i}{3},\ \alpha_i = \beta_i = 1.5\ \forall i \in \mathcal{N}$;

III $a_i = A_i + \frac{B_i - A_i}{2}, b_i = B_i - \frac{B_i - A_i}{2},\ \alpha_i = \beta_i = 1\ \forall i \in \mathcal{N}$.

Class	Q	W
AA	15	60
BB	20	60
CC	15	120
DD	20	120

Table 5.1: Characteristics of the RCL07 instances

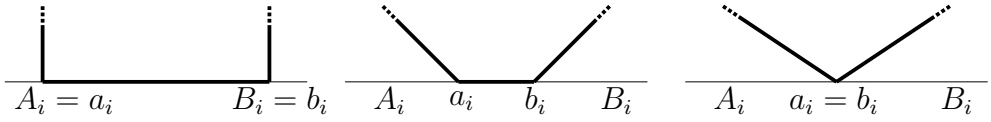

Figure 5.6: Penalty functions corresponding to the different soft time windows types.

The corresponding penalty functions are depicted in Figure 5.6.

Finally, to study the behavior of the proposed method in presence of an heterogeneous fleet, we modified the RCL07 with three vehicle types with capacities 15, 20 and 25. The dataset of instances with heterogeneous fleet and soft time windows of type II is referred to as RCL07_H.

In the remainder we present the outcome of three different sets of experiments. In Subsection 5.4.1 we show the performance of the column generation procedure to compute a lower bound; in Subsection 5.4.2 we present the results of the overall branch-and-price algorithm; in Subsection 5.4.3 we discuss the impact of the soft time windows on the optimal solutions of the pickup and delivery problem.

5.4.1 Lower bound

Table 5.2 shows the computation of the root node lower bound either by using only the exact dynamic programming algorithm or by calling one of the heuristic algorithms described in Section 5.3.1.2 and the exact dynamic programming only when the heuristic fails. In particular, for each instance we show the best known solution (UB), the root node lower bound (LB), and the behavior of the three considered approaches. For each of them we

present the computation time in seconds (time) and the number of iterations (iter) of column generation. In the case of the heuristic approaches, we also show the number of calls (fail) to the exact dynamic programming (e.g. 1 means that the exact dynamic programming was called just to prove that no new column was to be generated). As one can see, the heuristic HDP often finds all the columns, however the computing time is sometimes larger than that of the exact DP. Instead, RGDP is able to reduce the computing time (mostly on the largest instances), even if several calls to the exact DP are necessary.

In Table 5.3, we report the results of the computation of the root node lower bound for the instances RCL07_H. We show the behavior of the exact dynamic programming that considers just one vehicle type at a time (Standard Pricer) with respect to the aggregated pricer (that considers all vehicle types at once). The table shows for each of the two pricers the computing time expressed in second (time), the number of iterations (iter) of column generation and the number of generated columns (cols). A first consideration is that these instances are more difficult than the previous ones, indeed it was not always possible to compute the lower bound within the time limit of one hour. This is due to the fact that different types of vehicles are involved, but it is also a consequence of the larger capacities of the vehicles. The aggregated pricer is not helpful in reducing the computing time. One of the possible explanation for this behavior is the following: when applying the aggregated pricer, the exact DP starts considering the vehicle type with the smallest capacity; when paths exceeding this capacity are found, they are stored and examined for the successive vehicle type with larger capacity and so on. However, some of these paths would not even need to be considered when the standard pricer is directly applied for the corresponding vehicle type, thus leading to a shorter computing time.

Table 5.4 shows the results obtained when inequalities (5.17) are introduced: for each instance the table shows the gap to the best known solution, the computing time in seconds and the number of cuts. The gap is reduced by 0.5% on average, but the computing time is 10 times larger. This behaviour

			DP		HDP			RGDP		
Instance	UB	LB	time	iter	time	iter	fail	time	iter	fail
AA30	31119.1	26171.0	1.65	107	1.53	103	1	3.22	222	107
AA35	31299.8	26361.3	4.31	136	3.80	128	1	7.65	362	97
AA40	31515.9	31506.6	5.29	152	5.02	145	6	9.81	346	131
AA45	31759.8	31759.8	8.56	164	9.17	149	1	13.03	346	108
AA50	41775.0	34092.9	11.64	148	13.00	157	6	20.67	350	116
AA55	41907.8	36988.9	13.79	164	10.67	135	3	23.26	367	141
AA60	42140.7	38887.7	14.05	125	16.33	139	1	28.32	375	115
AA65	42250.2	40017.2	19.78	126	21.09	134	3	35.18	351	100
AA70	42452.3	41239.3	21.55	109	39.21	148	2	38.87	356	79
AA75	52461.6	43163.7	28.10	87	58.22	154	3	54.32	350	69
BB30	31086.3	22628.2	1.10	61	1.27	71	1	2.05	160	49
BB35	31281.2	27057.8	2.04	64	2.07	59	1	3.64	143	49
BB40	31493.4	30303.7	3.29	87	3.34	78	1	6.41	230	63
BB45	41555.1	32753.3	5.94	87	6.97	88	3	9.53	199	62
BB50	41701.0	35928.4	6.00	58	12.95	95	3	11.58	180	43
BB55	41885.7	39781.5	16.17	109	16.77	84	1	19.39	226	61
BB60	62420.1	54925.4	7.26	104	8.83	104	3	18.21	324	78
BB65	62639.1	55910.3	7.42	74	11.27	99	3	17.52	250	56
BB70	62951.0	61532.9	8.85	72	16.23	117	1	23.29	281	57
BB75	63127.5	62229.8	13.52	91	22.18	129	1	33.84	330	63
CC30	31087.7	22810.1	5.70	98	4.66	99	5	10.79	240	48
CC35	31230.6	24248.1	12.19	113	10.34	83	2	27.42	310	63
CC40	31358.5	25289.4	58.61	112	38.56	93	6	71.95	299	94
CC45	31509.1	28939.1	198.40	121	66.15	90	3	128.76	315	54
CC50	41685.3	34058.2	44.26	102	59.95	90	3	51.56	245	48
CC55	41836.3	36425.0	75.58	114	93.77	118	4	80.01	281	63
CC60	42009.3	37838.7	113.24	109	148.19	118	2	128.08	314	59
CC65	42164.0	39480.2	162.46	86	241.72	99	1	162.02	299	63
CC70	52201.7	42116.0	316.28	107	374.05	98	1	311.07	375	70
CC75	52359.0	43562.3	375.02	144	681.53	133	4	332.89	446	82
DD30	21133.3	18702.2	17.05	69	18.09	75	5	28.05	182	43
DD35	31210.9	21524.9	25.64	70	43.83	94	10	42.34	210	50
DD40	31352.2	23138.2	63.76	78	94.50	95	5	99.13	245	68
DD45	31483.9	24872.1	131.53	102	210.37	112	6	154.96	260	64
DD50	31600.9	26587.2	235.32	124	283.00	127	7	243.42	324	85
DD55	31743.3	28831.3	310.05	101	412.85	106	2	286.19	277	61
DD60	32069.2	31458.3	237.65	90	689.97	118	2	263.76	261	61
DD65	42107.3	35313.1	210.87	97	378.63	126	5	156.39	299	60
DD70	42214.2	36690.6	716.85	119	834.35	154	5	634.30	411	75
DD75	42359.9	38759.5	518.75	127	851.77	153	3	514.14	456	68

Table 5.2: Pricing algorithm configuration on dataset RCL07, soft TW type I.

		Standard pricer			Aggregated Pricer		
Instance	LB	time	iter	cols	time	iter	cols
AA30	21006.1	9.67	121	3231	15.28	94	2802
AA35	24405.8	12.26	101	4905	23.72	109	4981
AA40	31204.6	15.03	104	3460	25.38	94	3132
AA45	31350.1	20.87	108	4257	46.12	100	4117
AA50	31446.4	39.29	111	5968	77.42	105	4789
AA55	33245.4	61.14	130	5889	163.82	128	5894
AA60	33382.4	85.26	133	8453	151.32	105	8153
AA65	33509.9	141.93	127	12735	358.54	135	11857
AA70	34178.3	279.75	150	11489	600.52	150	10111
AA75	35182.8	545.64	142	14702	1570.56	115	13509
BB30	22132.7	1.88	65	2437	2.06	61	2040
BB35	26218.2	3.38	56	3078	4.54	56	3027
BB40	27692.3	7.29	65	4299	9.64	62	3169
BB45	30153.7	8.91	58	5577	16.07	53	4798
BB50	34217.8	23.18	70	6239	49.14	57	5312
BB55	36945.9	25.77	69	6924	72.60	56	6259
BB60	50930.4	8.27	69	3373	13.68	79	3081
BB65	51652.5	10.88	62	5279	18.26	65	4718
BB70	55016.5	16.49	74	5719	30.78	84	5145
BB75	56069.9	24.45	89	6693	49.62	79	5996
CC30	19721.1	91.79	74	3618	353.83	84	3552
CC35	20196.1	536.71	106	4712	646.94	90	4136
CC40	20872.8	866.17	102	7048	3687.50	119	8316
CC45	22878.2	2495.52	116	9045	3640.08	65	8272
CC50	-	3700.36	145	10224	3739.88	63	10662
CC55	-	3604.37	97	11786	3669.76	51	8145
CC60	-	3631.25	85	9966	3652.47	49	11119
CC65	-	3704.41	58	10082	3694.24	40	12541
CC70	-	3669.85	64	9413	3700.38	40	12762
CC75	-	3706.14	64	10855	3706.14	40	8288
DD30	17742.6	64.84	65	3108	206.17	74	2819
DD35	20588.4	199.93	64	4144	383.39	51	3880
DD40	21705.1	707.28	94	4828	1350.68	87	5504
DD45	23102.3	1120.90	83	6189	2443.53	77	7536
DD50	24084.3	3234.46	95	8271	3689.20	102	9457
DD55	26164.7	2385.49	82	8778	3615.46	79	9635
DD60	28380.6	2624.36	89	16525	3630.05	73	16212
DD65	32106.5	3002.32	112	11553	3728.88	102	11192
DD70	-	3916.31	94	11117	3606.89	76	16478
DD75	-	3609.06	109	13945	3655.90	70	14128

Table 5.3: Standard Pricer and Aggregated Pricer on dataset RCL07_H.

is in line with the statement in [3]. Thus, we decided not to use these inequalities in the full branch-and-price algorithm.

5.4.2 Branch-and-price

The second set of experiments concerns the branch-and-price algorithm. It uses the heuristic pricer with the restricted graph (RGDP) combined with the exact dynamic programming. We consider dataset RCL07 with the three different types of time windows (type I, II and III), where type I coincides with hard time windows, and the other two types correspond to soft time windows. In Tables 5.5, 5.6 and 5.7 we show the results obtained by the branch-and-price algorithm on instances featuring time windows of type I, II, and III, respectively. We show average results for each class of instances: in particular, we present the number of solved instances (out of 10), the average percentage gap (for the instances not solved to optimality), the average computing time and average number of explored nodes (for the instances solved to optimality). As one can see, the number of solved instances is basically the same despite the considered type of time windows. Thus, the approach is robust with respect to the specific penalty function considered. The percentage gap can be high for some instances. Indeed, no sophisticated primal heuristic has been introduced, as this was not the main focus of this work. Thus, this gap is probably due to poor upper bound (rather than weak lower bound). In addition, these instances have very high fixed cost of the vehicles: as a consequence, a difference of a single vehicle between the primal and the dual solutions is enough to produce a gap as large as 20-30%.

The approach by Ropke and Cordeau [74] and the one by Baldacci et al. [3] study the pickup and delivery problem with hard time windows. The case of hard time windows is considered in our approach, as well (i.e. time windows of type I). However, the goal of our work is not to design an algorithm for this specific problem, as we want to deal with a more general setting. Testing the behavior of our method on type I time windows is especially to show that our algorithm is stable under different setting, and hard time windows can

	LB		LBC		
Instance	Gap %	Time	Gap %	Time	# of cuts
AA30	18.91	1.65	17.33	120.57	1335
AA35	18.73	4.31	15.60	290.52	1419
AA40	0.03	5.29	0.00	6.70	35
AA45	0.00	8.56	0.00	8.56	0
AA50	22.53	11.64	22.46	52.87	558
AA55	13.30	13.79	12.62	40.99	1264
AA60	8.37	14.05	8.26	56.74	509
AA65	5.58	19.78	5.57	29.10	65
AA70	2.94	21.55	2.84	64.61	126
AA75	21.54	28.1	21.42	81.37	95
BB30	37.38	1.1	35.51	10.07	2061
BB35	15.61	2.04	11.77	34.79	611
BB40	3.93	3.29	3.32	67.73	619
BB45	26.87	5.94	26.47	186.94	208
BB50	16.07	6	14.93	61.63	271
BB55	5.29	16.17	5.05	59.61	137
BB60	13.65	7.26	12.93	13.08	659
BB65	12.03	7.42	11.92	12.46	336
BB70	2.30	8.85	1.92	22.54	268
BB75	1.44	13.52	1.09	50.43	273
CC30	36.29	5.7	35.24	3635.80	250
CC35	28.80	12.19	28.02	3285.13	666
CC40	24.00	58.61	23.39	3653.78	734
CC45	8.88	198.4	8.88	4145.11	194
CC50	22.39	44.26	22.11	3865.61	323
CC55	14.86	75.58	14.51	471.38	97
CC60	11.02	113.24	10.88	595.09	232
CC65	6.80	162.46	6.80	162.46	0
CC70	23.95	316.28	23.95	316.28	0
CC75	20.19	375.02	20.16	585.73	39
DD30	13.00	17.05	11.16	139.72	1254
DD35	45.00	25.64	44.59	3768.03	399
DD40	35.50	63.76	35.02	1935.08	833
DD45	26.58	131.53	26.46	1002.08	164
DD50	18.86	235.32	18.73	1146.29	126
DD55	10.10	310.05	10.04	796.59	48
DD60	1.94	237.65	1.92	313.36	22
DD65	19.24	210.87	19.24	210.87	0
DD70	15.05	716.85	14.89	3432.59	49
DD75	9.29	518.75	9.29	1002.78	9

Table 5.4: Lower bound with (LB) and without (LBC) inequalities 5.17 on dataset RCL07, soft TW type I.

Class	Solved	Avg. gap %	Avg. time	Avg. nodes
AA	8/10	2.87	393.82	68.75
BB	8/10	9.25	43.06	28.00
CC	6/10	18.43	1385.36	129.83
DD	4/10	25.52	808.77	83.00

Table 5.5: BCP average results, dataset RCL07, type I

Class	Solved	Avg. gap %	Avg. time	Avg. nodes
AA	8/10	30.88	299.78	15.25
BB	9/10	0.39	913.02	27.22
CC	7/10	24.43	1327.70	35.14
DD	4/10	173.3	1185.13	21.50

Table 5.6: BCP average results, dataset RCL07, type II

be seen as an extreme case of soft time windows. We mention that Ropke and Cordeau ([74]) solve 30 out of 40 instances to optimality, while Baldacci et al. ([3]) are able to solve 39 of them in some hours of computing time. Our algorithm has a similar outcome to that of Ropke and Cordeau ([74]), as it solves 26 instances within one hour.

In Table 5.8 the same information is shown for instances RCL07_H. As mentioned before, these instances are more involved, thus we are able to solve to optimality a smaller number of them: not only the root node lower bound is hard to be computed, but the branching on the number of vehicles is less effective in presence of an heterogeneous fleet.

Class	Solved	Avg. gap %	Avg. time	Avg. nodes
AA	8/10	14.41	422.56	22.88
BB	9/10	14.15	478.90	36.56
CC	5/10	22.18	258.66	26.40
DD	4/10	35.45	959.64	12.00

Table 5.7: BCP average results, dataset RCL07, type III

Class	Solved	Avg. gap %	Avg. time	Avg. nodes
AA	6/10	29.07	1292.77	1292.77
BB	4/10	9.32	483.57	483.57
CC	2/10	19.24	2215.63	2215.63
DD	2/10	60.97	1016.47	1016.47

Table 5.8: BCP average results, dataset RCL07_H.

	Type I	Type II		Type III	
Class	Routing cost	Routing cost	Avg Δ%	Routing cost	Avg Δ%
AA	1691.30	1786.71	5.34%	1786.90	5.35%
BB	1757.75	1870.61	6.03%	1889.72	6.98%
CC	1439.69	1631.44	11.75%	1626.18	11.47%
DD	1232.14	1445.38	14.75%	1418.06	13.11%

Table 5.9: Change in the routing costs of the optimal solutions values with different time windows configurations.

5.4.3 Impact of soft time windows

The last set of experiments was conducted in order to understand the impact of the soft time windows on the optimal solutions. In the first test we analyze how the routing cost changes when soft time windows are introduced with respect to a scenario where only hard time windows are imposed. In Table 5.9, we show average results for the four classes of instances. For each class, we present the routing cost for each type of time windows and, for type II and III (i.e. soft time windows), the percentage difference with respect to type I (i.e. hard time windows). The routing cost increases of up to 15% when dealing with soft time windows, since in this case one is willing to travel longer distances in order to reduce the penalty cost. We recall that in this test instances soft time windows are cut inside the hard ones and, thus, they are narrower. Hence, in a context where soft time windows do not represent a real cost, but a customer preference, the penalty functions must be carefully weighted, since even moderate panalties for soft time windows violation, may produce significant increase in routing cost.

Let us now consider a different point of view: since it is complicated to

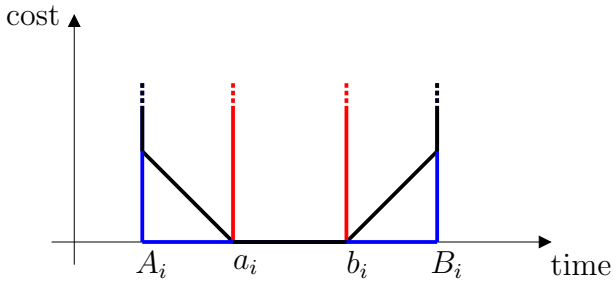

Figure 5.7: Representation of soft time window (black), outer hard time window (blue) and inner hard time window (red) at a generic vertex $i \in V$.

	soft TW	outer		inner	
Class	value	value	loss	value	loss
AA	38675.92	39190.25	514.33	61866.96	22462.43
BB	40540.02	41107.18	567.16	65354.56	19692.36
CC	33960.45	35112.73	1152.29	50284.82	13692.09
DD	28268.47	29447.99	1179.52	39052.05	9777.47

Table 5.10: Impact of soft time windows measured by the increase in the objective function if inner or outer hard time windows are used.

tackle soft time windows, one may think to approximate the problem using only hard time windows. We consider instances RCL07 with time windows type II: each vertex $i \in V$ has a hard time window $[A_i, B_i]$ and a smaller soft time window $[a_i, b_i]$. We solve to optimality a problem with only hard time windows $[A_i, B_i]$, which we call outer time windows, and a problem with only hard time windows $[a_i, b_i]$, which we call inner time window (see Figure 5.7). Then we evaluate the solutions found with respect to the real penalty functions and we compare them to the optimal solution obtained when solving the problem with soft time windows. The results are shown in Table 5.10. For each class of instances we show the optimal objective function value obtained for the problem with soft time windows, outer time windows and inner time windows. In addition we show, for outer and inner time windows, the increase of cost (loss) that is caused by neglecting the soft time windows. As it can be seen the loss is significant, especially for inner time windows, where it is often necessary to increase the number of vehicles used in order to satisfy the strict time constraints.

5.5 Conclusion

In this work we have proposed an exact method to solve an important optimization problem in distribution logistics, namely the MDHPHPSTW. In particular, we have modified a known branch-and-price scheme to handle different constraints, such as soft time windows. Our experiments have shown that the approach can solve instances with up to 70 customers. We have also presented an analysis of the impact of the soft time windows on the optimal solution both in terms of routing and overall costs.

Chapter 6

Conclusions

In this thesis, we have studied transportation problems, that extend classical routing problems by introducing more realistic features, such as multiple depots instead of just one depot where all the vehicles are based, a heterogeneous fleet of vehicles (i.e. vehicles with different capacities), soft time windows (i.e. time windows where a linear penalty is applied if the vehicle visits the customer outside the specified time window) and pickup and delivery to the customers instead of just pickup. The first studied problem is the Multi-Depot Heterogeneous Vehicle Routing Problem with Time Windows and the second one is the Multi-Depot Heterogeneous-Fleet Pickup and Delivery Problem with Soft Time Windows. To the best of our knowledge, no exact method was designed for routing problems including all these additional features. In order to tackle these very general routing problems, we have extended an effective solution approach introduced by Righini and Salani, by applying relevant changes both in the structure of the algorithm and in the data structure implementation. Extensive computational experiments have been performed in order to compare the proposed algorithms with state-of-the-art algorithms that however do not include these additional features. In addition, computational testing has been done on more realistic instances, containing the described additional characteristics. The outcome of the proposed approaches is promising on both computational experiments, and encouraging in further extending the class of studied problems. Future

research can be devoted to the study of real-world problems, that can be characterized by more involved constraints and larger instances. The presented methods can be applied and modified either for finding optimal solutions, or for heuristically solving more involved problems. In addition, the experience gained during the PhD study will be applied to different fields of Operations Research, characterized by large scale problems where branch-and-price approaches are more effective.

Bibliography

[1] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007.

[2] N Balakrishnan. Simple heuristics for the vehicle routing problem with soft time windows. *Journal of the Operational Research Society*, 44:279–287, 1993.

[3] R. Baldacci, E. Bartolini, and A. Mingozzi. An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, to appear.

[4] R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Math. Prog.*, 115:351–385, 2007.

[5] R. Baldacci and A. Mingozzi. A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming*, to appear, 2008.

[6] P.P. Belfiore and L.P.L. Fvero. Scatter search for the fleet size and mix vehicle routing problem with time windows. *Central European Journal of Operations Research*, 15:351–368, 2007.

[7] R. Bent and P.V. Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893, 2006.

[8] J. Berger and M. Barkaoui. A new hybrid genetic algorithm for the capacitated vehicle routing problem. *Journal of the Operational Research Society*, 54(12):1254–1262, 2003.

[9] A. Bettinelli, A. Ceselli, and G. Righini. A branch-and-price algorithm for the multi-depot heterogeneous fleet vehicle routing problem with time windows. AIRO Winter, January 2009.

[10] A. Bettinelli, A. Ceselli, and G. Righini. A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transportation Research C*, to appear, 2010. DOI: 10.1016/j.trc.2010.07.008.

[11] A. Bettinelli, A. Ceselli, and G. Righini. Branch-and-price for the multi-depot pickup and delivery problem with heterogeneous fleet and soft time windows. In *EURO XXIV*, Lisbon, 2010.

[12] A. Ceselli, G. Righini, and M. Salani. A column generation algorithm for a rich vehicle-routing problem. *Transportation Science*, 43(1):56–69, 2009.

[13] W.C. Chiang and R.A. Russell. A metaheuristic for the vehicle-routeing problem with soft time windows. *Journal of the Operational Research Society*, 55:1298–1310, 2004.

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

[15] J.-F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54:573–586, 2006.

[16] J.F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, and J.S. Sormany. New heuristics for the vehicle routing problem. *Logistics systems: design and optimization*, pages 279–297, 2005.

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

[18] J.F. Cordeau, G. Laporte, and S. Ropke. Recent models and algorithms for one-to-one pickup and delivery problems. In B.l. Golden, S. Raghavan, and E.A. Wasil, editors, *Vehicle Routing: Latest Advances and Challenges*, pages 327–357. Springer, 2008.

[19] J.F. Cordeau, G. Laporte, M.W.P. Savelsbergh, and D. Vigo. *Vehicle routing*, volume 14, page 367. North-Holland, 2007.

[20] G.B. Dantzig and J.H. Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[21] M. Dell'Amico, M. Monaci, C. Pagani, and D. Vigo. Heuristic approaches for the fleet size and mix vehicle routing problem with time windows. *Transportation Science*, 41(4):516 – 526, 2007.

[22] M. Dell'Amico, G. Righini, and M. Salani. A branch-and-price algorithm for the vehicle routing problem with simultaneous pick-up and delivery. *Transportation Science*, 40:235–247, 2006.

[23] U. Derigs and T. Döhmer. Indirect search for the vehicle routing problem with pickup and delivery and time windows. *OR Spectrum*, 30(1):149–165, 2008.

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

[25] G. Desaulniers, J. Desrosiers, and M. Solomon. Accelerating strategies in column generation methods for vehicle routing and crew scheduling. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 309–324, 2002.

[26] G. Desaulniers, J. Desrosiers, and M.M. Solomon. *Column Generation*. Springer, 2005.

[27] G. Desrochers. An algorithm for the shortest path problem with resource constraints. Technical report, 1988.

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

[29] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–353, 1992.

[30] J. Desrosiers, Y. Dumas, M.N. Solomon, and F. Soumis. Time constrained routing and scheduling. In *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, pages 35–139. INFORMS - North Holland, 1995.

[31] J. Desrosiers and M.E. Lübbecke. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

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

[33] R. Dondo and J. Cerdá. A cluster-based optimization approach for the multi-depot heterogeneous fleet vehicle routing problem with time windows. *European Journal of Operational Research*, 176:1478–1507, 2007.

[34] R. Dondo and J. Cerdá. A hybrid local improvement algorithm for large-scale multi-depot vehicle routing problems with time windows. *Computers & Chemical Engineering*, to appear.

[35] J.J. Dongarra. Performance of various computers using standard linear equations software. Technical report, University of Tennessee, 2009. available at http://www.netlib.org/benchmark/performance.ps.

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

[37] M. Dror. Note on the complexity of the shortest path models for column generation in vrptw. *Operation Research*, 42(5):977–978, 1994.

[38] W. Dullaert, G.K. Janssens, K. Sorensen, and B. Vernimmen. New heuristics for the fleet size and mix vehicle routing problem with time windows. *Journal of the Operational Research Society*, 53:1232–1238, 2002.

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

[40] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54:7–22, 1991.

[41] Ö. Ergun, J.B. Orlin, and A. Steele-Feldman. Creating very large scale neighborhoods out of smaller ones by compounding moves. *Journal of Heuristics*, 12(1):115–140, 2006.

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

[43] R. Fukasawa, H. Longo, J. Lysgaard, M.P. Aragão, M. Reis, E. Uchoa, and R.F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical programming*, 106(3):491–511, 2006.

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

[45] P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

[46] B. Golden, S. Raghavan, and Edward Wasil, editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008.

[47] B.L. Golden, A. Assad, L. Levy, and F. Gheysens. The fleet size and mix vehicle routing problem. *Computers and Operations Research*, 11:49–65, 1984.

[48] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno, and M. Yagiura. Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science*, 39:206–232, 2005.

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

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

[51] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer Verlag, 2005.

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

[53] Y.A. Koskosidis, W.B. Powell, and M.M. Solomon. An optimization based heuristic for vehicle routing and scheduling with soft time window constraints. *Transportation Science*, 26:69–85, 1992.

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

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

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

[57] F. Liberatore, M. Salani, and G. Righini. A pricing algorithm for the vehicle routing problem with soft time windows. In L. Bertazzi, M.G. Speranza, and J.A.E.E. van Nunen, editors, *Proceedings of the International Workshop on Distribution Logistics 2006*, volume 619, pages 251–266, Brescia, 2009.

[58] F.H. Liu and S.Y. Shen. The fleet size and mix vehicle routing problem with time windows. *Journal of the Operational Research Society*, 50:721–732, 1999.

[59] Q. Lu and M.M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 175(2):672–687, 2006.

[60] S. Martello and P. Toth. *Knapsack problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.

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

[62] D. Naddef and G. Rinaldi. *Branch-and-cut algorithms for the capacitated vrp*, chapter 3, pages 53–81. SIAM monograph on discrete mathematics and applications, 2002.

[63] G. Pankratz. A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum*, 27(1):21–41, 2005.

[64] S. Parragh, K. Doerner, and R. Hartl. A survey on pickup and delivery problems: Part ii: Transportation between pickup and delivery locations. *Journal für Betriebswirtschaft*, 58, 2008.

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

[66] M. Polacek, Richard F. Hartl, K. Doerner, and M. Reimann. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10:613–627, 2004.

[67] N. Prindezis and C.T. Kiranoudis. An internet-based logistics management system for enterprise chains. *Journal of Food Engineering*, 70(3):373 – 381, 2005.

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

[69] A.G. Qureshi, E. Taniguchi, and T. Yamada. An exact solution approach for vehicle routing and scheduling problems with soft time windows. *Transportation Research Part E: Logistics and Transportation Review*, 45:960–977, 2009.

[70] C. Ribeiro and F. Soumis. A column generation approach to the multiple-depot vehicle scheduling problem. *Operations Research*, 42:41–52, 1994.

[71] G. Righini and M. Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 30(3):255–273, 2006.

[72] G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, 51:155–170, 2008.

[73] S. Ropke and J.-F. Cordeau. Branch-and-cut-and-price for the pickup and delivery problem with time windows. Technical report, HEC Montral, Montral, Quebec, Canada, 2008.

[74] S. Ropke and J.F. Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *TRANSPORTATION SCIENCE*, 43:267–286, 2009.

[75] S. Ropke, J.F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49:258–272, 2007.

[76] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[77] L.M. Rousseau, M. Gendreau, and D. Feillet. Interior point stabilization for column generation. *Operations Research Letters*, 35(5):660–668, 2007.

[78] M. Salani. *Branch-and-Price Algorithms for Vehicle Routing Problems*. PhD thesis, Università degli Studi di Milano, 2006.

[79] L. Santos, J. Coutinho-Rodrigues, and J. R. Current. Implementing a multi-vehicle multi-route spatial decision support system for efficient trash collection in portugal. *Transportation Research Part A: Policy and Practice*, 42(6):922 – 934, 2008.

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

[81] M.W.P. Savelsbergh and M. Sol. Drive: Dynamic routing of independent vehicles. *Oper. Res.*, 29:474–490, 1998.

[82] T.R. Sexton and Y.M. Choi. Pickup and delivery of partial loads with soft time windows. *American Journal of Mathematical and Management Sciences*, 6:369–398, 1986.

[83] M.M. Solomon. Algotithms for the vehicle routing and scheduling problems with time windows constraints. *Operations Research*, 35:254–265, 1987.

[84] E Taillard, P Badeau, M Gendreau, F Guertin, and J.Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31:170–186, 1997.

[85] CD Tarantilis and CT Kiranoudis. Boneroute: an adaptive memory-based method for effective fleet management. *Annals of Operations Research*, 115(1):227–241, 2002.

[86] P. Toth and A. Tramontani. An integer linear programming local search for capacitated vehicle routing problems. *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 275–295, 2008.

[87] P. Toth and D. Vigo. *Branch-and-bound algorithms for the capacitated vrp*, chapter 2, pages 29–49. SIAM monograph on discrete mathematics and applications, 2002.

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

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

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

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

[92] H. Xu, Z.L. Chen, S. Rajagopal, and S. Arunapuram. Solving a practical pickup and delivery problem. *Transportation Science*, 37(3):347–364, 2003.