Pontifícia Universidade Católica
do Rio de Janeiro

Diego Galindo Pecin

Exact Algorithms for the Capacitated Vehicle Routing Problem

TESE DE DOUTORADO

Thesis presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática, PUC-Rio as partial fulfillment of the requirements for the degree of Doutor em Informática.

Advisor: Prof. Marcus Vinicius Soledade Poggi de Aragão
Co-Advisor: Prof. Eduardo Uchoa Barboza

Rio de Janeiro
April 2014

Pontifícia Universidade Católica
do Rio de Janeiro

Diego Galindo Pecin

Exact Algorithms for the Capacitated Vehicle Routing Problem

Thesis presented to the Programa de Pós–Graduação em Informática, of the Departamento de Informática do Centro Técnico Científico da PUC-Rio, as partial fulfillment of the requirements for the degree of Doutor.

Prof. Marcus Vinicius Soledade Poggi de Aragão
Advisor
Departamento de Informática – PUC-Rio

Prof. Eduardo Uchoa Barboza
Co–Advisor
UFF

Prof. A. Ridha Mahjoub
LAMSADE

Prof. Abilio Pereira de Lucena Filho
UFRJ

Prof. Paolo Toth
UNIBO

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática – PUC-Rio

Prof. Eduardo Sany Laber
Departamento de Informática – PUC-Rio

Prof. José Eugenio Leal
Coordinator of the Centro Técnico Científico da PUC-Rio

Rio de Janeiro, April 25th, 2014

Diego Galindo Pecin

Diego Galindo Pecin obtained a MSc in Computer Science from Universidade Federal de Goiás (UFG) and a BSc in Computer Science from the same university. During his doctoral studies he held a scholarship from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). He has experience as a teaching assistant in graduate courses at PUC-Rio and as a temporary teacher at Universidade Federal Fluminense (UFF).

Bibliographic Data

Pecin, Diego

Exact Algorithms for the Capacitated Vehicle Routing Problem / Diego Galindo Pecin; advisor: Marcus Vinicius Soledade Poggi de Aragão; co–advisor: Eduardo Uchoa Barboza. – 2014.

116 f. : il. (color); 30 cm

Tese (Doutorado em Informática) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2014.

Inclui bibliografia

1. Informática – Teses. 2. Problemas de Roteamento. 3. Geração de Colunas. 4. Separação de Cortes. 5. Branch-Cut-and-Price. 6. Programação Inteira. 7. Engenharia de Algoritmos. I. Poggi, Marcus. II. Uchoa, Eduardo. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

Acknowledgments

To my parents, João and Alexânia, for the love and teaching they have given me, which is still such a big part of who I am today.

To Thiago, my older brother, for being my family support in Rio de Janeiro and for his ever lasting friendship.

To my dear brothers Giselle, João Lucas, Guilherme and Fillipe, for their friendship and for always being there. To the loving memory of my grandmother Eni, and my cousin, Alex, for so many great times.

To my advisor Marcus Poggi, for the opportunity to work at PUC-Rio, for the trust he has placed in me and for showing me that hard work pays off. And for always being easy-going and making me see academic life in an easier, more relaxed way.

To my co-advisor Eduardo Uchoa, for his deep involvement, every day, with the main issue addressed in this thesis, CVRP. For the tireless meetings and thousands of emails exchanged throughout this work, the tests performed assiduously in code and for showing me how to improve where necessary. And for always believing that we would get the desired results and passing on this confidence to me.

To Humberto Longo, my MSc advisor, for introducing me to and guided me in my first CVRP-related work.

To my doctoral colleague Rafael Martinelli for constantly working with me in the first two years of my PhD which resulted, among other publications, in a significant portion of the content described in Chapter 5 of this thesis.

To Artur Pessoa, for working with me on CVRP, especially for the technical support and cooperation in coding.

To Claudio Contardo whose previous work in CVRP was fundamental to the development of this thesis. And for being so kind to have come to Rio de Janeiro in late 2012 to remedy our questions concerning this work.

To the entire board of examiners for accepting to evaluate this work, in particular to Professor Paolo Toth for having come from Italy for this purpose.

To my friends Renato, Rodson, Rafael Augusto, Emerson, Bruno and Rodrigo for their longstanding friendship and the good times we have shared.

To the friends I have made in Rio de Janeiro, in particular Joabson, Christiano, Zé, Achá, Diego, Pantoja and Iran for the friendship they have shown along this journey.

To the friends of PUC-Rio, especially Mayra, Rafael Martinelli, Thuener and Aline, for those great moments of relaxation.

To PUC-Rio, for offering me great working conditions, and to CNPq for their financial support.

Abstract

Pecin, Diego; Poggi, Marcus; Uchoa, Eduardo. **Exact Algorithms for the Capacitated Vehicle Routing Problem**. Rio de Janeiro, 2014. 116p. DSc Thesis — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Vehicle Routing Problems are among the most difficult combinatorial problems to solve to optimality. They were proposed in the late 1950's and since then have been widely studied. This interest arises from their practical importance, as well as the difficulty of providing efficient algorithms to solve them. This thesis is mainly concerned with the exact resolution of the Capacitated Vehicle Routing Problem (CVRP). In this problem, a set of customers, each one associated to a demand, must be serviced by a fleet of vehicles. All vehicles have the same (limited) capacity and initially are located in the same central depot. A solution is a set of routes, starting and ending at the depot, that visit every customer exactly once. The only constraint on a route is that the sum of the demands of its customers does not exceed the vehicle capacity. The objective is to find a solution with minimum total cost. The best performing exact algorithms for the CVRP developed in the last 10 years are based in the combination of cut and column generation. Some authors only used cuts expressed over the variables of the original formulation, in order to keep the pricing subproblem relatively easy. Other authors could reduce the duality gaps by also using a restricted number of cuts over the Master LP variables, stopping when the pricing becomes prohibitively hard. A particularly effective family of such cuts are the Subset Row Cuts. This thesis introduces a technique for greatly reducing this impact on the pricing of these cuts, thus allowing much more cuts to be added. The newly proposed Branch-Cut-and-Price algorithm also incorporates and combines for the first time (often in an improved way) several elements found in previous works, like route enumeration, variable fixing and strong branching. All the instances used for benchmarking exact algorithms, with up to 199 customers, were solved to optimality. Moreover, some larger instances with up to 360 customers, only considered before by heuristic methods, were solved too.

Keywords

Routing Problems; Column Generation; Cut Separation; Branch-Cut-and-Price; Integer Programming; Algorithmic Engineering.

Resumo

Pecin, Diego; Poggi, Marcus; Uchoa, Eduardo. **Algoritmos Exatos para o Problema de Roteamento de Veículos Capacitado**. Rio de Janeiro, 2014. 116p. Tese de Doutorado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Os Problemas de Roteamento de Veículos estão entre os problemas combinatoriais mais difíceis de se resolver à otimalidade. Eles foram propostos no final da década de 1950, e desde então eles têm sido amplamente estudados. O interesse deve-se a sua importância prática, bem como da dificuldade de se fornecer algoritmos eficientes para resolvê-los. Esta tese trata principalmente da resolução exata do Problema de Roteamento de Veículos com Capacidades (PRVC). Neste problema, um conjunto de clientes, cada um associado a uma demanda, deve ser atendido por uma frota de veículos. Todos eles têm o mesma capacidade e, inicialmente, estão localizados no mesmo depósito. Uma solução é um conjunto de rotas que começam e terminam no depósito e visitam cada cliente uma única vez. A restrição em uma rota é que a soma das demandas de seus clientes não exceda a capacidade do veículo. O objetivo é encontrar uma solução com um custo mínimo. Os melhores algoritmos exatos para o PRVC desenvolvidos nos últimos dez anos são baseados na combinação de geração de cortes e colunas. Alguns autores utilizaram apenas cortes sobre as variáveis da formulação original, com a finalidade de manter o subproblema de geração de colunas relativamente fácil. Outros puderam reduzir os limites duais utilizando também um número restrito de cortes expressos nas variáveis do problema mestre, parando de incluir tais cortes quando o subproblema tornava-se proibitivamente difícil. Uma família eficaz de tais cortes são os *Subset Row Cuts*. Esta tese apresenta uma técnica para reduzir consideravelmente o impacto que tais cortes causam no subproblema de geração de colunas, permitindo assim que muito mais cortes sejam adicionados. O novo algoritmo *Branch-Cut-and-Price* proposto também incorpora e combina pela primeira vez vários elementos presentes em trabalhos anteriores, como enumeração de rotas, fixação de variáveis e *strong branching*. Todas as instâncias usadas em algoritmos exatos, com até 199 clientes, foram resolvidas à otimalidade. Além disso, algumas maiores, com até 360 clientes, apenas consideradas antes em métodos heurísticos, também foram resolvidas.

Palavras–chave

Problemas de Roteamento; Geração de Colunas; Separação de Cortes; Branch-Cut-and-Price; Programação Inteira; Engenharia de Algoritmos.

Contents

List of Figures

List of Tables

Before going to sleep, ask what you have done for the CVRP during the day, if it was nothing, it was a wasted day!

Eduardo Uchoa Barboza.

1
Introduction

Vehicle Routing Problems (VRPs) correspond to an important class of combinatorial optimization problems that aim at determining an optimal set of routes for a fleet of vehicles. Initially, the vehicles are considered to be located in one or more depots in order to deliver goods to a set of customers. In general, a solution requires finding routes, that begin and end in the same depot, such that all customers' demands are serviced and all operational constraints are satisfied. Some typical constraints are limited capacity of the vehicles, customers requiring delivery or collection of goods (in some cases both), and periods of the day on which a customer can be serviced. The overall goal is to minimize the operational cost of the transport. For instance, this cost can be a function of the total distance or time traveled.

VRPs play a central role in fields such as transportation, distribution and logistics. They were first introduced in the literature by Dantzig and Ramser (14) in 1959, when a problem of oil distribution from supply stations was studied. It is remarked in Toth and Vigo (59) that a high percentage of the value added to commercial goods comes from the costs related to their transportation (between 5% to 20%). This suggests that the use of computational methods for planning the transport may often result in significant savings.

Since the work in (14) the VRPs have become widely studied. This interest arises from their practical importance, they can model many real-world logistic problems, as well as the difficulty of providing efficient algorithms to solve them. Note that part of the difficulty of solving VRPs comes from the fact that it generalizes the well-known Traveling Salesman Problem (TSP), a strongly $\mathcal{NP}$-Hard problem. Moreover, depending on the constraints involved in the routing problem, only to prove that a given problem has a feasible solution may require solving of a difficult combinatorial problem. For instance, determining whether a given set of vehicles can service all customers is equivalent to solving a Bin Packing Problem, which is also known to be $\mathcal{NP}$-Hard.

The Capacitated Vehicle Routing Problem (CVRP) is the basic variant introduced in (14). It can be described as follows. A set of customers, each one

associated to a demand, must be serviced by a fleet of vehicles. All vehicles have the same (limited) capacity and initially are located in the same central depot. A solution is a set of routes, starting and ending at the depot, that visit every customer exactly once. The only constraint on a route is that the sum of the demands of its customers does not exceed the vehicle capacity. The objective is to find a solution with minimum total cost. Many authors also assume that the *number of routes* is fixed to an additional input number. The CVRP is a widely studied problem due to its own applications, since it can model adequately a significant number of real logistic systems. Furthermore, it plays a particularly important role on general vehicle routing research, for being the most basic and prototypical variant. This thesis is mainly concerned with the exact resolution of the CVRP, i.e. in proving a given solution is, in fact, optimal for this combinatorial problem.

A short list of important VRPs that have been studied in the last decades follows. The Vehicle Routing Problem with Time Windows (VRPTW) corresponds to a variation of the CVRP that restricts customer's services to occur inside given time windows, vehicles also have identical limited capacity. The Multi Depot Capacitated Vehicle Routing Problem (MDCVRP) is the variation that considers more than one depot. Another variation is the Vehicle Routing Problem with Simultaneous Pick-up and Delivery (VRPSPD) where, as the name says, customers can be served by a delivery, a pick-up or both. This problem includes the CVRP as the special case where there are only pick-up's (or deliveries). In the Heterogeneous Fleet Vehicle Routing Problem (HFVRP) vehicles may have distinct capacities and costs. Finally, the Generalized Vehicle Routing Problem (GVRP) considers clusters of customers. Each cluster has an associated demand and can contain one or more customers. The demand of each cluster must be fully collected in exactly one customer of the cluster. This problem is a generalization of the CVRP and, as in the classical CVRP, identical vehicles are given, routes must start and end at the depot and the capacity of the vehicle must not be exceeded.

Several exact methods have been proposed to solve VRPs in the last three decades. Branch-and-Bound is a widely used approach to solve many combinatorial problems such as the VRPs. An algorithm based on this approach uses a search tree to enumerate all possible feasible solutions of the problem being optimized, where each tree node represents a part of the solution space. Starting on the root node, a problem can be divided into two by splitting the feasible region of the problem. This procedure is called branching. On each node of the search tree one can solve some relaxation of the problem to calculate a bound on the value of an optimal solution for this node. For instance, a

bound can be obtained solving the Linear Programming (LP) relaxation of a Mixed Integer Programming (MIP) formulation, which is obtained by relaxing the integrality of its integer variables. This procedure is called bounding. A lower bound may allow stopping the search on a given branch of the tree by indicating that no solution of an interesting (smaller) value can be found. The fundamental idea of the Branch-and-Bound algorithm is to safely discard a node if its lower bound is greater or equal to the best known upper bound (the value of a solution) for the problem. This step is called pruning. This allows to discard large subsets of fruitless candidate solutions.

The current most efficient approaches for VRPs are those based on solving MIP formulations. One way to improve bounds from relaxed MIPs is to add to the formulation cutting planes, i.e. constraints that are satisfied by all integer feasible solutions but may not be satisfied by some continuous solutions of the relaxed MIP. Suppose one solves the linear relaxation of the formulation. The obtained optimum is tested for being an integer feasible solution. If it is not, a linear inequality that separates this fractional optimum from the set of feasible solutions must exist. These inequalities are called violated inequalities. Finding such an inequality is the goal of a so called separation problem and such violated inequality is called a cut. Cuts can be added to the relaxed linear program. Then, the current non-integer solution is no longer feasible to the relaxation. This process can be repeated until an optimal integer solution is reached or no violated inequalities is found. Note that the value of the optimal solution of the relaxed problem defines a lower bound, possibly stronger (greater) than the first bound obtained without cuts. When it is not possible to find additional violated inequalities, the algorithm proceeds by branching, like in the Branch-and-Bound approach. An algorithm that combines cutting planes with Branch-and-Bound is called Branch-and-Cut. These algorithms are among the best methods for solving arbitrary MIP formulations because general cutting planes have been widely studied (see, for example, Nemhauser and Wolsey (44) and Wolsey (62)).

However, nowadays the best results for VRPs are obtained by column generation techniques. These techniques are used to solve formulations with arbitrarily many variables, or columns as they are referred within this technique. It was introduced in 1958 by Ford and Fulkerson (23) for the multicommodity flow problem and it was later generalized to linear programming by Dantzig and Wolfe in 1960 (15). The algorithm starts with a small number of columns, which gives rise to a small linear program called restricted master (or Master LP). Next, the algorithm tries to find new columns suitable to improve the current solution. Finding such columns corresponds to solve an auxiliary problem

called pricing subproblem. This iterative process continues until no improving columns are found. The huge number of columns considered in column generation algorithms correspond to the exponentially many variables obtained when Dantzig-Wolfe decomposition is applied to MIP formulations (see Chapter 2 for more details). Since column generation is used to solve a linear program, the relaxed MIP, its optimal solution is a lower bound for the formulation and thus can be fractional. Therefore, to ensure that a feasible integer solution is found, the technique must be combined with Branch-and-Bound. The resulting algorithm is called Branch-and-Price (see, for example, Desaulniers, Desrosiers and Solomon (16), Lübbecke and Desrosiers (36)).

Branch-and-Bound can be used together with column and cut generation. This combination is called *Branch-Cut-and-Price* (BCP). The focus of this thesis is to develop BCP algorithms for the CVRP.

The CVRP can be formally defined on a complete undirected graph $G = (V, E)$, where $V = \{0, 1, \ldots, n\}$ is the set of vertices and E is the set of edges. The vertices from set $V_+ = V \backslash \{0\}$ represent *customers* and the vertex labeled 0 represents the *depot*. Each customer $i \in V_+$ has an associated integer demand $d_i > 0$ and the depot has a fictitious demand $d_0 = 0$. At the depot are located K vehicles, each one having the same integer *capacity* Q. Additionally, each edge $(i, j) \in E$ has an associated cost $c_{ij} \geq 0$, expressing the cost (distance or time, for example) to go from vertex i to vertex j. The cost structure is assumed symmetric, i.e., $c_{ij} = c_{ji}$ and $c_{ii} = 0$. The CVRP consists of finding K routes satisfying the following constraints: (i) each route starts and ends at the depot; (ii) each customer is visited by exactly one route, and (iii) the total demand of all customers in any route is at most Q. The goal is to minimize the sum of the lengths of all routes.

1.1 Brief History of Exact Algorithms

There is a wealth of publications on exact algorithms for the CVRP. Christofides, Mingozzi and Toth (11) presented in 1981 a landmark exact algorithm that uses Lagrangean bounds. One of these bounds is based on *q-routes*, a relaxation of a feasible CVRP route that allows routes repeat customers as long as the vehicle capacity constraint is satisfied. Therefore, the set of valid CVRP routes is strictly contained in the set of q-routes. The Branch-and-Bound algorithm obtained could solve to optimality instances with up to 25 vertices, a respectful size at the time.

Several other algorithms using Lagrangean relaxation appear in the literature. Christofides et al. (11) also describe a lower bound based on k-degree

center trees, which are minimum spanning trees having degree $K \leq k \leq 2K$ at the depot, plus $2K - k$ least-cost edges. Lagrangean bounds based on K-trees (sets of $n + K - 1$ edges spanning V) having degree $2K$ at the depot were used by Fisher (22) and by Martinhon, Lucena, and Maculan (41), among others. Miller (42) presented an algorithm based on minimum b-matchings having degree $2K$ at the depot and 2 on the remaining vertices. Lagrangean bounds can be improved by dualizing capacity inequalities (22, 42) and also comb and multistar inequalities (41).

Another family of exact algorithms stems from the formulation of the CVRP as a set partitioning problem by Balinski and Quandt (5). A column covers a set of vertices S with total demand not exceeding Q and has the cost of a minimum route over $\{0\} \cup S$. Unfortunately, the formulation was not practical at the time because pricing over the exponential number of columns requires the solution of a hard problem. Agarwal, Marthur, and Salkin (2) proposed a column generation algorithm on a modified set partitioning problem where column costs are given by a linear function over the vertices that yields a lower bound on the actual route cost. Columns with the modified cost can be priced by solving easy knapsack problems. Hadjiconstantinou et al. (29) derive lower bounds from heuristic solutions to the dual of the set partitioning formulation. The dual solutions are obtained by the so-called additive approach, combining the q-route and k-shortest path relaxations. For further information and comparative results on the algorithms mentioned above, we refer the reader to the surveys by Toth and Vigo (58, 59).

Column generation started to be used in exact algorithms for the VRPTW. Desrosiers, Soumis and Desrochers (18) and Desrochers, Desrosiers and Solomon (17) showed that this technique could perform very well on tightly constrained instances (those with narrow time windows). As the CVRP can be regarded as the particular case of VRPTW where time windows are arbitrarily large, column generation was viewed as a non-promising approach for the problem. In fact, until the early 2000's, the best performing algorithms for the CVRP were branch-and-cut algorithms that separated quite complex families of cuts identified by polyhedral investigation. Naddef and Rinaldi (43) survey the most effective exact branch-and-cut methods proposed in the literature up to 2002. In spite of their sophistication, some instances from the literature with only 50 customers could not be solved to optimality in the work of Lysgaard et al. (38), which is considered the most efficient branch-and-cut algorithm for the CVRP. At that moment, the Branch-Cut-and-Price algorithm by Fukasawa et al. (24) showed that the combination of cut and column generation could be much more effective than each of those techniques taken alone.

According to the classification proposed in (51), the BCP algorithm in (24) only uses *robust cuts*. A cut is said to be robust when the value of the dual variable associated to it can be translated into costs in the pricing subproblem. Therefore, the structure and the size of that subproblem remain unaltered, regardless of the number of robust cuts added. On the other hand, *non-robust cuts* are those that change the structure and/or the size of the pricing subproblem, each additional cut makes it harder. Robustness is a desirable property of a BCP. Even with good pricing heuristics, at least one call to the exact pricing is necessary to establish a valid dual bound. With the addition of non-robust cuts, there is a risk of having to solve to optimality an intractable subproblem. Nevertheless, since Jepsen et al. (33) and Baldacci et al. (3) it is known that some non-robust cuts[1] can be effectively used, at least if their separation is restricted in order to avoid an excessive impact on the pricing.

We briefly review the most recent works proposing exact algorithms for the CVRP, all of them are based on the combination of column and cut generation.

1. **Fukasawa et al. (24)** presented a BCP algorithm where the column are associated to the q-routes without k-cycles (32). The separated cuts are the same used in previous algorithms over the edge CVRP formulation. Those cuts are robust with respect to q-route pricing. If the column generation at the root node is found to be too slow, the algorithm may automatically switch to a branch-and-cut. This may be advantageous in a few instances. All benchmark instances from the literature with up to 134 customers could be solved to optimality, an expressive improvement over previous methods.

2. **Baldacci, Christofides and Mingozzi (3)** presented an algorithm based on column and cut generation. The columns are associated to elementary routes. Besides cuts for the edge formulation, Strengthened Capacity Cuts and Clique Cuts are separated. Those later cuts are effective but non-robust. An important new idea is introduced. Instead of branching, the algorithm finishes in the root node (therefore, it is not a BCP) by enumerating all elementary routes with reduced cost smaller than the duality gap. A set-partitioning problem containing all those routes is then given to a MIP solver. The algorithm could solve almost all instances solved in (24), usually taking much less time. However, the

[1]While the adjective *non-robust* focus on their negative aspect; some authors call them *strong cuts*, focusing on their positive aspect, a greater potential for significantly reducing the integrality gaps. A more neutral nomenclature is *master cuts*, since they are defined over the variables of the Master LP obtained by the Dantzig-Wolfe decomposition.

exponential nature of some algorithmic elements, in particular the route enumeration, made it fail on some instances with many customers per vehicle.

3. **Pessoa et al. (47)** presented some improvements over (24). Cuts from an extended formulation with load indices were also separated. Those cuts do not change the complexity of the pricing of q-routes by dynamic programming. Additionally, the idea of performing elementary route enumeration and MIP solving to finish a node was borrowed from (3). However, in order to avoid a premature failure when the root gap is too large, it was hybridized with traditional branching. The overall improvement was not sufficient for solving larger instances.

4. The algorithm by **Baldacci, Mingozzi and Roberti (4)** improved upon (3). It introduces the concept of ng-routes, a route relaxation that is more effective than the q-routes without k-cycles. Instead of Clique Cuts, Subset Row Cuts (33) and Weak Subset Row Cuts, that have less impact on the pricing, are separated. The resulting algorithm is not only faster on average, it is much more stable than the algorithm in (3), being able to solve even some instances with many customers per vehicle.

5. The recent work by **Contardo (12)** introduced new twists on the use of non-robust cuts and on route enumeration. The columns are associated with q-routes without 2-cycles, a relatively poor relaxation. The partial elementarity of the routes is enforced by non-robust Strong Degree Cuts. Robust cuts from edge formulation and non-robust Strengthened Capacity Cuts and Subset Row Cuts are also separated. The enumeration of elementary routes is directed to a pool of columns. As soon as the duality gap is sufficiently small to produce a pool with reasonable size (a few million routes), the pricing starts to be performed by inspection. From this point, an aggressive separation of non-robust cuts can be performed, leading to very small gaps. The reported computational results are very consistent. In particular, the hard instance M-n151-k12 (150 customers, 12 routes) was solved to optimality for the first time (in 6 hours), setting a new record.

6. Finally, the recent work by **Røpke (56)** went back to robust BCP. Its linear relaxation differs from (24) only by the use of the more effective ng-routes. The power of the algorithm comes mainly from a sophisticated and aggressive strong branching, able to reduce significantly the average size of the enumeration trees. The overall results are comparable with

the results in (12, 4). A long run of that algorithm (5.5 days) could also solve M-n151-k12.

1.2 Contributions of this Thesis

The central problem addressed in this thesis is the Capacitated Vehicle Routing Problem. However, contributions for two other important problems are also presented, the Shortest Path Problem with Resource Constraints (SPPRC) and the Elementary Shortest Path Problem with Resource Constraints (ESPPRC). These problems arise as the pricing subproblem to be solved when VRPs are tackled by column generation and branch-and-price.

1.2.1 Contributions to the SPPRC and ESPPRC

The algorithms for both the SPPRC and ESPPRC are presented in Chapter 5. The main contributions of this chapter are listed below.

- Describes a clever combination of Decremental State-Space Relaxation (DSSR) (54), *ng*-routes (3), bidirectional search (53) and completion bounds to compute solutions to the SPPRC.
- Extends this algorithm in order to solve the more difficult ESPPRC.
- Solves the ESPPRC for CVRP instances with up to 199 customers, a result that doubles the size of the ESPPRC instances solved to date.

A significant part of the result of this chapter was obtained in the joint work with Rafael Martinelli published in Martinelli et al. (40) and previously partly reported in (39). In particular, this previous work does not contain the bidirectional search and some experiments involving the ESPPRC.

1.2.2 Contributions to the CVRP

The Branch-Cut-and-Price algorithm proposed for the CVRP are described in Chapters 6, 7 and 8. It is a quite complex and sophisticated BCP that incorporates, often in an enhanced way, elements from all previous algorithms described in Section 1.1.

- The most important original contribution of this part of the thesis is the introduction of the limited memory Subset Row Cuts (lm-SRCs). They are a weakening of the SRCs that can be controlled and dynamically adjusted, making the lm-SRCs as effective in improving the lower bounds as SRCs, but still much less costly in the pricing.

– The underlying formulation used in the BCP has extended arc-load variables. This allows a particularly effective fixing of variables by reduced costs (superior to the fixing in (31)), with direct benefits on the pricing.

Other elements to be remarked in the proposed BCP are: (i) the columns in the BCP are associated to *ng*-routes (4). The pricing subproblem is solved by a bidirectional dynamic programming algorithm that also uses completion bounds to eliminate labels; (ii) like in (47), the BCP hybridizes branching with the route enumeration technique introduced in (3). Actually, inspired by (56), it performs an aggressive strong branching; (iii) as soon as the gap of a BCP node is sufficiently small, the elementary routes that can be part of the optimal solution can be enumerated into a large pool, as suggested in (12). From that point, since the pricing will be performed by inspection, all lm-SRCs may be immediately lifted to SRCs; and (iv) the lm-SRCs are still non-robust cuts. If the separation of some of them makes the algorithm substantially slower, the BCP performs a rollback, the offending cuts are removed even if it decreases the lower bound of the node.

Overall, we believe that this is one of the most sophisticated BCP algorithms ever implemented. All the instances used for benchmarking exact algorithms, with up to 199 customers, were solved to optimality. Moreover, some larger instances with up to 360 customers, only considered before by heuristic methods, were solved too.

The techniques introduced in this work, including the lm-SRCs, can be possibly applied on many other problems where BCP is currently applied, including several VRP variants, parallel machine scheduling or network design.

An extended abstract of this work was published in Pecin et al. (45) and a full paper was submitted to Operations Research.

There is also another working paper, motivated by this recent research, proposing a new set of benchmark instances able to push the limits of the state-of-the-art algorithms for the CVRP.

1.3 Thesis Outline

This work is organized as follows.

– Chapter 2 briefly reviews the Dantzig-Wolfe decomposition and the column generation technique. It also shows how cuts expressed over the original formulation (robust cuts) can be translated and added to the reformulation.

- Chapter 3 presents the CVRP formulations used by the algorithms proposed in this thesis.
- Chapter 4 proposes a column and cut generation algorithm to compute lower bounds to the CVRP. The columns are associated to ng-routes and only robust cuts (Rounded Capacity Cuts and Strengthened Combs) are separated.
- Chapter 5 describes the pricing algorithms of the column and cut generation algorithm for the CVRP.
- Chapter 6 proposes a branch-cut-and-price algorithm for the CVRP.
- Chapter 7 presents the labeling algorithms for pricing, variable fixing and route enumeration of the proposed BCP algorithm.
- Chapter 8 reports detailed computational results of the BCP.
- Chapter 9 contains the conclusions of this work.

2
IP Reformulation

Many mixed integer programming problems have a special structure that makes them suitable for the application of the Dantzig-Wolfe decomposition (also refered to as Dantzig-Wolfe reformulation), which was originally proposed to be applied to linear programs (15). This decomposition aims at explicitly considering a subset of the original problem constraints. When this decomposition was proposed the main objective was to reduce as much as possible the number of constraints present in the linear program to be solved, allowing it to fit in the memory of the computers available at the time.

The constraints removed of the mixed integer problem, usually those with a well-defined structure, are treated by separated subproblems. The solutions of these subproblems are then combined into a coordinating problem. The reasoning of the method is that it may be easier to solve a large number of smaller size (typically well-structured) subproblems than to solve the original problem, whose size and complexity are beyond what can be solved within a reasonable amount of time. The drawback is that the coordinating problem may potentially have to consider an exponential number of variables. However, the decomposition relies on column generation technique for improving the tractability of large-scale linear programs.

This chapter briefly review the Dantzig-Wolfe approach: the decomposition of a mixed integer program, its reformulation and the column generation procedure. We also show how cuts expressed over the variables of the original formulation can be translated to be added to the reformulation, thus defining what Poggi de Aragão and Uchoa called by robust cuts (51).

To simplify the presentation, we assume a pure integer program (IP) whose variables are bounded. The extension to the unbounded case is presented in (60), while the extension to the mixed integer case is presented in (61).

2.1
The Dantzig-Wolfe Decomposition

Consider the following integer program IP with n variables:

$$\begin{array}{lllll} Z_{IP} = min & cx & & & \\ s.t. & Ax & = & b & \\ & Dx & \leq & d & \\ & x & \in & \mathbb{Z}^n_+. & \end{array} \tag{IP}$$

Since we are supposing that $\mathcal{Q} = \{x \in \mathbb{Z}^n_+ \mid Dx \leq d\}$ is a finite set, let p be the number of elements in $\mathcal{Q}$, that is, $\mathcal{Q} = \{x^1, \ldots, x^p\}$. Let Q be a $n \times p$ matrix where each column correspond to an element of $\mathcal{Q}$. For each $x \in \mathcal{Q}$, there exist an unique vector λ satisfying the following relation:

$$\begin{array}{lll} x = & Q\lambda & \\ s.t. & \mathbf{1}\lambda = \mathbf{1} & \\ & \lambda \in \{0,1\}^p. & \end{array} \tag{$\mathcal{X}$}$$

The traditional reformulation of integer programming problems, proposed by Gilmore and Gomory (25), (26), consists in replacing the x variables of IP by its equivalent expression given by $\mathcal{X}$. Relaxing the integrality constraints, we get the Dantzig-Wolfe Master LP (DWM):

$$\begin{array}{lllll} Z_{DWM} = min & (cQ)\lambda & & & \\ s.t. & (AQ)\lambda & = & b & \\ & \mathbf{1}\lambda & = & \mathbf{1} & \\ & \lambda & \geq & 0. & \end{array} \tag{DWM}$$

Let Z_{LP} be the value of an optimal solution of the linear relaxation of IP. The value Z_{LP} gives a lower bound on the value of Z_{IP}. An interesting property of the Dantzig-Wolfe decomposition is that Z_{DWM} is always $Z_{LP} \leq Z_{DWM} \leq Z_{IP}$, since solve the DWM is equivalent to solve the following problem:

$$\begin{array}{lllll} Z_{DWM} = min & cx & & & \\ s.t. & Ax & = & b & \\ & x & \in & Conv(Dx \leq d; x \in \mathbb{Z}^n_+), & \end{array} \tag{2.1}$$

where $Conv(S)$ denotes the convex hull of set S. The drawback of this approach is that in general p (the number of λ variables) is very large. On the other hand, it is common that the lower bound Z_{DWM} be considerably better than Z_{LP}. This is possibly the main motivation for the use of that decomposition.

Block diagonal case

If the integer program has a constraint matrix of the form

$$\begin{pmatrix} A_1 & A_2 & \cdots & A_k \\ D_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & D_2 & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & D_k \end{pmatrix},$$

then k disjoint sets of constraints are identified: $Dx \leq d$ partitions into $D_i x_i \leq d_i$, $i = 1, \ldots, k$, where $x = (x_1, x_2, \ldots, x_k)$, with x_i being an n_i-vector for $i = 1, \ldots, k$. In this case, the problem can be written as follows:

$$\begin{aligned} Z_{IP'} = min \quad & \sum_{i=1}^{k} (c_i x_i) & & & \\ s.t. \quad & \sum_{i=1}^{k} (A_i x_i) & = & b & \text{(IP')} \\ & D_i x_i & \leq & d_i \quad \forall i = 1, \ldots, k & \\ & x_i & \in & \mathbb{Z}_+^{n_i} \quad \forall i = 1, \ldots, k. & \end{aligned}$$

For each $i = 1, \ldots, k$, let $\mathcal{Q}_i$ be the set (finite by our hypothesis) of elements defined by the subset of constraints $D_i x_i \leq d_i$ and let $\mathcal{X}_i$ be the $\mathcal{X}$ relation for the variables x_i and λ_i. Replacing each x_i by its equivalent expression ($x_i = Q_i \lambda_i$) and relaxing the integrality constraints of each λ_i, we get the following linear program:

$$\begin{aligned} Z_{DWM'} = min \quad & \sum_{i=1}^{k} (c_i Q_i) \lambda_i & & & \\ s.t. \quad & \sum_{i=1}^{k} (A_i Q_i) \lambda_i & = & b & \text{(DWM')} \\ & \mathbf{1} \lambda_i & = & \mathbf{1} \quad \forall i = 1, \ldots, k & \\ & \lambda_i & \geq & 0 \quad \forall i = 1, \ldots, k. & \end{aligned}$$

2.2 Column Generation

Column generation is a technique commonly used to solve a linear program with a huge number of variables (columns), as those obtained with the Dantzig-Wolfe decomposition. Let consider the solution of DWM. If the number p is too large, it may not be practical to solve DWM by explicitly considering all of its variables at once. Since most of the variables will be non-

basic and therefore assume a value of zero in the optimal solution, the technique leverages this idea to solve DWM by considering only implicitly the total set of its variables. The DWM is split into two problems, the master problem and the subproblem. Initially, just a small number of the λ variables are considered, generating the so-called restricted master linear program (Restricted DWM). Next, the algorithm tries to find new columns with negative reduced cost, suitable to improve the current solution of the restricted master. This is done by solving the subproblem, which incorporates the dual variables of the Restricted DWM as part of the objective function. If such columns are found, the restricted master is updated and then re-solved. This iterative process continues until no improving columns is found. In many cases, this allows large linear programs that had been previously considered intractable to be solved.

A more formal description of the technique follows. The procedure starts by considering a Q' matrix containing a subset of the columns of Q. The following linear program is called Restricted Dantzig-Wolfe Master LP (RDWM):

$$\begin{array}{lllll} Z_{RDWM} = min & (cQ')\lambda' & & & \\ s.t. & (AQ')\lambda' & = & b & (1) \\ & \mathbf{1}\lambda' & = & \mathbf{1} & (2) \\ & \lambda' & \geq & 0 & \end{array} \qquad \text{(RDWM)}$$

Consider that there is a feasible solution for RDWM with the columns in Q'. If necessary, to obtain feasibility one can use artificial columns not present in Q with high cost in the objective function. Given an optimal solution of RDWM, let μ and ν be the vector of dual multipliers associated with constraints (1) and (2), respectively. The following subproblem, refered to as Pricing Subproblem (PS), must be solved to find variables with negative reduced cost with respected to μ and ν:

$$\begin{array}{lllll} v(\mu,\nu) = min & (c-\mu A)x - \nu & & & \\ s.t. & Dx & \leq & d & \\ & x & \in & \mathbb{Z}^n_+ & \end{array} \qquad \text{(PS)}$$

The column generation algorithm solves DWM by alternating the solution of RDWM and the solution of PS. If the value of the optimal solution of the subproblem is negative (that is, if $v(\mu,\nu) < 0$) then x^*, the optimal solution found for PS, defines a new column $q' = (Ax^*)$ (with cost cx^*), which is added to matrix Q' of RDWM. In this scheme, the solution of RDWM gives duals for PS and the solution of PS gives columns for RDWM. This process is repeated until $v(\mu,\nu) = 0$. When this happens, the current optimal solution

of RDWM is also an optimal solution of DWM.

Note that the solution of DWM by column generation may require the solution of the pricing subproblem several times, therefore this scheme is only practical when constraints $Dx \leq d$ have a "nice structure", which allows the pricing supbroblem to be solved within a reasonable amount of time. If the subproblem is known to be a strongly $\mathcal{NP}$-Hard, the column generation algorithm can fail, since there is a risk of having to solve to optimality an intractable subproblem.

Consider the case in which the constraint matrix has a block diagonal structure and the decomposition was conducted in order to take advantage of this structure, thus obtaining the DWM' problem shown above. Let Q'_i be a matrix containing a subset of the columns of the matrix Q_i, for each $i = 1, \ldots, k$. The restricted master problem defined by such matrices is as follows:

$$
\begin{array}{lllllll}
Z_{RDWM'} = min & \displaystyle\sum_{i=1}^{k}(c_i Q'_i)\lambda'_i & & & & & \\
s.t. & \displaystyle\sum_{i=1}^{k}(A_i Q'_i)\lambda'_i & = & b & & (1) & \text{(RDWM')} \\
& 1\lambda'_i & = & 1 & \forall i = 1, \ldots, k & (2) & \\
& \lambda'_i & \in & 0 & \forall i = 1, \ldots, k. & &
\end{array}
$$

Let μ_i and ν_i be the vector of dual multipliers associated with constraints (1) and (2), respectively. The pricing consists in solving k independent subproblems, each one defined as follows:

$$
\begin{array}{lllll}
v(\mu_i, \nu_i) = min & (c_i - \mu A)x_i - \nu_i & & & \\
s.t. & D_i x_i & \leq & d_i & \text{(PS')} \\
& x_i & \in & \mathbb{Z}^n_+ &
\end{array}
$$

2.3 Adding Cuts

Consider a fractional solution $\bar{\lambda}$ of DWM, which in the original formulation IP corresponds to the solution $\bar{x} = Q\bar{\lambda}$. A violated cut $a^i x \leq b_i$ (such that $a^i \bar{x} > b_i$) can be added to DWM to cut the solution $\bar{\lambda}$, just as if $a^i x \leq b_i$ belonged to the original subset of constraints $Ax = b$ in IP:

$$
\begin{array}{lllll}
Z_{DWM''} = min & (cQ)\lambda & & & \\
s.t. & (AQ)\lambda & = & b & \\
 & (a^iQ)\lambda & \leq & b_i & \text{(DWM'')} \\
 & \mathbf{1}\lambda & = & \mathbf{1} & \\
 & \lambda & \geq & 0. &
\end{array}
$$

The addition of the row a^i to the matrix A and the dual variable μ_i to the vector μ do not change the structure of the pricing subproblem. Of course, these two elements must now be considered in the calculation of $(c - \mu A)x$. According to the classification proposed by (51), this cut is said to be *robust* because the value of the dual variable associated to it can be translated into costs in the pricing subproblem.

On the other hand, a violated cut $\pi^i\lambda \leq b_i$ (such that $\pi^i\bar{\lambda} > b_i$) usually introduces coefficients in the DWM that can not be calculated by a linear transformation of the subproblem solutions. Therefore, the effect of the new dual variable in the reduced costs can only be taken into account in the pricing subproblem by changing its structure. These changes have the potential of making it considerably more difficult to solve. Such cuts will be referred as *non-robust* cuts, according to (51).

3
Mathematical Formulations

This chapter revisits the CVRP formulations that may be considered as starting points in the design of the algorithms presented in this thesis. To explain the formulations, the following notation is used. Given a vertex set $S \subseteq V$, let $\delta(S)$ denote the set of edges $e \in E$ which have only one endpoint in S. As usual, when a single vertex $i \in V$ is considered, we write $\delta(i)$ rather than $\delta(\{i\})$. For any set $S \subseteq V$, let $d(S) = \sum_{i \in S} d(i)$, and $k(S) = \lceil d(S)/Q \rceil$. Edges are also indicated through a single index $e = (i, j)$.

3.1
The Two-Index Formulation

The classical Two-Index Formulation (TIF), a.k.a. Edge Formulation, proposed by Laporte and Nobert (34), uses a variable x_e for each edge $e \in E$, indicating the number of times the edge is traversed by any vehicle. If edge e is not adjacent to the depot, then there are only two possible values for x_e: 0, which means that the edge is not used or 1, otherwise. If the edge is adjacent, then the value 2 may be assigned, thus allowing routes with a single customer. The formulation is defined as:

$$\text{(TIF)} \quad \min \quad \sum_{e \in E} c_e x_e \tag{3.1}$$

subject to

$$\sum_{e \in \delta(i)} x_e = 2, \qquad \forall\, i \in V_+, \tag{3.2}$$

$$\sum_{e \in \delta(0)} x_e = 2K, \tag{3.3}$$

$$\sum_{e \in \delta(S)} x_e \geq 2k(S), \qquad \forall\, S \subseteq V_+, \tag{3.4}$$

$$x_e \in \{0, 1\}, \qquad \forall\, e \in E \setminus \delta(0), \tag{3.5}$$

$$x_e \in \{0, 1, 2\}, \qquad \forall\, e \in \delta(0). \tag{3.6}$$

Constraints (3.2) ensure that each customer is serviced by only one vehicle. Constraint (3.3) imposes that K vehicles must leave and enter in the depot. These constraints are also referred to as Degree Constraints. Constraints (3.4) states that any subset S that does not include the depot must have at least $k(S)$ vehicles leave and enter S. These constraints are usually referred as Rounded Capacity Cuts (RCCs) and ensure that a solution will not have any vehicle servicing more than its capacity Q and will not have any subtours, i.e., every route must include the depot. Although computing the minimum number of vehicles with capacity Q required to serve subset S corresponds to solve a NP-hard problem (the Bin-Packing Problem – BPP), it can be shown that the trivial lower bound $k(S)$ does not invalidate the TIF formulation, i.e. all integer solutions satisfy the vehicle capacity constraints. Finally, constrains 3.5 and 3.6 are the integrality constraints.

Note that there is a exponential number of constraints of type (3.4), which means that may be impractical to solve its linear relaxation considering all these constraints at once. A cutting plane algorithm may be used to solve this relaxation by generating these constraints dynamically (cuts) during the optimization process.

3.2 The Set Partitioning Formulation

The Set Partitioning Formulation (SPF) was originally proposed by Balinski and Quandt (5) and is a general form to formulate VRPs. A binary variable is assigned to each possible route. A route is a path that starts at the depot, traverses a sequence of customers with total demand at most Q, and returns to the depot (11). Let Ω be the set of routes and let λ_r be the binary variable indicating whether the route is used or not. Each route r has a cost c_r, given by the sum of the cost of the edges in its path. Given coefficients a_i^r indicating the number of times that route r visits customer i, the Set Partitioning Formulation follows:

$$\text{(SPF)} \quad \min \quad \sum_{r \in \Omega} c_r \lambda_r \tag{3.7}$$

subject to

$$\sum_{r \in \Omega} a_i^r \lambda_r = 1, \qquad \forall i \in V_+, \tag{3.8}$$

$$\sum_{r \in \Omega} \lambda_r = K, \tag{3.9}$$

$$\lambda_r \in \{0, 1\}, \qquad \forall r \in \Omega. \tag{3.10}$$

The objective function (3.7) minimizes the overall cost of the selected routes. Constraints (3.8) guarantee that each customer is serviced by exactly one route. Constraint (3.9) imposes the use of K vehicles. Finally, constraints (3.10) force the variables to be binary.

The SPF has some important characteristics. First, it is a general formulation and hence applies to a large class of VRPs because some specific route constraints can be incorporated into the definition of the set Ω (e.g., time windows). Second, in general the dual bounds obtained with its linear relaxation are tight, thus it has been widely used to formulate VRPs. Third, the routes defined in the set Ω need not to be elementary routes (or feasible routes), those visiting a customer at most once. We remark that, even allowing routes to visit a customer more than once, the SPF naturally discards such routes, since constraints (3.8) forbid $\lambda_r = 1$ for any route r with a coefficient $a_i^r > 1$. Therefore, this enlargement of Ω still leaves SPF as a formulation for VRPs. This last characteristic plays an important role in the design of algorithms based on the SPF, as explained later in this section.

However, as a variable is assigned to each possible route, there is a exponential number of such variables. This is the drawback of this formulation, as it requires the use of column generation techniques to deal with this huge number of columns (variables). The pricing subproblem consists of finding a column (interpreted as a route) with minimum reduced cost. Let π_i be the dual variable of the constraint in (3.8) corresponding to $i \in V_+$ and π_0 be the dual variable of constraint (3.9). We can define the reduced cost $\bar{c}_r$ of a route r as follows:

$$\bar{c}_r = c_r - \pi_0 - \sum_{i \in V_+} a_i^r \pi_i. \tag{3.11}$$

Therefore, the reduced cost of a route is its actual cost minus the dual variable π_0 and the dual variable π_i of each customer i it visits. This means that it is possible to compute the reduced cost of a route as the sum of the reduced cost of the edges in its path. In view of this, we can formulate the problem of finding the minimum reduced cost route as a shortest path problem with a capacity constraint (since feasible routes must satisfy the vehicle capacity) on a graph where the cost of an edge $e = (u, v) \in E$ is given by its reduced cost:

$$\bar{c}_e = c_e - (\pi_u + \pi_v)/2. \tag{3.12}$$

Balinski and Quandt defined Ω as the set of elementary routes. In this definition, coefficients a_r^i are always binary and the pricing subproblem requires

finding the minimum cost capacitated elementary route, a strongly NP-hard problem. A more tractable pricing problem could be obtained by changing the definition of the Ω. The set of routes can be redefined to include all walks leaving and returning to the depot such that the sum of the demands of the visits of the walk does not exceed the vehicle capacity. These walks could revisit the same customer, but each additional visit required a new accounting of its demand. Christofides et al. (11) coined such routes as q-routes and used them in their Lagrangean method.

The advantage of solving the relaxation of the SPF using Ω as the set of q-routes is that the resulting pricing subproblem is weakly NP-hard and a pseudo-polynomial dynamic programming algorithm for its resolution is available. However, if the pricing becomes easier, on the other side, the resulting lower bounds obtained are usually significantly worse than those obtained with Ω defined as the set of elementary routes.

Route relaxation

Balancing the complexity of the pricing with the resulting lower bounds has become crucial to the efficiency of the algorithms based on the SPF. One idea is to impose some controlled amount of partial elementarity on the routes. The goal is to obtain bounds as close as possible to the elementary route bound, while still keeping the associated pricing problem tractable. Regarding route relaxation, we highlight two alternatives:

- The more classical q-routes without k-cycles allows multiple visits to a customer i, on the condition that at least k other customers are visited between successive visits.
- The more recent ng-routes requires the definition of neighborhood sets $NG(i) \subseteq V_+$, for each customer $i \in V_+$. A set $NG(i)$ may stand for the $|NG(i)|$ (this cardinality is decided *a priori*) closest customers and includes i itself. An ng-route allows multiple visits to a customer i, on the condition that at least one customer j such that $i \notin NG(j)$ is visited between successive visits.

This last route relaxation aims at obtaining a better compromise between pricing efficiency and lower bound quality. It is extensively discussed in Chapter 5.

3.3 The Arc-Load indexed Formulation

This extended formulation for the Asymmetric CVRP (therefore valid for the CVRP) was presented in (47) (also in (27) for the unitary demand case). Now $G_D = (V, A)$ is a complete directed graph with arc costs c_a, $a \in A$. For any set $S \subseteq V$, $\delta^-(S) = \{(i,j) \in A : i \in V \setminus S, j \in S\}$, and $\delta^+(S) = \{(i,j) \in A : i \in S, j \in V \setminus S\}$. Let define binary variables x_a^q indicating that arc $a = (i,j)$ is traversed by some vehicle (from i to j) carrying a load — the sum of the demands of vertex i and of its preceding vertices in the route — of exact q units. For convenience, let $Q_i = \{d_i, \ldots, Q\}$. The Arc-Load indexed Formulation (ALF) is:

$$\text{(ALF) min} \qquad \sum_{a \in A} c_a \sum_{q=0}^{Q} x_a^q \tag{3.13}$$

subject to

$$\sum_{a \in \delta^-(i)} \sum_{q=1}^{Q} x_a^q = 1, \qquad \forall i \in V_+, \tag{3.14}$$

$$\sum_{i \in V_+} x_{0i}^0 = K, \tag{3.15}$$

$$\sum_{a \in \delta^-(i)} x_a^{q-d_i} - \sum_{a \in \delta^+(i)} x_a^q = 0, \qquad \forall i \in V_+, q \in Q_i, \tag{3.16}$$

$$x_a^q \in \{0,1\}, \qquad \forall a \in A, q \in Q_1, \tag{3.17}$$

$$x_{(i,0)}^q = 0, \qquad \forall\, i \in V_+, q \in Q_1. \tag{3.18}$$

Equations (3.14) and (3.15) are customer and depot degree constraints. Balance equations (3.16) state that if an arc with index $q - d_i$ enters vertex i then an arc with index q must leave i. This both prevents cycles and routes with total demand greater than Q. Variables with index distinct from 0 to the depot can be removed. Note that variables x_{ij}^q with $q < d_i$ can also be removed. To provide a more simple and precise notation of this formulation, we define a directed multigraph $G_Q = (V, A_Q)$, where A_Q contains arcs $(i,j)^q$, for all $i \in V_+, j \in V$ and for all $q = d_i, \ldots, Q$, plus arcs $(0,j)^0$, for all $j \in V_+$. When working with variables x_a^q it is assumed that $\delta^-(S)$ and $\delta^+(S)$ are the subsets of arcs in A_Q, with any index, entering and leaving S, i.e., for any set $S \subseteq V$, $\delta^-(S) = \{(i,j)^q \in A_Q : i \in V \setminus S, j \in S\}$, and $\delta^+(S)$ is defined in a

similar way. The ALF can be rewritten as:

$$(\text{ALF})\ \min \quad \sum_{a^q \in A_Q} c_a x_a^q \tag{3.19}$$

subject to

$$\sum_{a^q \in \delta^+(i)} x_a^q = 1, \qquad \forall i \in V_+, \tag{3.20}$$

$$\sum_{a^q \in \delta^+(0)} x_a^q = K, \tag{3.21}$$

$$\sum_{a^{q-d_i} \in \delta^-(i)} x_a^{q-d_i} - \sum_{a^q \in \delta^+(i)} x_a^q = 0, \quad \forall i \in V_+, q \in Q_i \tag{3.22}$$

$$x_a^q \in \{0,1\}, \qquad \forall a^q \in A_Q. \tag{3.23}$$

This formulation can be viewed as defining an acyclic network $\mathcal{N} = (V_Q, A_Q)$ with a set of nodes $V_Q = \{(i,q) : i \in V; q = d_i, \ldots, Q\}$. The set of arcs is also A_Q, but an arc $(i,j)^q \in A_Q$ is interpreted as going from (i,q) to $(j, q+d_i)$. Figure 3.1 gives an example of an integral solution depicted as a set of K paths in such a network, in that case $n = Q = 5$, $d_1 = d_3 = d_4 = 2$, and $d_2 = d_5 = 3$.

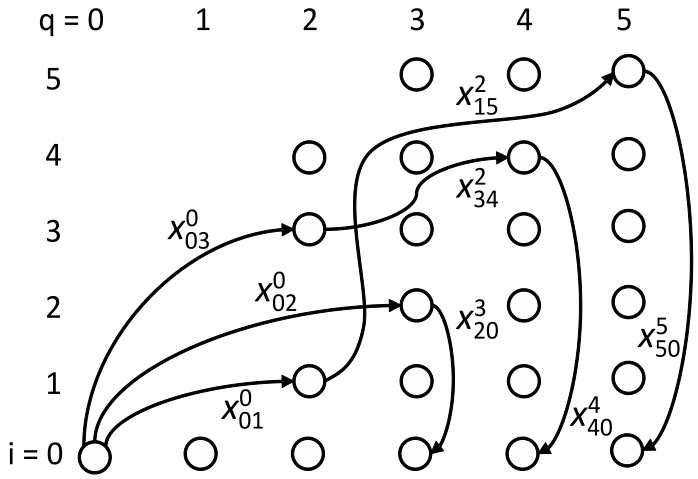

Figure 3.1: Representation of a solution as a set of paths in $\mathcal{N}$.

The linear relaxation of this formulation yields a weak bound, the same of the bound obtained with the SPF with Ω defined as the set of q-routes without any cycle elimination. Therefore, using the Arc-Load indexed Formulation directly in a branch-and-bound algorithm is not interesting. However, this formulation may be useful in a branch-and-cut approach for the CVRP. Of course, since $x_{ij} = \sum_{(i,j)^q \in A_Q} x_{ij}^q + \sum_{(j,i)^q \in A_Q} x_{ji}^q$, any inequality valid for

the TIF could be used in ALF. But the potential advantage of ALF is to allow the derivation and separation of new families of cuts defined over this pseudo-polynomially large extended variable space and a particularly effective variable fixing. Anyway, working directly with this formulation is only practical for small values of capacity, as there are $O(|A|Q)$ variables and $O(|V|Q)$ constraints.

4
A Column and Cut Generation Algorithm

In the work of Fukasawa et al. (24), a branch-and-cut approach is combined with the q-routes approach (which are interpreted as column generation instead of the original Lagrangean relaxation of Christofides et al. (11)) to derive superior lower bounds than those obtained with pure column or cut generation. Following their ideas, we combine cut generation with the more promising ng-routes. This chapter presents the formulation used to build our basic algorithm to compute lower bounds for the CVRP. As in (24), we optimize simultaneously over both the Two-Index Formulation (TIF) and the Set Partitioning Formulation (SPF). Since the resulting formulation has an exponential number of both columns and rows, this leads to a column and cut generation algorithm.

4.1
Formulation

Let us consider the set Ω (see Section 3.2) as the set of all q-routes. For each $r \in \Omega$ define a binary variable λ_r and coefficients a_r^e, for each edge $e \in E$, indicating how many times e is traversed with load q in route r. The new formulation for the CVRP follows:

$$\min \quad \sum_{e \in E} c_e x_e \tag{4.1}$$

subject to

$$\sum_{r \in \Omega} a_r^e \lambda_r = x_e, \qquad \forall\, e \in E, \tag{4.2}$$

$$\sum_{e \in \delta(i)} x_e = 2, \qquad \forall\, i \in V_+, \tag{4.3}$$

$$\sum_{e \in \delta(0)} x_e = 2K, \tag{4.4}$$

$$\lambda_r \in \{0,1\}, \qquad \forall\, r \in \Omega, \tag{4.5}$$

$$x_e \in \{0,1\}, \qquad \forall\, e \in E \setminus \delta(0), \tag{4.6}$$

$$x_e \in \{0,1,2\}, \qquad \forall\, e \in \delta(0). \tag{4.7}$$

Constraints 4.2 define the equivalence between variables λ_r and x_e. Since the set Ω contains q-routes (or any feasible route), the vehicle capacity constraints are immediately satisfied.

This formulation can be rewritten by replacing every occurrence of x_e by its equivalent given by 4.2. The resulting formulation is indeed the Set Partitioning Formulation as shown below, but here presented in a different format, with edge costs c_e and coefficients a_e^r. This new format is more convenient to see how a generic cut expressed over the edge variables x_e can be included in the SPF and also how the reduced cost of a q-route (column) can be expressed as the sum of the reduced costs of its edges. We will refer to its linear relaxation as Dantzig-Wolfe Master (DWM):

$$\text{(DWM)} \quad \min \quad \sum_{r\in\Omega}\left(\sum_{e\in E} c_e a_r^e\right)\lambda_r \tag{4.8}$$

subject to

$$\sum_{r\in\Omega}\left(\sum_{e\in\delta(i)} a_e^r\right)\lambda_r = 2, \quad \forall i \in V_+, \tag{4.9}$$

$$\sum_{r\in\Omega}\left(\sum_{e\in\delta(0)} a_e^r\right)\lambda_r = 2K, \tag{4.10}$$

$$\lambda_r \geq 0, \qquad \forall r \in \Omega. \tag{4.11}$$

It is easy to see that DWM is in fact the linear relaxation of SPF. First, the total cost of a route r is $c_r = \sum_{e\in E} c_e a_r^e$, $\forall r \in \Omega$. Second, since a_i^r denotes the number of times route r traverses customer $i \in V_+$, we can state that $\sum_{e\in\delta(i)} a_e^r = 2a_i^r$, $i \in V_+$. Finally, we can state that $\sum_{e\in\delta(0)} a_e^r = 2$ because every route must start and end at the depot.

The relationship between variables x_e and λ_r given by equation 4.2 is interesting because allows us to improve the DWM by including cuts expressed in terms of x_e variables. This means that any cut valid for the TIF can be separated, translated and added as a valid cut for the SPF. For example, the Rounded Capacity Cuts can be rewritten as follows:

$$\sum_{r\in\Omega}\sum_{e\in\delta(S)} a_e^r\lambda_r \geq 2k(S) \qquad \forall S \subseteq V_+. \tag{4.12}$$

A generic cut of format $\sum_{e\in E}\alpha_e x_e \geq b$ can be included in the DWM as $\sum_{r\in\Omega}(\sum_{e\in E}\alpha_e a_e^r)\lambda_r \geq b$. This additional cut contributes with the value $-\alpha_e\pi$ to the computation of the reduced cost $\bar{c}_e$ of an edge $e \in E$, where π is the

corresponding dual variable. Therefore, the structure of the pricing problem is not changed, meaning that such cuts are robust.

4.2 Column Generation

The reduced cost of a column (λ variable) is the sum of the reduced costs of the edges in the corresponding q-route. Suppose that, at a given instant, there are n_R constraints over the x_e variables in the DWM, including equalities (4.10) and (4.9). Constraint (4.10) has the dual variable π_0, the constraint in (4.9) corresponding to $i \in V_+$ has the dual variable π_i, and each additional constraint l, $n < l < n_R$, of format $\sum_{r\in\Omega} (\sum_{e\in E} \alpha_e^l a_e^r)\lambda_r \geq b^l$, has the dual variable π_l. The reduced cost of an edge $e \in E$ is defined as:

$$\bar{c}_e = c_e - \sum_{l=0}^{n_R-1} \alpha_e^l \pi_l. \tag{4.13}$$

The pricing subproblem for solving the DWM consists in finding a shortest path in G from node 0 (the depot) to node 0 satisfying the capacity constraint (i.e., the shortest q-route), with respect to the current edge reduced costs $\bar{c}_e$. This problem is a particular case of the well known Shortest Path Problem with Resource Constraints (SPPRC) and can be solved in $O(n^2Q)$ time, as shown in Chapter 5. As mentioned before, a significantly stronger linear relaxation for DWM would be obtained if Ω was redefined as the set of elementary routes, thus resulting in solving the Elementary Shortest Path Problem with Resource Constraints (ESPPRC). On the other hand, the pricing subproblem would become strongly NP-hard. A possible alternative is pricing q-routes without s-cycles (32). While pricing q-routes without 3-cycles is not much more costly than pricing ordinary q-routes (24), using values of s larger than 4 can make the algorithm too slow.

In view of this, our pricing algorithms are based on the ng-route relaxation (4), a more recent alternative for imposing partial elementarity. From now on, Ω is redefined to be a set of ng-routes. Since the definition of the ng-routes can be naturally extended to generate only elementary routes (simply setting $NG(i) = V_+, \forall i \in V_+$), such routes are also considered. The main contribution of our pricing algorithms (in this part of the thesis) is a clever combination of Decremental State-Space Relaxation (DSSR) (54), ng-routes (3), bidirectional search (53) and completion bounds, as described in detail in Chapter 5.

4.3 Cut Generation

Besides the RCCs 3.4, there are several known families of valid CVRP cuts over the edge variables. The package CVRPSEP (37) contains effective heuristic separation procedures for the following families of cuts: rounded capacity, framed capacities, strengthened combs, multistars, and extended hypotours. While all those families may play a significant role in branch-and-cut algorithms over the TIF (like (38)), only RCCs and (by a much smaller degree) strengthened combs cuts can strengthen the SPF in a significant way (24). It seems that most cuts in the other families are already implicitly given by constraints (3.8)-(3.9). Indeed, Letchford and Salazar (35) proved that all generalized large multistar cuts are implied by those constraints even if the definition of Ω includes all q-routes.

Therefore, we included only the rounded capacity and strengthened combs cuts, using the separation procedures available in the CVRPSEP package. Initially, the DWM includes only degree constraints 4.9 and 4.10, then those cuts are being added during the algorithm. We convert a solution $\bar{\lambda}$ from DWM into a corresponding $\bar{x}$, using Equation 4.2. If this solution is fractional, it is given as input to the CVRPSEP package (37) The violated cuts found are translated back into λ variables to be introduced in the DWM.

4.4 A Column and Cut Generation Algorithm

The algorithm starts by adding columns using the heuristic pricing described in Section 5.4.6. When the heuristic is not able to find more columns with negative reduced cost, it looks for all violated cuts. We have found that this strategy works better than call the cut separation procedure after the convergence of the exact pricing (both heuristic pricing and cut separation perform quickly). The violated cuts found are added to the DWM and the heuristic pricing restarts. These passes are repeated until both heuristic and separation fail. At this point, the exact pricing (Section 5.4) is called. If negative reduced cost are found, the whole processes restarts. This procedure is repeated until both column generation and cut separation fail.

5
Pricing without Non-Robust Cuts

Column generation has become a widely applied technique for exactly solving different routing problems. Currently, it is involved in almost all of the current most efficient approaches to routing problems. These approaches use the Set Partitioning Formulation, defined in Section 3.2. The resolution of its linear relaxation requires the use of column generation techniques. In order to obtain the best possible lower bounds, ideally the set Ω should only contain elementary routes. In that case, the pricing subproblem to be solved can be modeled as the Elementary Shortest Path Problem with Resource Constraints (ESPPRC).

The ESPPRC is a shortest path problem on a graph where the customers have an amount of resources that are consumed during a visit. The resource constraints require that the total of the resources consumed by any feasible solution does not exceed the existing limits. There may be edges with negative cost, and these edges may generate negative cycles, but because a feasible solution must be an elementary path, revisiting a customer is strictly forbidden.

The ESPPRC is a difficult to solve $\mathcal{NP}$-Hard problem (see Dror (20)). In general, the current best performing algorithms have acceptable processing times when the optimal solution is a path with at most fifteen customers. We refer to Pugliese and Guerriero (52) for a review of the approaches proposed throughout the last three decades. The following publications are central: Feillet et al. (21), Chabrier (8), Righini and Salani (53, 54), and Boland (7).

Instead of solving the ESPPRC, the original work of Christofides et al. (11) solves its relaxation, the Shortest Path Problem with Resource Constraints (SPPRC). This is possible because the SPF remains a valid formulation even if the pricing subproblem is redefined to be the SPPRC. This relaxation does allow revisiting a same customer in a route. The resulting non-elementary routes are often called q-routes. However, the resource constraints are the same as the ESPPRC, and therefore, every time a customer is visited, the relevant resource consumption is counted, and the total consumption must still respect the existing limits. A recent survey about the SPPRC can be found in (19). This relaxation has some interesting properties. First, as distinct

from the original problem, it can be solved in pseudo-polynomial time using a dynamic programming algorithm. In addition, even relaxing the elementarity constraint, the SPF bounds found by its linear relaxation are still reasonable, especially when there are few customers per route. Furthermore, to strengthen the bounds of the linear relaxation, (11) also demonstrated that size two cycles can be forbidden with almost no extra effort. In search of even better bounds, researchers tried other cycle elimination devices in order to make non-elementary routes closer to elementary routes, without dealing with the whole complexity of the ESPPRC.

The work of Irnich and Villeneuve (32) devised an algorithm that solves the SPPRC by forbidding cycles up to a given size. This algorithm is significantly more complicated, resulting in a complexity that grows factorially with the size of the cycles being forbidden. On account of this, their method quickly becomes impractical. Eliminating cycles of size four or more is already too time consuming compared with the bound improvement obtained. Such behavior was verified in practice by Fukasawa et al. (24) for the CVRP.

Recently, Baldacci et al. (4) proposed another compromise between elementary routes and q-routes: ng-routes. These ng-routes are restricted non-elementary routes built accordingly to customer sets, ng-sets, which are associated with each customer and limit their "memory". When a path arrives at given customer, it "forgets" previous visits to customers that do not belong to the its ng-set. Further visits to those customers are them allowed. As the set sizes increase, the problem becomes harder to solve. This is due to the fact that the ng-routes generated are going to be increasingly closer to elementary routes. Although the SPPRC with ng-routes can be solved in pseudo-polynomial time for a fixed ng-set size, to the best of our knowledge, there is no work that solves this problem for ng-sets larger than 26 (55).

The work described in this chapter aims at efficiently solving both the SPPRC with restricted non-elementary routes and the ESPPRC. The first improvement is obtained by adapting the Decremental State-Space Relaxation (DSSR) technique of Righini and Salani (54) to the SPPRC with ng-routes. This technique was initially proposed for the ESPPRC, where the elementarity restriction is relaxed and the problem is then solved iteratively, rebuilding the restrictions as needed, until the optimal solution is found. The main difference of our algorithm is to relax the restriction imposed by the ng-sets instead of relaxing the elementarity of the routes. A quite similar approach was developed independently by Roberti and Mingozzi (55), their algorithm was applied on the Delivery Man Problem.

Our algorithm is accelerated by using completion bounds. Because an

iteration of the DSSR is a relaxation for its next iteration, the completion bounds estimate lower bounds for completing the paths being built on a given iteration. Also, given an upper bound for the optimal solution (which is usually equal to zero for pricing algorithms), it avoids the extension of paths that may exceed the known upper bound on the next DSSR iteration. Therefore the DSSR algorithm is also used to systematically compute improved completion bounds.

These two techniques were already used together in the work of Pecin (46). In that work, a column generation procedure that uses the ESPPRC as the pricing subproblem was proposed for the CVRP. Using the algorithm of (24), instances with up to 100 customers could be solved to optimality pricing only elementary routes.

Furthermore, we also present algorithms based on bidirectional dynamic programming, including the combination of this technique with DSSR. This combination was first used in (54) to solve the ESPPRC. We extend their work to the context of the *ng*-routes.

Finally, we demonstrate how our algorithms for the SPPRC with restricted non-elementary routes can be easily extended to generate only elementary routes. We also highlight the two new elements existing in our approach that allow us to double the size of the ESPPRC instances solved thus far. We compare our results for the ESPPRC with our implementation of the best algorithm proposed in (54).

The proposed algorithms are then applied to the CVRP. We report experiments demonstrating that our algorithms are able to solve the SPPRC with *ng*-set sizes up to 64 and the ESPPRC for hard instances obtained when solving the column generation. The results of the column generation algorithm also provide a clear idea of the gains in the lower bounds comparing the SPPRC with different *ng*-set sizes and also with the ESPPRC, as well as the time required for computing them. We show that our algorithms price elementary routes for instances up to 199 customers.

As mentioned in Section 1.2.1, part of the contents described in this chapter is a joint work with the doctoral thesis of Rafael Martinelli (39) and therefore has similarities with his text.

This chapter is organized as follows. Section 5.1 presents the (E)SPPRC and explains the required mathematical notation. The basic *q*-route relaxation is presented in Section 5.2. The *ng*-route relaxation is described in Section 5.3. Section 5.4 explains our labeling algorithms to price *ng*-routes. Section 5.5 shows how our algorithms can be used to obtain only elementary routes and also highlights the main elements that allowed us to build a very efficient

method for solving the ESPPRC. Section 5.6 reports the computational results for the CVRP. Finally, Section 5.7 presents some conclusions.

5.1 Shortest Path Problem with Resource Constraints

The pricing subproblem for solving the linear relaxation of the SPF (for general VRPs) by column generation has been modeled by many authors as a Shortest Path Problem with Resource Constraints (SPPRC). Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a directed graph with $\mathcal{V}$, a vertex set composed of a set of customers $\mathcal{C}$ plus a source vertex s and a destination vertex t and A, a complete arc set minus $\{(s,t),(t,s)\}$. Let $\mathcal{R}$ be a set of resources. For each arc $(i,j) \in A$, let $\bar{c}_{ij}$ (unrestricted in sign) be the cost of the arc and w_{ij}^r be the consumption of the edge, for each $r \in \mathcal{R}$. For each pair $i \in \mathcal{C}$ and $r \in \mathcal{R}$ let a_i^r and b_i^r be two non-negative values, such that the total resource consumption along a path from s to i must belong to the interval $[a_i^r, b_i^r]$. The SPPRC aims to find a minimum cost, non necessarily elementary, path from s to t that satisfies all resource constraints. When the shortest path must be elementary, the problem becomes the Elementary SPPRC (ESPPRC).

The resources constraints can model different types of restrictions. For instance, most vehicle routing problems consider that the vehicles have a known capacity, and this capacity cannot be exceeded in a single route. Other problems have time windows, which require the route to visit a customer in a given interval of time. Moreover, one can also view the elementarity constraint as resource constraints, where each customer defines a binary resource and when a route visits a customer, it consumes all of the associated resource.

In this work, we deal only with the capacity constraint, in addition to the obvious elementarity constraint. The customer set $\mathcal{C}$ corresponds to the set V_+, and the source s and the destination t are interpreted as a single vertex representing the depot, labeled 0 (therefore, the solution of the problem is a route instead of a path). The cost $\bar{c}_{ij}$ is equal to the current reduced cost of the corresponding edge (given by Equation 4.13). The pricing of q-routes corresponds to an SPPRC with a single resource 0 (the vehicle capacity). For each $(i,j) \in A$, $w_{ij}^0 = d_j$, and for each $i \in V_+$, $a_i^0 = d_i$ and $b_i^0 = Q$. However, the pricing of elementary routes requires the definition of $|V_+|$ additional resources. For each arc $(u,v) \in A$ and each $i \in V_+$, w_{uv}^i is defined as 1 if $i = v$ and 0 otherwise, and for each $i \in V_+$, $a_i^i = 1$ and $b_i^i = 1$.

5.2 The q-route Relaxation

The pricing of q-routes can be computed by the following dynamic programing algorithm. Let T be a dynamic programming matrix of dimension $|V|\times(Q+1)$. Let each entry $T(i,q)$ be the minimum reduced cost value of a path that starts at the depot, visits a set of customers (some may be visited more than once) and ends at customer i with cumulative demand of q (including d_i). Initially, define $T(i,q) = +\infty$, for all $i \in V$ and $q \in \{0,\ldots,Q\}$. Then, use the following recurrence equation to fill matrix T, which gives a complexity of $\mathcal{O}(n^2Q)$:

$$\begin{cases} T(i,d_i) &= \bar{c}_{0i} \\ T(i,q) &= \min_{j\in V_+}\{T(j,q-d_i)+\bar{c}_{ji}\} \end{cases}$$

Houck et al. (30) and Christofides et al. (11) noticed that one can find minimum cost q-routes without 2-cycles (subpaths $i \to j \to i$, $i \neq 0$) without changing this complexity. Restricting the q-routes to those without such cycles improves the SPF bounds. Remark that minimum cost q-routes without s-cycles can be found in $\mathcal{O}(s!s^2n^2Q)$ time (32, 24), which is pseudo-polynomial for fixed s. Of course, larger values of s give stronger bounds. However, experiments with q-routes with s-cycle elimination in (24) suggest that it is not worthy to use values of s above four. From this point, the pricing indeed becomes much slower. Unhappily, 4-cycle elimination is not enough to obtain bounds really close to the elementary bounds in some instances.

5.3 The ng-Route Relaxation

A simple alternative to cycle elimination for obtaining partial elementarity in the routes would be to select a subset S of customers to forbid being revisited. The algorithm for the SPPRC would be run defining unitary resources only for those customers. However, this is not likely to produce near-elementary routes if $|S|$ is significantly smaller than $|V_+|$. Recently, Baldacci et al. (4) devised a better way to impose partial elementarity. Instead of carrying the information that a certain customer was already visited by a path in all its extensions, a more limited memory mechanism is proposed. It takes advantage of the fact that the cycles that are likely to appear when pricing non-elementary routes, even when they are too long to be efficiently avoided by s-cycle elimination, only use edges with small cost and are confined to relatively small "neighborhoods" in the graph.

This new route relaxation, coined ng-route relaxation in (4) was introduced for both the CVRP and VRPTW, and it was later extended to the

Generalized VRP (GVRP) by (6), where it was used to solve transformed Capacitated Arc Routing Problem (CARP) instances.

For each customer $i \in V_+$, let $NG(i) \subseteq V_+$ be a subset of customers that have a relationship with i, such that $NG(i) \ni i$. A possible representation for this relationship can be a neighborhood relationship, i.e., $NG(i)$ contains the nearest customers of i, including i. These sets are called ng-sets, and they contain the customers that customer i is able to "remember." For instance, when a path P is being built, by the time it arrives at customer i, it has a set $\Pi(P)$ that represents its "memory" thus far. If customer i belongs to set $\Pi(P)$, the extension is forbidden. On the other hand, if i does not belong to set $\Pi(P)$, the extension is allowed and set $\Pi(P)$ is updated to "forget" the customers that customer i is not able to "remember," i.e., the customers that do not belong to $NG(i)$. It is clear that if a customer is "forgotten," it can be used to form a cycle in future extensions of path P. At this point, we can conclude that the size of the ng-sets is an important factor in the quality of solutions because the larger the ng-sets are, the greater the smallest cycles that can appear in a path. The size of each ng-set $NG(i)$ is limited by $\Delta(NG(i))$, which is a parameter defined *a priori*. Obviously, this size also changes the pricing complexity.

Let $P = (0, i_1, \ldots, i_{p-1}, i_p)$ be a path starting at the depot, visiting a sequence of customers and ending at customer i_p and $V(P)$ be the set of customers visited by path P. The set $\Pi(P)$ of prohibited extensions (the "memory") of path P can be defined as follows:

$$\Pi(P) = \left\{ i_k \in V(P) \backslash \{i_p\} : i_k \in \bigcap_{s=k+1}^{p} NG(i_s) \right\} \cup \{i_p\}. \tag{5.1}$$

That is, $\Pi(P)$ contains customer i_p and every customer i_k, $k = 1, \ldots, p-1$, of P that belongs to all sets $NG(i_{k+1}), \ldots, NG(i_p)$. Given $q(P) = \sum_{i \in V(P)} d_i$ as the total demand (load) collected by path P and $\bar{c}(P)$ as the total reduced cost of path P, let $L(P) = (\bar{c}(P), v(P), q(P), \Pi(P))$ be a label associated with a path P, which ends at customer $v(P) = i_p$. We say that a label $L(P)$ can be extended to a vertex i_{p+1} if the following two conditions hold:

1. $i_{p+1} \notin \Pi(P)$ and
2. $q(P) + d_{i_{p+1}} \leq Q$.

If the extension is performed, the vertex i_{p+1} becomes the last vertex of a new path $P' = (0, \ldots, i_p, i_{p+1})$ and a new label $L(P')$ can be obtained from $L(P)$ by the following operations:

$$L(P') = (\bar{c}(P) + \bar{c}_{i_p i_{p+1}}, i_{p+1}, q(P) + d_{i_{p+1}}, \Pi(P) \cap NG(i_{p+1}) \cup \{i_{p+1}\}). \tag{5.2}$$

If vertex i_{p+1} is the depot, then the new path P' defines a route. From this point, $L(P')$ can not be extended to any vertex. We call ng-routes (resp. ng-paths) w.r.t ng-sets $NG(i)$ the routes (paths) obtained by any label setting algorithm that implements the above ideas. An ng-route allows multiple visits to a customer i, on the condition that at least one customer j such that $i \notin NG(j)$ is visited between successive visits.

A forward dynamic programming algorithm can be used to compute ng-routes. This algorithm is exponential on the size of $\Delta(NG(i))$, remaining pseudo-polynomial for fixed $\Delta(NG(i))$. Furthermore, its efficiency depends on the use of some techniques to speed up its execution.

Let $P_1 = (0, i_1, \ldots, i_{p-1}, i_p)$ and $P_2 = (0, i_{p+u}, \ldots, i_{p+2}, i_{p+1})$ be feasible ng-paths w.r.t ng-sets $NG(i)$, associated, respectively, with labels $L(P_1)$ and $L(P_2)$, we say that P_2 is a feasible completion for P_1 if the following proposition holds:

Proposition 1 *The route $R = (0, i_1, \ldots, i_{p-1}, i_p, i_{p+1}, i_{p+2}, \ldots, i_{p+u}, 0)$ given by the concatenation of $L(P_1)$ and $L(P_2)$ is a feasible ng-route w.r.t ng-sets $NG(i)$ iff $q(P_1) + q(P_2) \leq Q$ and $\Pi(P_1) \cap \Pi(P_2) = \emptyset$.*

Proof 1 *Obviously, if $q(P_1) + q(P_2) > Q$ then the resulting route can not be a feasible ng-route because it does not satisfy the capacity constraint imposed to feasible routes.*

Suppose that there exists a customer $v \in (\Pi(P_1) \cap \Pi(P_2))$ and that the last occurrence of v on P_1 has index k. Thus, $v \in NG(i_k) \cap \cdots \cap NG(i_p)$. Analogously, let k' be the index of the last occurrence of v on P_2. Therefore, $v \in NG(i_{p+1}) \cap \cdots \cap NG(i_{k'})$. The route R is not a feasible ng-route w.r.t ng-sets $NG(i)$ since the repetition of v on cycle $H = (i_k = v, \ldots, i_p, i_{p+1}, \ldots, v = i_{k'})$ does not respect the ng-sets.

If $\Pi(P_1) \cap \Pi(P_2) = \emptyset$ then there are two cases. If the resulting route has no cycle H passing by (i_p, i_{p+1}) then it must be a feasible ng-route w.r.t ng-sets $NG(i)$ because the cycles on P_1 or on P_2 are allowed, since both P_1 and P_2 are feasible ng-paths. Otherwise, if it has such cycle, then H must be allowed by the ng-sets, because any vertex v that repeats in H does not belong to both $\Pi(P_1)$ and $\Pi(P_2)$.

If P_2 is a feasible completion for P_1 and $L(P_1)$ is concatenated with $L(P_2)$, the resulting new label represents a route with total demand (load) and total reduced cost given by, respectively, $q(P_1) + q(P_2)$ and $\bar{c}(P_1) + \bar{c}_{i_p i_{p+1}} + \bar{c}(P_2)$.

To reduce the number of possible labels, a dominance rule is incorporated into the algorithm. A label $L(P_1)$ dominates a label $L(P_2)$ if every feasible completion of P_2 yields a route with reduced cost not smaller than the feasible

route obtained by applying the same completion into P_1. Sufficient conditions for that are:

$$(i)\ v(P_1) = v(P_2), \quad (ii)\ q(P_1) \leq q(P_2), \quad (iii)\ \Pi(P_1) \subseteq \Pi(P_2), \text{ and}$$

$$(iv)\ \bar{c}(P_1) \leq \bar{c}(P_2).$$

The maximum number of non-dominated labels is bounded by $2^{\Delta-1}nQ$, where $\Delta = \max_i\{\Delta(NG(i))\}$. Unless Q is very large, if Δ is small the algorithm is guaranteed to be reasonably fast. However, large values of Δ may cause an exponential explosion on the number of labels.

5.4 Pricing Algorithms

As discussed earlier, a basic *ng*-route relaxation implementation does not allow the use of large *ng*-sets, a limitation which weakens the quality of the lower bounds found. To address this issue, we provide an efficient implementation, adapting the Decremental State-Space Relaxation for the *ng*-route relaxation. This technique was introduced in (54) to solve the ESPPRC. The original version of the algorithm helps reduce the number of labels to be managed during the dynamic programming algorithm that builds elementary paths. First, it relaxes the elementarity of the paths and, at each iteration, identifies which customers are being repeated on the best path found and then prohibits the repetition of these customers in subsequent iterations.

The main difference of our algorithm is that instead of relaxing the elementarity of the paths, the new algorithm relaxes the *ng*-set of each customer, therefore relaxing the *ng*-route restrictions.

The algorithms presented also use bidirectional dynamic programming that exploits the symmetry of the CVRP and completion bounds.

5.4.1 Forward Labeling

The basic exact forward dynamic programming labeling algorithm assigns a label $L(P) = (\bar{c}(P), v(P) = i, q(P), \Pi(P), pred(P))$ to each feasible *ng*-path $P = (0, \ldots, i)$, $i \in V$, storing its reduced cost, end vertex, load, set of prohibited extensions, and a pointer to its predecessor label. Each pair (i, q) where $i \in V$ and $q \in \{0, \ldots, Q\}$ defines a *bucket* $F(i, q)$. Buckets $F(i, q)$ are used to store labels representing paths that start at the depot and end at vertex i with total load equal to q. Only non-dominated labels are stored in the buckets.

Algorithm 5.1 presents the pseudocode of our basic Forward Labeling. At first, we set $F(0,0)$ with a single starting label $\mathcal{L} = (0,0,0,\emptyset,nil)$ and all other buckets with no label. Next, a forward dynamic programming is used to fill the buckets $F(i,q)$, running from $q = 0$ up to $q = Q$. When processing the bucket $F(i,q)$, the algorithm iterates through all labels $L(P_1)$ belonging to $F(j, q-d_i)$, for all vertices $j \in V_+$, such that $q - d_i > 0$. As the basic ng-route relaxation algorithm, the extension from $L(P_1)$ to i can only be performed if $i \notin \Pi(P_1)$. If this condition holds, a new label, say $L(P_2)$, is then created and it must be stored in the bucket $F(i,q)$. Therefore, this is the right time to check for the dominance rule, which can be verified for all the labels in $F(i,q'), \forall q' \leq q$. Surprisingly, we have found that the algorithm usually runs faster if the dominance rule is tested only for labels of the same bucket, i.e., for the labels from inside $F(i,q)$ (unless Q is very large). This comes from the fact that labels associated with paths using less load are less likely to dominate others using higher load. To accelerate the checking for dominated labels, we keep the labels of a same bucket ordered by reduced cost.

Another way to improve the implementation of the dominance rule is described in (4). When the paths of the bucket $F(i,q)$ are being computed, the algorithm uses a dominance list associated with the customer i, which stores the best costs for every possible configuration of $\Pi(P)$ and $q' < q$, to do the dominance. This way, it is faster to check the list than to iterate through the buckets $F(i,q')$ for each $q' < q$. We do not use this technique because it is not scalable, as the size of the dominance list is exponential in the value of $\Delta(NG(i))$, reaching its size limit when $\Delta(NG(i)) \approx 13$.

Let $NG(0)$ be defined as $\{0\}$. At the end of Algorithm 5.1 each bucket $F(0,q)$, $1 \leq q \leq Q$, will contain a single label representing the minimum reduced cost route with load q. The best routes are extracted by procedure *buildRoutes*, which uses the pointers stored in the labels to build the routes.

5.4.2 Symmetric Bidirectional Labeling

We can take advantage of the symmetry of the CVRP to develop the Symmetric Bidirectional Labeling algorithm, whose pseudocode is presented in Algorithm 5.2. It starts by running the Forward Labeling in order to obtain all non dominated labels representing paths with load $q \leq Q/2$. In fact, an additional extension beyond the load $Q/2$ must be performed, as pointed in (53). This happens because the demand of a customer must be fully serviced by a path that visits it, thus there may be a path that can not be splitted exactly in the middle of its load. In view of this, the new algorithm actually performs

Algorithm 5.1 Forward Labeling Algorithm

```
1: procedure FORWARDLABELING(N)
2:     input: ng-sets NG(i) ⊆ V+, ∀i ∈ V+.
3:     output: the best ng-routes w.r.t ng-sets NG(i).

4:     F(i,q) ← ∅, ∀i ∈ V, q ∈ {0,...,Q}
5:     F(0,0) ← {(0,0,0,∅,nil)}
6:     for q := 0,...,Q do
7:         for all i ∈ V do
8:             if q − d_i > 0 then
9:                 for all j ∈ V+ do
10:                    for all L1 = (c̄1, j, q − d_i, Π1, _) ∈ F(j, q − d_i) do
11:                        if i ∉ Π1 then
12:                            L2 = (c̄1 + c̄_ji, i, q, (Π1 ∩ NG(i)) ∪ {i}, pointer to L1)
13:                            insert ← true
14:                            for all L ∈ F(i,q) do
15:                                if L2 dominates L then delete L
16:                                else if L dominates L2 then insert ← false, break
17:                            if insert then
18:                                F(i,q) ← F(i,q) ∪ L2

19:    return buildRoutes()
```

the Forward Labeling to obtain all labels $L(P)$ ($q(P) = q$, $v(P) = i$) where $q - d_i \leq Q/2$ (this imposes a slight change in the pseudocode of Algorithm 5.1, because now line 8 must include the condition that tests if $q - d_i \leq Q/2$). Next, the routes are obtained by a concatenation step that exploits the symmetry of the problem, as explained below.

Let $B(j, q)$ be a bucket containing all non dominated labels representing paths that start at the depot and end at customer j with total load *less* than or equal q. The routes with load $q \leq Q/2$ are obtained from buckets $F(0, q)$, as before. All the other routes are obtained by concatenating the labels from $F(i, q)$ with the labels from $B(j, Q - q)$, for all $i \in V_+$, $j \in V$ and $q > Q/2$. Since the concatenation can be time consuming, we use the fact that the labels are sorted in the buckets in increasing reduced cost order (IRCO) to accelerate this step. Procedure *Save* saves pointers to labels for subsequent concatenation and procedure *buildRoutes* concatenates the labels to build the routes.

Remark that if the problem is represented using a directed graph (e.g., if the problem is the Asymmetrical CVRP), the associated pricing will be performed over an asymmetric reduced cost matrix. In this case, the bidirectional algorithm must explicitly run some backward labeling procedure before the concatenation, extending the labels from the depot but following the opposite direction of the arcs. In the case of the ACVRP, the backward procedure can be performed running the Forward Labeling over the transposed cost matrix.

Algorithm 5.2 Symmetric Bidirectional Labeling Algorithm

procedure BIDIRECTIONALLABELING($\mathbf{N}$)
  input: ng-sets $NG(i) \subseteq V_+$, $\forall i \in V_+$.
  output: the best ng-routes w.r.t ng-sets $NG(i)$.

  Run Forward Labeling to obtain all labels $L(P)$ where $q(P) - d_{v(P)} \leq Q/2$

  $B(i,q) \leftarrow F(i,q), \forall i \in V, q \in \{0,\ldots,Q/2\}$ ▷ Copies only the pointers
  for all $i \in V$ **do**
    for $q := 1,\ldots,Q/2$ **do**
      for all $L \in B(i,q-1)$ **do**
        $B(i,q) \leftarrow B(i,q) \cup \{L\}$ ▷ Copies only the pointers

  for all $i \in V_+$ **do**
    for all $j \in V$ such that $j \neq i$ **do**
      for $q = Q/2+1$ to Q **do**
        for all $L_1 = (\bar{c}_1, i, q, \Pi_1, \_) \in F(i,q)$ in IRCO **do**
          for all $L_2 = (\bar{c}_2, j, \_, \Pi_2, \_) \in B(j, Q-q)$ in IRCO **do**
            $\bar{c}_r \leftarrow \bar{c}_1 + \bar{c}_{ij} + \bar{c}_2$
            if $\bar{c}_r \geq 0$ **then break**
            if $\Pi_1 \cap \Pi_2 = \emptyset$ **then**
              $Save$(pointer to L_1, pointer to L_2)

  return buildRoutes()

5.4.3 Decremental State-Space Relaxation

The adapted DSSR is an iterative algorithm and it works by relaxing the state-space of the original ng-sets $NG(i)$. At each iteration k, the algorithm uses the subsets $\Gamma_i^k \subseteq NG(i)$ as a replacement for $NG(i)$. These subsets Γ_i^k take the role of $NG(i)$ in the definition of the function Π, described in Equation (5.1), and in the creation of new labels, as shown in Equation (5.2). The algorithm initializes Γ_i^0, $\forall i \in V$, as empty sets and executes the basic forward dynamic programming Algorithm 5.1. As the best routes found by this dynamic programming are not necessarily ng-routes w.r.t. the original ng-sets $NG(i)$, they cannot be considered as the output of the pricing without verifying their feasibility. This test is performed and the Γ_i^k subsets are updated if necessary, as described hereafter. If at the end of iteration k some subset Γ_i^k is updated, the forward dynamic programming algorithm is executed again with new subsets Γ_i^{k+1}.

Let a cycle of customers be defined as a sub-path $H = (i,\ldots,j)$, where $i = j$, and let $\mathscr{H}(P)$ be the set of all cycles of customers in the path P. To evaluate if the best route R_k^* found in the end of iteration k is an ng-route, the algorithm must check if there is no cycle $H \in \mathscr{H}(R_k^*)$ which would not

be allowed to be created if the original *ng*-sets $NG(i)$ were being used. This happens only when the first customer j of cycle H (we also refer to it as the repeated customer of the cycle) is in all $NG(l)$, $\forall l \in V(H)$, i.e., customer j is not "forgotten" by any other customer of the cycle. In this case, we designate such a cycle H as a *forbidden* cycle w.r.t the original *ng*-sets $NG(i)$. If any such cycle H is found, we add the repeated customer j to subsets Γ_l^{k+1}, $\forall l \in V(H)$. This prohibits cycle H from appearing in any path obtained in the next iterations. More than that, this prohibits any cycle $H^{'} = (j, \ldots, j)$ in which $V(H^{'}) \subseteq V(H)$ from appearing in the next iterations. On the other hand, if no such forbidden cycle is found at the end of iteration k, then the best route R_k^* is an *ng*-route and the algorithm stops.

Algorithm 5.3 reports the DSSR pseudocode. The input parameter of the algorithm are the original *ng*-sets $NG(i)$, $\forall i \in V$. The procedures with self-explanatory names *selectBestRoute*, *isNGRoute* and *updateNGSets* are responsible, respectively, for extracting the best route of a set of routes, determining if a given route is a valid *ng*-route with respect to *ng*-sets $NG(i)$ and to update the subsets Γ_i^k to the next iteration. Algorithm 5.3 also uses the procedure *forwardLabeling*, which is presented in Algorithm 5.1, to obtain *ng*-routes with respect to subsets Γ_i^k passed as input parameters.

It is noteworthy to mention that if the best route found is indeed an *ng*-route, the algorithm can stop and return only this route. However, if the objective is to find a set of feasible solutions, as is the case of the pricing, the algorithm can also return this route together with any other route certified to be an *ng*-route. In fact this is a small drawback of DSSR, since the state-space relaxation may produce the undesirable effect of losing some suboptimal routes. Nevertheless, if the best route is not an *ng*-route and one needs to find *any* feasible solution, any route that is indeed an *ng*-route can be returned, even if the best route is not feasible. In this case, we consider it as being a heuristic run of the algorithm, not an exact one. This method is useful to quickly price routes on intermediate iterations of column generation algorithms, where there is no need to generate the optimal solution.

5.4.4 Decremental State-Space Relaxation with Bidirectional Search

Righini and Salani (54) also solve the ESPPRC by combining the Decremental State-Space Relaxation technique with bidirectional dynamic programming. Following their ideas, we also built a similar algorithm to compute *ng*-routes, whose pseudocode is shown in Algorithm 5.4. The new algorithm works exactly as Algorithm 5.3, but instead of solving each iteration

Algorithm 5.3 Pure DSSR Algorithm

procedure DSSR(**N**)
  input: ng-sets $NG(i) \subseteq V_+$, $\forall i \in V_+$.
  output: the best ng-routes w.r.t ng-sets $NG(i)$.

  $\Gamma_i \leftarrow \emptyset, \forall i \in V$, $ng \leftarrow$ **false**, $k \leftarrow 0$
  while not ng **do**
    $\mathcal{R} \leftarrow$ forwardLabeling(Γ)
    $R_k^* \leftarrow$ selectBestRoute($\mathcal{R}$)
    if isNGRoute(R_k^*) **then**
      $ng \leftarrow$ **true**
    else
      updateNGSets(**N**, R_k^*)
    $k \leftarrow k + 1$
  return buildRoutes()

procedure isNGRoute(R)
  for all $H = (v, \ldots, v) \in \mathscr{H}(R)$ **do**
    $forbiddenCycle \leftarrow$ **true**
    for all $l \in V(H)$ **do**
      if $v \notin NG(l)$ **then**
        $forbiddenCycle \leftarrow$ **false**
        break
    if $forbiddenCycle$ **then**
      return false
  return true

procedure updateNGSets(**N**, R)
  for all $H = (v, \ldots, v) \in \mathscr{H}(R)$ **do**
    $forbiddenCycle \leftarrow$ **true**
    for all $l \in V(H)$ **do**
      if $v \notin NG(l)$ **then**
        $forbiddenCycle \leftarrow$ **false**
        break
    if $forbiddenCycle$ **then**
      for all $l \in V(H)$ **do** $\Gamma_l \leftarrow \Gamma_l \cup \{v\}$

of the DSSR using the Forward Labeling (Algorithm 5.1) it uses the Symmetric Bidirectional Labeling (Algorithm 5.2).

Algorithm 5.4 DSSR with Bidirectional Search Algorithm

procedure DSSRwithBidSearch(**N**)
  input: ng-sets $NG(i) \subseteq V_+$, $\forall i \in V_+$.
  output: the best ng-routes w.r.t ng-sets $NG(i)$.

  $\Gamma_i \leftarrow \emptyset, \forall i \in V$, $ng \leftarrow$ **false**, $k \leftarrow 0$
  while not ng **do**
    $\mathcal{R} \leftarrow$ bidirectionalLabeling(Γ)
    $R_k^* \leftarrow$ selectBestRoute($\mathcal{R}$)
    if isNGRoute(R_k^*) **then**
      $ng \leftarrow$ **true**
    else
      updateNGSets(**N**, R_k^*)
    $k \leftarrow k + 1$
  return buildRoutes()

5.4.5 Completion Bounds

Besides dominance, there is a second mechanism for eliminating labels along the labeling algorithms. If it can be proved that a partial path P can not be completed into a route with negative reduced cost, then $L(P)$ can be removed from its bucket. Of course, it is not possible to find the exact cost of the best completion for every P without running the full original algorithms. Instead, one needs to devise *completion bounds*, i.e., lower bounds on the cost of the completions; they should be obtained in a relatively fast way.

To further speed up the Algorithm 5.3, we calculate completion bounds during the DSSR in a similar manner as performed by Pecin (46) for the elementary route. At the end of iteration k, the completion bounds are calculated for each customer $i \in V_+$ with every load $q \in \{0, \ldots, Q\}$, and then used at iteration $k + 1$. As mentioned before, the completion bounds are used to estimate a lower bound on the value of a route during its creation, thus discarding any route that would not lead to a negative reduced cost. Given $T_k^*(i, q)$, the value of the best path which starts at customer i and ends at the depot with load equal q (including d_i), the completion bounds $\widehat{T}_k(i, q)$ are calculated as shown in Equation (5.3) and represents a lower bound on the value of the best path that starts at customer i and ends at the depot with load less than or equal q.

$$\widehat{T}_k(i,q) = \min_{q' \leq q} \{T_k^*(i,q')\}\,. \tag{5.3}$$

It is important to observe that due to the symmetry of the CVRP, $T_k^*(i,q)$ can be obtained directly from the buckets $F(i,q)$. This is true because the value of the best path that starts at the depot and ends at customer i with total load equal q has the same value as $T_k^*(i,q)$, as shown by (4). On the other hand, if the problem is the ACVRP, to obtain these values, the direction of the edges has to be reversed, as also shown by (4). In this case, the last iteration of the DSSR algorithm has to be executed again before the calculation.

After calculating the completion bounds at the end of iteration $k-1$, they can be used at iteration k to avoid the extension of a given label $L(P) = (\bar{c}(P), j, q(P), \Pi(P))$ to a customer i if the following conditions hold:

$$\bar{c}(P) + \bar{c}_{ji} + \widehat{T}_{k-1}(i, Q - q(P)) \geq 0. \tag{5.4}$$

This equation calculates a lower bound on the value of the reduced cost of any route the label $L(P)$ can generate, because $\widehat{T}_k(Q - q(P), i)$ is a lower bound on the value of the best path, which would close path P after it is extended to customer i. Obviously, if the value of Equation (5.4) is greater or equal than zero, the label $L(P)$ cannot generate any route with a negative reduced cost, and therefore, it can be discarded. It is interesting to highlight that the completion bounds becomes stronger along the iterations of the DSSR, because iteration k is a relaxation of iteration $k+1$, i.e., given that $\Gamma_i^k \subseteq \Gamma_i^{k+1}$, $\forall i \in V_+$.

Algorithm 5.5 reports the pseudocode for pricing *ng*-routes with DSSR and completion bounds. It differs from Algorithm 5.3 only due to the use of completion bounds. The procedures *selectBestRoute*, *isNGRoute* and *updateNGSets* have the same meaning as before and *generateCompletionBounds* is a new procedure that calculates the bounds as shown in Equation (5.3). The procedure *boundedForwardLabeling* is a slight modification of procedure *forwardLabeling* of Algorithm 5.1, in order to include the completion bounds calculated as shown in Equation (5.4).

Finally, Algorithm 5.6 combines *ng*-routes, DSSR, symmetric bidirectional search and completion bounds. It differs from 5.4 only by using completion bounds. Given $\bar{c}(R_k^*)$, the reduced cost of the best route at the k-th iteration of the DSSR, this algorithm calculates completion bounds when $\bar{c}(R_k^*) > \rho\bar{c}(R_{k'}^*)$ (we use $\rho = 0.5$), where $k^{'} < k$ is the iteration in which the last calculation was performed. These completion bounds are obtained by running the $boundedForwardLabeling$ procedure. Next, the $bidirectionalLabeling$ procedure uses these bounds by running the $boundedForwardLabeling$ instead of

Algorithm 5.5 DSSR with Completion Bounds Algorithm

procedure DSSRwithBounds($\mathbf{N}$)
  input: ng-sets $NG(i) \subseteq V_+$, $\forall i \in V_+$.
  output: the best ng-routes w.r.t ng-sets $NG(i)$.

  $\Gamma_i \leftarrow \emptyset, \forall i \in V$, $ng \leftarrow$ **false**, $k \leftarrow 0$
  $\widehat{T}(i,q) \leftarrow -\infty, \forall i \in V_+, q \in \{0,\ldots,Q\}$
  while not ng **do**
    $\mathcal{R} \leftarrow$ boundedForwardLabeling(Γ, $\widehat{T}$)
    $R_k^* \leftarrow$ selectBestRoute($\mathcal{R}$)
    if isNGRoute(R_k^*) **then**
      $ng \leftarrow$ **true**
    else
      updateNGSets($\mathbf{N}$, R_k^*)
      $\widehat{T} \leftarrow$ generateCompletionBounds(F)
    $k \leftarrow k+1$
  return buildRoutes()

procedure BoundedForwardLabeling(Γ, $\widehat{T}$)
  $F(i,q) \leftarrow \emptyset, \forall i \in V, q \in \{0,\ldots,Q\}$
  $F(0,0) \leftarrow \{(0,0,0,\emptyset,nil)\}$
  for $q := 0,\ldots,Q$ **do**
    for all $i \in V$ **do**
      if $q - d_i > 0$ **then**
        for all $j \in V_+$ **do**
          for all $L_1 = (\bar{c}_1, j, q-d_i, \Pi_1, \_) \in F(j, q-d_i)$ **do**
            if $i \notin \Pi_1$ **then**
              if checkCompletionBound($\widehat{T}$, L_1, i) **then**
                $L_2 = (\bar{c}_1 + \bar{c}_{ji}, i, q, (\Pi_1 \cap NG(i)) \cup \{i\}, \text{pointer to } L_1)$
                $insert \leftarrow$ **true**
                for all $\mathcal{L} \in F(i,q)$ **do**
                  if L_2 dominates $\mathcal{L}$ **then** delete $\mathcal{L}$
                  else if $\mathcal{L}$ dominates L_2 **then**
                    $insert \leftarrow$ **false**, **break**
                if $insert$ **then**
                  $F(i,q) \leftarrow F(i,q) \cup L_2$

  return buildRoutes()

procedure generateCompletionBounds(F)
  $\widehat{T}(i,q) \leftarrow -\infty, \forall i \in V_+, q \in \{0,\ldots,Q\}$
  $\widehat{T}(0,0) \leftarrow 0$
  for $i \in V_+$ **do**
    for $q := 1,\ldots,Q$ **do**
      $\widehat{T}(i,q) \leftarrow \min(F(i,q))$ ▷ Minimum red. cost or $+\infty$, if empty
      if $\widehat{T}(i,q-1) < \widehat{T}(i,q)$ **then**
        $\widehat{T}(i,q) \leftarrow \widehat{T}(i,q-1)$

the Forward Labeling in line 4. The computation of completion bounds is also accelerated with the bounds obtained in the previous calculation.

Algorithm 5.6 DSSR with Bidirectional Search and Completion Bounds

1: **input:** ng-sets $NG(i) \subseteq V_+$, $\forall i \in V_+$.
2: **output:** the best ng-routes w.r.t ng-sets $NG(i)$.

3: $\Gamma_i \leftarrow \emptyset, \forall i \in V$, $ng \leftarrow$ **false**, $k \leftarrow 0$
4: $\widehat{T}(i,q) \leftarrow -\infty, \forall i \in V_+, q \in \{0,\ldots,Q\}$
5: $last \leftarrow -\infty$
6: **while** not ng **do**
7: $\quad\mathcal{R} \leftarrow$ symmetricBidirectionalLabeling(Γ) ▷ Uses $\widehat{T}$
8: $\quad R_k^* \leftarrow$ selectBestRoute($\mathcal{R}$)
9: $\quad$**if** isNGRoute(R_k^*) **then**
10: $\quad\quad ng \leftarrow$ **true**
11: $\quad$**else**
12: $\quad\quad$updateNGSets(R_k^*)
13: $\quad k \leftarrow k+1$
14: $\quad$**if** $\bar{c}(R_k^*) > \rho.last$ **then**
15: $\quad\quad$boundedForwardLabeling(Γ, $\widehat{T}$)
16: $\quad\quad\widehat{T} \leftarrow$ generateCompletionBounds(F)
17: $\quad\quad last \leftarrow \bar{c}(R_k^*)$
18: **return** buildRoutes()

5.4.6 Heuristic Pricing

Even with the improvements described in sections 5.4.3, 5.4.4 and 5.4.5, the exact ng-route pricing still requires a long time to be executed. Because of this, a simple but effective heuristic was developed to quickly price a large initial set of routes with negative reduced cost. It was based on the heuristic pricing done for the elementary route pricing by Pecin (46). The purpose of this heuristic is to reduce the number of calls to the exact ng-route pricing. Therefore, the heuristic ng-route pricing is used as a hot-start for the exact ng-route pricing.

The heuristic closely resembles the q-route pricing without eliminating any cycle. The main difference between the pricing algorithms is that when extending one path, the heuristic ng-route pricing respects the ng-sets $NG(i)$. Its data structure is also a $(Q+1)\times|V_+|$ matrix, and each entry consists of just one label (a quite similar approach is presented in (24) for q-routes). For each customer and each capacity, this label is chosen as the best one with respect to the reduced costs. In addition, as the ng-sets $NG(i)$ must be respected, each label of the dynamic programming matrix must contain the Π sets for each

customer and capacity. This means that many non-dominated labels, those that are less likely to lead to optimal solutions, may be dropped.

Notice that unlike the exact algorithms, the heuristic algorithms use neither the dominance rules nor the speed-up techniques (DSSR, bidirectional search and completion bounds) described in Section 5.4. Nevertheless, they are responsible for obtaining more than 90% of the routes during the column generation. Moreover, it is straightforward to verify that its resulting complexity is $\mathcal{O}(n^2Q)$.

5.5 Achieving Elementarity

To achieve elementarity, we just define $\Delta(NG(i)) = V_+$, for all $i \in V_+$ and use the same algorithms based on DSSR described before. This means that the algorithms start with $\Gamma_i^0 = \emptyset, \forall i \in V_+$. At the end of each iteration k, they identify all cycles on the best solution, and the repeated customer of each cycle $H \in \mathscr{H}(R_k^*)$ is inserted on subsets Γ_l^{k+1}, $\forall l \in V(H)$. Thus, if cycle $H = (i, \ldots, j)$, $i = j$, is identified at the end of iteration k (that is, if cycle H belongs to $\mathscr{H}(R_k^*)$), the next DSSR iterations will not generate any path with a cycle $H' = (i, \ldots, j)$, in which $V(H') \subseteq V(H)$. Note, however, that it is still possible to obtain a path that visits customer j more than once at iteration $k+1$, as this customer is not present in all subsets Γ_l^{k+1}. The algorithms stop when the best route does not contain any cycle, i.e. it is an elementary route, or its reduced cost is non-negative.

That way of increasing the state-space along the DSSR iterations differs from the method used by Righini and Salani (54), allowing a better control on the growth of the number of labels, turning the algorithm capable of dealing with larger instances. The DSSR of (54) is performed prohibiting the customers which repeat on the best route R_k^* from repeating again in subsequent iterations until an elementary route is found. This is equivalent to inserting each repeated customer of each cycle $H \in \mathscr{H}(R_k^*)$ in all subsets Γ_i^{k+1}, rather than just considering this inclusion in subsets Γ_l^{k+1}, $\forall l \in V(H)$. We noted in computational experiments that this more aggressive increase of the state-space is quite dangerous because the whole algorithm fails if the number of labels to be treated in the dynamic programming becomes critical. In contrast, the size of the largest subset Γ_i^k hardly exceeded 20 in our algorithms, even for instances with 199 vertices.

Another important difference from our approach and that of (54) is the way we use the DSSR to calculate completion bounds at each iteration to accelerate the subsequent DSSR iterations. Computational results reveal that

the use of completion bounds allow us to solve the column generation for CVRP instances that would not be solved in an acceptable time if they were not used.

5.5.1 The Righini and Salani's Algorithm

The best exact algorithm for the ESPPRC presented by Righini and Salani (54) is based on DSSR combined with bidirectional dynamic programming. Our implementation of this algorithm corresponds to Algorithm 5.4 using $\Delta(NG(i)) = V_+$, for all $i \in V_+$. Additionaly, when running this algorithm as being the Righini and Salani's algorithm, the state-space along the DSSR iterations is increased by inserting each repeated customer of each cycle $H \in \mathscr{H}(R_k^*)$ in all subsets Γ_i^{k+1} instead of inserting in the subsets Γ_l^{k+1}, $\forall l \in V(H)$, as performed in our DSSR based algorithms.

5.6 Experimental Results

For the computational experiments, all algorithms were implemented in C++ using Microsoft Visual C++ 2010 Express and IBM ILOG CPLEX Optimizer 12.5 for solving the formulations. The experiments were conducted on an Intel Core i7-3960X 3.30GHz with 64GB RAM running Linux Ubuntu Server 12.04 LTS and are divided into three parts. First, we compare the six exact pricing algorithms described in this chapter, running each one inside a column generation schema for some classical CVRP instances. All six algorithms were tested using different values of $\Delta(NG(i))$, allowing us to analyze the scalability of each one when the state-space relaxation is increased. Second, using three of our algorithms we price elementary routes for these selected CVRP instances. We compare the results obtained with our own implementation of the best algorithm proposed by Righini and Salani (54). Third, to improve the lower bounds and also demonstrate that our algorithms still works well when robust cuts are added in the SPF, we included Rounded Capacity Cuts and Strengthened Combs using the separation routines from the CVRPSEP package (37). We separated and added these cuts in a similar manner as done by Fukasawa et al. (24) for the CVRP.

For all tests, the column generation starts by calling the heuristic pricing at each iteration. The heuristic pricing returns the best 20 routes with negative reduced costs. If the heuristic is no longer capable of finding routes with negative reduced cost, the column generation algorithm calls the exact pricing. If the latter succeeds in obtaining at least one route with negative reduced cost, the column generation procedure restarts by calling the heuristic pricing.

Otherwise, the column generation stops and the current value is returned as a lower bound. This procedure is stopped prematurely if the time limit of two hours is exceeded.

5.6.1 Problem Instances

We used just a representative set extracted from the classical instance datasets `A`, `B`, `E`, `P` and `M`. The general format of an instance name is X-nY-kZ, where X corresponds to the type of dataset, Y refers to the number of vertices and Z corresponds to the fix number of routes any feasible solution is constrained. As usually adopted by exact methods, for all problem instances, the cost matrix is calculated using Euclidean distance rounded to the nearest integer value.

All optimal values shown in the tables were extracted from the work of (24), except the optimal value for instance M-n151-k12, which was first proved by (12) and the optimal values for instances M-n200-k17 and M-n200-k16, which were first proved by the algorithm presented in Chapter 6.

5.6.2 Performance Evaluation

The results for the six algorithms are shown in Tables 5.1 and 5.2. Columns `Ins` and `OPT` show the name and the optimum value of each instance. Following these columns, the results for different values of $\Delta(NG(i))$ are shown. For each X, where $\Delta(NG(i)) = X$, `NG=X` consists of eight columns, `LB`, `It`, `T1`, `T2`, `T3`, `T4`, `T5`, and `T6` which show the lower bound, the number of calls to the exact pricing (for Algorithm 5.6) and the total time required to compute it for, respectively, Algorithms 5.1, 5.2, 5.3, 5.4, 5.5 and 5.6.

The results from Tables 5.1 and 5.2 show that for small ng-set sizes (`NG=8`), the best approach comes from Algorithm 5.2 followed by Algorithm 5.1. These algorithms do not use state-space relaxation. This is not a surprise because the maximum number of non-dominated labels per bucket is limited to $2^{\Delta-1}$, and therefore, the total number of labels handled by these algorithms does not explode. In this case, a unique run of the dynamic programming considering the entire state-space is in general better than running it in the relaxed state-space several times. For an average ng-set size (`NG=16`), the pure DSSR still does not improve the times of both the Algorithms 5.1 and 5.2, but the combination of DSSR and completion bounds provides a reasonable improvement. For large ng-set sizes (`NG`$\geq$`32`), the pure DSSR significantly outperforms the basic dynamic programming algorithms, but it is still a poor

algorithm for most instances. However, the DSSR with completion bound drastically improves the times.

We can also note that the bounds for `NG=32` are almost as good as the elementary for most instances (shown in Table 5.3). This is evidence that the routes found by the pricing with large ng-sets are almost elementary when the average size of the routes is up to 12 or 13 customers, in tipically less time than the time required if the elementary constraint is imposed to the routes. But this is not necessarily a rule. It is noteworthy to mention that the greater is the ng-set size used, the better are the completion bounds obtained. In some cases, the improved completion bounds resulting from the use of a large ng-set may compensate the additional complexity imposed. This is indicated by the time for running the instance M-n121-k7, which is significantly greater for `NG=64` than the elementary for Algorithm 5.5.

On the other hand, our algorithms spend a lot of time trying to solve the column generation for instance M-n121-k7, especially for large ng-set sizes. This is mainly due to the average size of the routes (n/K) that are part of an optimum solution for this instance, which is greater than 17, causing the number of labels to be treated by the dynamic programming algorithm to be prohibitive.

5.6.3 Other Results

Table 5.3 shows the results for the CVRP using three different elementary route pricing. Columns `RS`, `ALG4`, `ALG5` and `ALG6` present, respectively, the results of the Algorithms of Righini and Salani (implemented as described in Section 5.5.1), 5.4, 5.5 and 5.6. Algorithm 5.4 was chosen because it differs from the Righini and Salani's Algorithm only in the way it increases the state-space along the DSSR iterations and Algorithms 5.5 and 5.6 because they presented the best results for large ng-set sizes. For each pricing algorithm, columns T and Ite show, respectively, the total time of the column generation algorithm and the number of calls to the exact pricing. At the end of each exact pricing, the average size of the final subsets $\Gamma_i^k \subseteq V_+$ were calculated, as well as the size of the largest subset. Columns Avg and Max show for each pricing algorithm the global average and largest size in the column generation, i.e., calculated considering all iterations of the exact pricing.

As shown in Table 5.3, the algorithm proprosed by Righini and Salani (54) is not able to solve the ESPPRC for instances with more than 100 customers within a reasonable amount of time. In fact, their algorithm has poor performance also for instances of moderate size, such as the instances A-n63-

k10, B-n68-k9 e E-n76-k7. Even without using completion bounds, Algorithm 5.4 drastically outperforms the algorithm in (54) because the Avg and Max values it obtains are usually smaller than 10% of the corresponding values obtained in (54).

Finally, Table 5.4 presents the results of the column generation for the CVRP considering the separation of capacity and strengthened comb cuts. These results were obtained using Algorithm 5.6. As performed previously, the results are shown for different values of $\Delta(NG(i))$. Results for the elementary routes are also presented (column `Elem`). Note that, considering the randomness factor included on the heuristic cut separation algorithms, larger $\Delta(NG(i))$ do not necessarily implies in larger lower bounds, as one can check for example on instance E-n76-k8.

5.7 Conclusions

The strength of the *ng*-route pricing is the ability of adjusting the size of the *ng*-sets in order to calculate a lower bound in a reasonable time, which is close as much as possible to the elementary route bound, and this property justifies an efficient implementation of the method. In this chapter, we presented an efficient *ng*-route pricing algorithm for *ng*-set sizes up to sixty-four, a number at least two times greater than we know so far. Furthermore, we showed how our restricted non-elementary route pricing algorithm can be easily extended in order to price only elementary routes. We highlighted the two elements that allowed us to price elementary routes even for CVRP instances with 199 customers, a result which doubled the size of the ESPPRC instances solved so far. The first element is the way we adapt the Decremental State-Space Relaxation (DSSR) technique of (54) for the *ng*-routes context, thus improving their way of increasing the state-space along the DSSR iterations. The second is the combination of the DSSR technique with completion bounds, which are calculated in each iteration of the DSSR for the purpose of accelerating the next iteration.

Table 5.1: Results for column generation with NG=8 and NG=16.

		NG=8								NG=16							
Ins	OPT	LB	It	T1	T2	T3	T4	T5	T6	LB	It	T1	T2	T3	T4	T5	T6
A-n62-k8	1288	1250.24	4	0.97	0.71	4.68	3.42	0.90	1.13	1254.83	3	4.54	1.25	5.96	3.05	0.86	1.25
A-n63-k10	1314	1286.58	2	0.47	0.36	2.84	1.81	0.55	0.56	1286.81	2	1.92	0.60	3.29	2.15	0.46	0.57
A-n64-k9	1401	1368.24	3	0.77	0.62	5.78	4.85	0.72	1.04	1374.49	3	3.40	0.98	6.83	4.39	0.82	0.89
A-n69-k9	1159	1129.97	2	0.79	0.61	4.76	3.26	0.78	0.78	1131.33	3	6.30	1.37	7.05	4.44	0.87	1.06
A-n80-k10	1763	1729.81	3	1.53	1.20	17.92	8.63	1.53	1.88	1731.45	3	6.68	2.24	19.56	7.18	1.60	2.07
B-n50-k8	1312	1266.45	3	0.47	0.26	2.76	1.78	0.32	0.44	1266.63	4	8.13	1.03	4.39	2.82	0.40	0.67
B-n68-k9	1272	1198.20	2	1.06	0.59	5.89	3.75	0.77	0.90	1203.78	2	49.68	2.68	25.52	12.36	1.10	1.56
B-n78-k10	1221	1166.73	2	1.56	0.94	21.47	10.68	1.14	1.66	1167.46	4	60.18	2.65	18.92	14.08	1.39	1.78
E-n51-k5	521	517.14	2	0.66	0.50	2.24	1.76	0.45	0.76	517.14	3	5.72	1.88	3.57	2.78	0.58	0.88
E-n76-k7	682	664.20	3	2.20	2.06	14.75	12.12	2.16	2.98	664.82	4	22.27	7.96	16.39	14.26	2.66	4.12
E-n76-k8	735	717.82	2	1.56	1.24	6.52	4.98	1.51	1.84	718.74	3	7.36	2.47	12.02	6.05	1.41	1.91
E-n76-k10	830	811.77	3	1.15	0.80	5.37	3.87	0.90	1.33	811.87	2	4.37	1.18	4.84	3.33	0.86	1.17
E-n76-k14	1021	1001.85	1	0.42	0.35	1.84	1.22	0.40	0.46	1002.77	1	0.81	0.35	2.11	1.36	0.36	0.54
E-n101-k8	815	789.37	2	7.00	5.45	38.90	37.86	5.95	8.39	790.71	3	56.63	18.17	55.08	42.11	6.30	8.17
E-n101-k14	1067	1047.2	2	1.82	1.48	11.30	7.43	1.67	1.92	1048.45	3	8.44	2.69	17.49	11.13	2.06	2.41
M-n121-k7	1034	1026.37	7	23.36	15.30	415.17	209.60	21.47	52.97	1029.21	23	—	3227.18	—	5514.44	319.39	1076.23
M-n151-k12	1015	995.73	3	16.66	14.24	146.49	74.36	18.08	20.51	996.64	4	89.65	36.12	228.35	133.59	21.78	29.12
M-n200-k16	1274	1250.23	4	35.54	30.07	445.23	222.58	45.00	51.70	1250.87	4	241.71	78.17	576.33	407.19	66.45	73.78
M-n200-k17	1275	1252.52	4	34.53	30.27	383.96	306.60	51.99	57.12	1252.87	5	178.56	69.92	512.97	284.95	53.05	61.27
P-n50-k8	631	614.64	1	0.18	0.14	0.67	0.49	0.15	0.21	615.55	1	0.51	0.19	0.88	0.63	0.17	0.26
P-n70-k10	827	809.29	2	0.62	0.44	3.48	2.38	0.51	0.69	810.61	1	1.58	0.59	2.64	1.84	0.53	0.62
Avg		2.44		6.35	5.13	73.43	43.97	7.47	9.97	2.30		37.92	164.75	76.21	308.29	23.00	60.49
Total	21			21	21	21	21	21	21			20	21	20	21	21	21

Table 5.2: Results for column generation with NG=32 and NG=64.

		NG=32								NG=64							
Ins	OPT	LB	It	T1	T2	T3	T4	T5	T6	LB	It	T1	T2	T3	T4	T5	T6
A-n62-k8	1288	1254.83	4	4630.53	105.69	6.64	3.37	0.97	1.21	1254.83	5	—	—	7.94	4.98	1.17	2.45
A-n63-k10	1314	1286.83	4	311.87	13.12	6.24	3.90	0.80	1.06	1286.83	2	—	—	4.58	3.09	0.86	0.98
A-n64-k9	1401	1376.90	3	351.21	7.45	10.37	3.82	0.81	1.01	1376.90	3	—	—	7.86	3.74	1.02	1.22
A-n69-k9	1159	1131.34	3	2923.22	44.22	9.07	5.84	0.90	1.13	1131.34	3	—	—	7.54	4.66	1.11	1.32
A-n80-k10	1763	1731.58	3	1236.43	38.80	17.98	8.59	1.68	2.04	1731.58	4	—	—	23.86	11.55	2.35	2.98
B-n50-k8	1312	1266.64	2	2514.87	67.84	3.50	2.28	0.30	0.57	1266.64	3	—	—	3.17	1.72	0.46	0.53
B-n68-k9	1272	1204.00	4	—	310.53	41.20	15.14	1.48	2.22	1204.00	2	—	—	32.28	19.15	1.58	2.15
B-n78-k10	1221	1167.52	4	—	733.60	29.49	11.08	1.63	1.94	1167.52	2	—	—	19.15	9.96	1.60	1.76
E-n51-k5	521	517.14	2	4394.19	145.67	2.59	3.04	0.56	0.96	517.14	3	—	—	3.63	2.87	0.64	0.88
E-n76-k7	682	665.59	4		1154.16	26.04	19.60	2.96	5.01	665.59	3	—	—	26.63	15.38	2.86	4.25
E-n76-k8	735	718.78	2	2405.49	92.44	12.03	6.32	1.57	2.11	718.78	3	—	—	14.73	10.63	1.90	2.43
E-n76-k10	830	812.48	2	986.20	12.64	6.54	4.30	0.94	1.21	812.48	3	—	—	9.26	6.21	1.16	1.69
E-n76-k14	1021	1002.77	1	28.51	0.68	2.10	1.36	0.42	0.54	1002.77	1	—	32.43	2.14	1.45	0.44	0.60
E-n101-k8	815	790.99	3		3360.97	67.22	40.37	7.84	8.29	790.99	3	—	—	49.37	37.13	6.94	8.57
E-n101-k14	1067	1050.42	2	1346.88	30.53	13.98	9.15	2.03	2.26	1050.42	4	—	—	26.19	16.70	2.61	3.10
M-n121-k7	1034	1029.75	17	—	—	—	—	1290.24	1570.25	1029.79	21	—	—	—	6184.45	1823.49	1015.82
M-n151-k12	1015	997.28	4	—	3987.38	226.49	121.65	24.58	31.99	997.43	4	—	—	200.38	136.14	27.85	33.37
M-n200-k16	1274	1251.36	4	—	3170.13	697.03	321.93	60.37	70.48	1252.05	7	—	—	1050.65	451.87	73.51	88.53
M-n200-k17	1275	1253.43	5	—	3514.90	472.94	380.88	56.90	72.70	1254.01	5	—	—	566.72	446.68	63.18	73.21
P-n50-k8	631	615.55	1	34.24	0.44	0.92	0.63	0.20	0.27	615.55	1	—	86.66	0.94	0.68	0.24	0.31
P-n70-k10	827	810.94	1	221.09	2.65	2.72	1.88	0.51	0.60	810.94	1	—	3055.68	2.60	1.87	0.66	0.74
Avg		2.26		1644.98	839.69	82.75	48.26	69.41	84.66	2.26		—	1058.26	102.98	351.00	95.98	59.38
Total	21			13	20	20	20	21	21			0	3	20	21	21	21

Table 5.3: Results for column generation with elementary routes.

			RS				ALG4				ALG5				ALG6			
Ins	OPT	LB	T	Ite	Avg	Max	T	Ite	Avg	Max	T	Ite	Avg	Max	T	Ite	Avg	Max
A-n62-k8	1288	1254.83	61.48	5	42.20	45	4.55	4	3.53	11	2.27	7	4.00	12	2.23	4	3.62	12
A-n63-k10	1314	1286.83	3795.50	2	55.00	56	2.62	2	3.62	9	0.99	2	3.62	9	1.07	2	3.62	9
A-n64-k9	1401	1376.90	35.55	3	48.33	50	3.84	3	3.58	8	1.78	4	4.16	11	2.10	4	4.10	10
A-n69-k9	1159	1131.34	18.77	3	49.00	54	7.04	5	3.69	9	1.97	5	3.71	9	2.18	5	3.69	9
A-n80-k10	1763	1731.58	134.16	3	58.33	60	10.31	3	4.30	11	3.36	3	4.28	11	3.84	3	4.30	11
B-n50-k8	1312	1266.64	4.32	3	35.33	40	1.79	3	3.41	8	0.68	4	3.29	6	0.76	3	3.41	8
B-n68-k9	1272	1204.00	671.43	3	49.33	56	12.14	4	4.40	12	2.66	4	4.32	12	3.48	4	4.40	12
B-n78-k10	1221	1167.52	41.32	4	53.50	61	11.85	3	4.03	10	4.64	5	4.18	10	5.34	5	4.15	10
E-n51-k5	521	517.14	4.94	3	37.33	40	2.95	3	3.57	9	0.85	3	3.60	9	1.05	3	3.57	9
E-n76-k7	682	665.59	137.14	4	55.75	59	15.85	3	4.52	13	4.95	4	4.45	13	5.28	3	4.52	13
E-n76-k8	735	718.78	25.35	3	51.67	53	10.29	3	3.74	9	3.18	3	3.76	9	3.99	3	3.74	9
E-n76-k10	830	812.48	10.86	2	51.00	55	4.75	2	3.43	9	1.80	2	3.42	9	1.98	2	3.43	9
E-n76-k14	1021	1002.77	2.51	2	43.50	50	2.09	2	1.99	8	0.83	2	2.00	8	0.99	2	1.99	8
E-n101-k8	815	790.99	1298.22	3	72.67	77	41.10	3	6.17	15	13.50	4	6.18	14	13.48	3	6.17	15
E-n101-k14	1067	1050.42	19.34	2	64.50	70	10.67	2	3.43	10	4.36	2	3.98	10	4.66	2	3.98	10
M-n121-k7	1034	1029.79	—	—	—	—	—	—	—	—	1005.69	33	8.38	21	860.00	19	8.62	21
M-n151-k12	1015	997.43	—	—	—	—	189.98	5	5.41	15	56.05	5	5.11	15	56.93	4	5.14	15
M-n200-k16	1274	1252.05	—	—	—	—	421.52	5	4.98	13	147.44	6	5.00	17	157.59	6	4.97	17
M-n200-k17	1275	1254.01	—	—	—	—	520.05	6	5.30	18	146.41	5	5.20	18	165.83	6	5.12	18
P-n50-k8	631	615.55	0.86	1	36.00	36	0.64	1	3.00	8	0.28	1	2.98	8	0.40	1	2.98	8
P-n70-k10	827	810.94	3.33	1	50.00	50	2.20	1	3.32	10	0.91	1	3.32	10	1.02	1	3.32	10
Avg		2.26	368.53		50.20	53.65	63.81		3.97	10.75	66.89		4.24	11.48	61.63		4.23	11.57
Total	21		17				20				21				21			

Table 5.4: Results for column generation with robust cuts.

		NG=8			NG=16			NG=32			NG=64			Elem		
Ins	OPT	LB	It	Time	LB	It	Time	LB	It	Time	LB	It	Time	LB	It	Time
A-n62-k8	1288	1280.76	4	2.9	1282.08	3	3.1	1282.08	4	4.0	1281.99	3	3.0	1281.99	3	3.0
A-n63-k10	1314	1302.42	2	1.0	1303.87	3	1.2	1303.87	5	1.7	1303.97	5	2.0	1303.97	5	1.9
A-n64-k9	1401	1387.83	3	1.5	1389.64	3	1.5	1389.91	3	1.7	1389.99	2	1.9	1389.99	2	1.9
A-n69-k9	1159	1143.78	3	1.8	1145.36	2	1.6	1145.38	3	1.9	1145.34	2	1.8	1145.34	3	2.3
A-n80-k10	1763	1756.29	2	2.7	1756.95	3	3.6	1756.80	3	3.6	1756.80	3	4.6	1756.80	4	5.1
B-n50-k8	1312	1303.53	3	1.1	1303.62	2	0.8	1303.71	3	1.1	1303.72	4	1.1	1303.72	4	1.1
B-n68-k9	1272	1263.76	2	2.4	1263.93	3	3.1	1263.90	4	3.8	1263.9	3	4.1	1263.93	4	3.7
B-n78-k10	1221	1217.18	4	4.2	1217.38	5	4.3	1217.48	3	4.0	1217.62	3	4.5	1217.62	4	5.5
E-n51-k5	521	519.28	3	1.7	519.52	3	1.4	519.49	3	1.7	519.36	3	1.4	519.36	3	1.4
E-n76-k7	682	671.10	3	5.1	671.30	3	5.0	671.94	4	6.9	671.94	3	5.9	671.94	3	6.3
E-n76-k8	735	726.57	2	3.6	727.10	4	5.2	727.15	3	4.4	727.22	3	4.5	727.12	3	5.8
E-n76-k10	830	817.12	1	1.5	817.42	2	1.9	818.01	1	1.5	818.00	3	2.5	818.01	2	2.4
E-n76-k14	1021	1006.94	2	0.8	1007.53	1	0.7	1007.53	1	0.7	1007.5	1	0.8	1007.54	1	1.0
E-n101-k8	815	804.66	3	16.1	804.96	3	16.6	805.11	4	23.1	804.93	3	21.6	805.08	4	25.1
E-n101-k14	1067	1053.69	2	2.7	1054.05	2	2.9	1055.01	1	2.8	1055.01	3	4.0	1054.98	2	4.8
M-n121-k7	1034	1032.47	9	75.3	1032.60	30	3053.7	1032.86	26	4319.3	1032.87	21	3918.9	1032.88	25	3723.3
M-n151-k12	1015	1000.38	4	33.2	1001.50	6	41.7	1002.21	4	40.0	1002.60	5	49.1	1002.67	4	62.8
M-n200-k16	1274	1252.50	3	62.5	1253.07	4	75.2	1253.51	5	91.6	1254.06	5	92.0	1254.08	5	143.7
M-n200-k17	1275	1255.13	6	83.6	1255.39	5	80.2	1255.76	4	77.2	1256.21	5	90.2	1256.04	7	168.5
P-n50-k8	631	617.17	1	0.4	618.03	1	0.4	618.02	1	0.4	618.03	1	0.5	618.03	1	0.5
P-n70-k10	827	814.33	2	1.2	815.28	3	1.5	815.29	2	1.3	815.29	2	1.4	815.39	1	1.4
Avg		1.10		14.54	1.02		157.41	1.00		218.70	1.00		200.75	0.99		198.64
Total	21	21			21			21			21			21		

6
A Branch-Cut-and-Price Algorithm

The most recent works proposing exact algorithms for the CVRP are based on the combination of column and cut generation. They are Baldacci, Christofides and Mingozzi (3), Pessoa et al. (47), Baldacci, Mingozzi and Roberti (4), Contardo (12) and Røpke (56) (see Section 1.1). The Branch-Cut-and-Price (BCP) algorithm proposed in this chapter contains elements from all those previous algorithms, usually enhanced and combined with new elements.

- Even if Ω only contains elementary routes and robust Rounded Capacity Cuts and Strengthened Combs are separated, the resulting bounds (like those showed in Table 5.4) still leave significant integrality gaps on many instances. In order to reduce those gaps, the algorithms in (3), (4) and (12) also use non-robust cuts, even if they increase the pricing complexity. The most important original contribution of this work is the introduction of the limited memory Subset Row Cuts (lm-SRCs). While the traditional SRCs are known to be effective, the number of separated cuts had to be very restricted in (4) and (12), due to their large impact on the pricing. The lm-SRCs are a weakening of the SRCs. This weakening can be controlled and dynamically adjusted, making the lm-SRCs as effective in improving the lower bounds as the traditional SRCs, but still much less costly in the pricing. In fact, in many instances from the literature, including quite large ones, it is possible to separate lm-SRCs to obtain bounds as good as those that would be obtained by separating all SRCs with cardinality up to 5.
- The underlying formulation used in the BCP has extended arc-load variables. This allows a particularly effective fixing of variables by reduced costs (superior to fixing in the work of Irnich et al. (31)), with direct benefits on the pricing. The formulation also allows the separation of Extended Capacity Cuts (47).
- The columns in the BCP are associated to *ng*-routes. The corresponding pricing subproblem is solved by a labeling algorithm that must also

consider the dual variables of the lm-SRCs. Its implementation is quite critical for the overall BCP performance. After experiments with a number of alternatives, the best performance was obtained by a bidirectional search that differs a little from the proposed by Righini and Salani (53) because the concatenation of the labels is not necessarily performed at the half of the capacity. Completion bounds are also used for eliminating labels. Anyway, the exact pricing algorithm is called just a few times per BCP node, most of the iterations use effective heuristics. A column generation stabilization by dual smoothing (48) may be also employed.

– Like in (47), the BCP hybridizes branching with route enumeration. Actually, it performs an aggressive hierarchical strong branching, with up to n candidates (partially) evaluated in the root node. The strong branching effort in each node depends on an estimate of the size of the subtree rooted in that node. The branching mechanism also keeps the history of candidate evaluations for helping on future decisions.

– As soon as the gap of a BCP node is sufficiently small, the elementary routes that can be part of the optimal solution are enumerated into a pool. From that point, since the pricing will be performed by inspection, all lm-SRCs may be immediately lifted to SRCs and additional non-robust cuts, including cliques, may be separated. On larger instances this may not be enough to reduce the number of routes to a size tractable by a MIP solver. In those cases, the standard strong branching continues to be done over the enumerated node.

– The lm-SRCs are still non-robust. There are cases where several hundreds such cuts are being normally handled by the pricing algorithm, and then, the separation of a few dozen additional lm-SRCs makes this algorithm 100 times slower. In this situation the BCP performs a rollback, the offending cuts are removed even if it decreases the lower bound of the node.

This chapter is organized as follows. Section 6.1 presents the new formulation. Section 6.2 shows how the reduced cost of an arc-load is defined. Section 6.3 reviews the non-robust cuts that have been used in the literature and describes the Limited-memory SRCs. Finally, Section 6.4 presents other important elements of our BCP, such as the variable fixing, non-robustness control, route enumeration end strong branching. All labeling algorithms for pricing, variable fixing and route enumeration are presented in Chapter 7.

6.1 Formulation

The Arc-Load indexed Formulation (ALF) presented in Section 3.3 can be naturally rewritten in terms of q-routes. Let Ω be the set of all q-routes. For each $r \in \Omega$ define a non-negative variable λ_r and binary coefficients a_{rq}^{ij}, for each $(i,j)^q \in A_Q$, indicating whether (i,j) is traversed with load q in route r. Equations (3.22) in ALF can be replaced by:

$$\sum_{r\in\Omega} a_{rq}^{ij}\lambda_r = x_{ij}^q, \quad \forall (i,j)^q \in A_Q. \tag{6.1}$$

Substituting the x variables and relaxing the integrality, the Dantzig-Wolfe Master LP is written as:

$$\text{(DWM)} \quad \min \quad \sum_{r\in\Omega}\left(\sum_{(i,j)^q\in A_Q} a_{rq}^{ij}c_{ij}\right)\lambda_r \tag{6.2}$$

subject to

$$\sum_{r\in\Omega}\left(\sum_{(i,j)^q\in\delta^+(i)} a_{rq}^{ij}\right)\lambda_r = 1, \qquad \forall i \in V_+, \tag{6.3}$$

$$\sum_{r\in\Omega}\left(\sum_{(i,j)^q\in\delta^+(0)} a_{rq}^{ij}\right)\lambda_r = K, \tag{6.4}$$

$$\lambda_r \geq 0 \qquad \forall r \in \Omega. \tag{6.5}$$

6.2 Column Generation

A generic constraint l of format $\sum_{(i,j)^q\in A_Q} \alpha_{ij}^{lq}x_{ij}^q \geq b_l$ can also be included in the DWM, using the variable substitution (6.1), as $\sum_{r\in\Omega}(\sum_{(i,j)^q\in A_Q} \alpha_{ij}^{lq}a_{ij}^{rq})\lambda_r \geq b_l$. Suppose that, at a given instant, there are n_R constraints over the x variables in the DWM, including equalities (6.4) and (6.3). Constraint (6.4) has the dual variable π_0, the constraint in (6.3) corresponding to $i \in V_+$ has the dual variable π_i, and each additional constraint l, $n < l < n_R$, has the dual variable π_l. The reduced cost of an arc $(i,j)^q$ is defined as:

$$\bar{c}_{ij}^q = c_{ij} - \sum_{l=0}^{n_R-1} \alpha_{ij}^{lq}\pi_l. \tag{6.6}$$

The pricing subproblem for solving the DWM consists in finding a shortest path in $\mathcal{N}$ (defined in Section 3.3) from node $(0,0)$ to nodes $(0,q)$, $1 \leq q \leq Q$, with respect to the current arc reduced costs $\bar{c}^q_{ij}$. This can be done in $O(n^2Q)$ time.

Instead of using q-routes, Ω is redefined to be a set of ng-routes. Section 7.1 describes in detail our pricing algorithms for the BCP.

6.3 Cut Generation

Our BCP includes cuts defined over the three formulations described in Chapter 3. Cuts for the TIF can be included in the DWM by using the transformation $x_{ij} = \sum_{(i,j)^q \in A_Q} x^q_{ij} + \sum_{(j,i)^q \in A_Q} x^q_{ji}$. In this work, Rounded Capacity Cuts and Strengthened Combs are separated by the CVRPSEP package (37). The Extended Capacity Cuts (47), defined directly over the arc-load variables are also used. All those cuts are robust, the effect of their dual variables is captured in the arc-load reduced costs (6.6).

In addition, the BCP also separates non-robust limited-memory SRCs. Before presenting these new cuts, Section 6.3.1 reviews some important non-robust cuts that have been used in recent works, including the traditional SRCs.

6.3.1 Non-Robust Cuts Review

Strengthened Capacity Cuts

Baldacci et al. (3) introduced a family of cuts defined over the variables of the SPF. For each $S \subseteq V_+$ and for each $r \in \Omega$ (possibly containing non-elementary routes), define binary coefficient ζ^r_S as 1 iff the route r visits at least one vertex in S. The Strengthened Capacity Cuts (SCCs) are:

$$\sum_{r \in \Omega} \zeta^r_S \lambda_r \geq k(S) \qquad \forall S \subseteq V_+. \tag{6.7}$$

Remark that an RCC over S corresponds to $\sum_{r \in \Omega} a^r_S \lambda_r \geq k(S)$, where a^r_S counts how many times the route r enters (or leaves) S. Inequalities (6.7) are stronger because they are not "fooled" by routes that enter and leave S more than once. On the other hand, SCCs are non-robust, since they change the pricing subproblem for the SPF linear relaxation. In order to continue solving it by dynamic programming, it is necessary include an additional binary dimension in each label indicating whether a partial route

have already visited S or not. Notwithstanding this fact, the algorithms in (3, 4, 12) have successfully used (in a controlled way, limiting the number of separated cuts) SCCs. It can also be observed that if there exists a set $S' \subset S$ where $k(S') = k(S)$, then the SCC corresponding to S' dominates the SCC corresponding to S. This means that only the cuts corresponding to minimal sets (with respect to the function $k(\cdot)$) need to be separated.

Strong Degree Cuts

Contardo et al. (13) introduced a family of cuts that correspond to SCCs over sets S of cardinality 1. Given a vertex $i \in V_+$ and for $r \in \Omega$, define binary coefficient ζ_i^r as 1 iff route r visits i. The Strong Degree Cut (SDC) is:

$$\sum_{r \in \Omega} \zeta_i^r \lambda_r \geq 1. \tag{6.8}$$

An SDC can only be non-redundant if the definition of set Ω allows non-elementary routes. In this case, the effect of SDCs is forbidding routes with cycles at certain vertices. Therefore, they are alternative ways of enforcing partial route elementarity. As happens with the more general SCCs, each SDC requires one additional binary dimension in the dynamic programming labels.

Contardo (12) also use weakened versions of the strong degree cuts. For an integer k, $i \in V_+$ and for $r \in \Omega$, define ν_i^{rk} as the number of times that route r visits i with at least k vertices between two consecutive visits. The corresponding k-Cycle Elimination Cut (k-CEC) is:

$$\sum_{r \in \Omega} \nu_i^{rk} \lambda_r \geq 1. \tag{6.9}$$

The effect of this cut is forbidding routes that have cycles over i of length k or less. The k-CECs have less impact in the dynamic programming pricing than the SDCs. The information that a vertex i was already visited is "forgot" after k visits to other vertices.

Subset Row Inequalities

Jepsen et al. (33) introduced the following family of cuts defined over the variables of the SPF. Given a base set $C \subseteq V_+$ and a multiplier p, $0 < p < 1$, the following (C, p)-Subset Row Cut (SRC)

$$\sum_{r \in \Omega} \left\lfloor p \sum_{i \in C} a_i^r \right\rfloor \lambda_r \leq \lfloor p|C| \rfloor \tag{6.10}$$

is valid, since it can be obtained by a Chvátal-Gomory rounding of the corresponding constraints in (3.8).

Note that any SRC is a Chvátal-Gomory Rank-1 cut, since they are obtained by multiplying the rows of set C by p followed by a Chvátal-Gomory rounding. The cuts where $|C| = 3$ and $p = 1/2$ are called 3-Subset Row Cuts (3SRCs). The earlier algorithm by Baldacci et al. (3) used clique cuts, that are more general than 3SRCs. However, 3SRCs are favored in more recent works (4, 12), since they seem to be more suited to a column generation context, having less impact in the pricing subproblem. Each 3SRC requires an additional binary dimension in each dynamic programming label to indicate the parity of the number of visits made by a route to a vertex in a triplet C. The successful use of those cuts depends on a carefully controlled separation, in order to avoid pricing intractability.

Baldacci et al. (4) have also used a weakened variant of the 3SRCs for the case where Ω only contains elementary routes. Define a binary coefficient ξ_C^r as 1 iff the route r visits at least one edge with both endpoints in C. The corresponding weakened 3SRC is:

$$\sum_{r \in \Omega} \xi_C^r \lambda_r \leq 1. \tag{6.11}$$

Since an elementary route can only visit two such edges if they are consecutive, it is easy to include the effect of the dual variables of weakened 3SRCs in a dynamic programming pricing. Therefore, there are no restrictions to its separation. A variant of this cut suitable to the case where Ω contains non-elementary routes would define ξ_C^r as the number of times that a route r visits non-consecutive edges with both endpoints in C.

6.3.2 limited-memory SRCs

The definition of the limited memory (C, M, p)-Subset Row Cut (lm-SRC) requires an additional set M, $C \subseteq M \subseteq V_+$. It can be written as:

$$\sum_{r \in \Omega} \alpha(C, M, p, r) \lambda_r \leq \lfloor p|C| \rfloor, \tag{6.12}$$

where the coefficients α are a function of C, M, p and r computed by the Algorithm 6.1.

When $M = V_+$, the Function α will return $\lfloor p \sum_{i \in C} a_i^r \rfloor$ and the lm-SRC will be identical to an SRC. On the other hand, when M is not equal to V_+, the lm-SRC may be a weakening of its corresponding SRC. This happens because

Algorithm 6.1 Procedure that calculates the coefficient of a route r in a lm-SRC

```
1: function α(C, M, p, r)
2: coeff ← 0, state ← 0
3: for every vertex i ∈ r (in order) do
4:     if i ∉ M then
5:         state ← 0
6:     else if i ∈ C then
7:         state ← state + p
8:         if state ≥ 1 then
9:             coeff ← coeff + 1, state ← state − 1
10: return coeff
```

every time the route r leaves M, the variable *state* in the function is set to zero, potentially decreasing the returned coefficient. Function α indicates how the lm-SRCs should be taken into account in the labeling algorithms used in the pricing. In fact, that procedural function is executed along the algorithm. Each label should have an additional dimension for each lm-SRC, storing their states in the corresponding partial paths. However, the coefficients do not need to be stored in the labels. Instead, whenever a label extension causes the increment of the coefficient of an lm-SRC, according to Function α, the value of its dual variable is immediately subtracted from the reduced cost of the new label. We remark that the number of possible states of an lm-SRC depends on its p. For example, for the frequent case where $p = 1/2$, the state can be only 0 or $1/2$. Therefore, it can be represented by a single bit.

The potential advantage of the lm-SRCs over classical SRCs is their much reduced impact on the labeling algorithm used in the pricing, when $|M| << |V_+|$. The reasons for that reduction will be explained in Section 7.1.1. In order to obtain small memory sets, we propose the following separation strategy for the lm-SRCs. First, it identifies a violated (C, p)-SRC. Then, it determines a *minimal set M such that the lm-(C, M, p)-SRC has the same violation*. In practice, even on instances with hundreds of customers, those minimal sets seldom have cardinality larger than 15. For example, suppose that, in a given fractional solution, the paths that visit $C = \{1, 2, 3\}$ at least twice are: $r_1 = (0 - 1 - 4 - 5 - 3 - 6 - 2 - 7 - 1 - 0)$ with $\lambda_{r_1} = 0.2$, $r_2 = (0-7-2-8-3-0)$ with $\lambda_{r_2} = 0.3$, and $r_3 = (0-5-3-4-1-7-9-2-0)$ with value $\lambda_{r_3} = 0.4$. The $(C,1/2)$-SRC $2\lambda_{r_1} + \lambda_{r_2} + \lambda_{r_3} \leq 1$ has a violation of 0.1. A minimal set M that yields a lm-$(C,M,1/2)$-SRC with the same violation can be $M = C \cup \{4, 5, 7\} \cup \{8\} \cup \{4\}$.

Given a base set $C \subseteq V_+$, for each integer d, $1 \leq d \leq n$, define a non-negative integer variable y_C^d as the sum of all variables λ_r such that

$\sum_{i \in C} a_i^r = d$. Variables with $d > |C|$ can only be non-zero if Ω contains non-elementary routes. The interesting SRCs, for sets C with cardinality up to 5, are the following:

- 3-Subset Row Cuts (3SRCs) which can be expressed as $y_C^2 + y_C^3 + 2y_C^4 + 2y_C^5 + \ldots \leq 1$. Although they are very effective in improving the lower bounds, only a relatively small number of those cuts could be separated in (4, 12), in order to keep the pricing tractable. The Weak 3SRCs of (4) are equivalent (when all routes are elementary) to lm-3SRCs with $M = C$. Since $p = 1/2$, the additional dimension required by lm-3SRCs (and 3SRCs) in the labeling algorithm is a bit that can be interpreted as follows: 0 means that an even (perhaps zero) number of visits to vertices in C is remembered, 1 means an odd number of visits. The value of the dual variable corresponding to such cut is subtracted from a label ending in a vertex in C when the bit goes from 1 to 0.
- Taking $|C| = 1$ and $p = 1/2$, the 1-Subset Row Cuts (1SRCs) $y_C^2 + y_C^3 + 2y_C^4 + \ldots \leq 0$ are obtained. They are equivalent to the Strong Degree Cuts $y_C^1 \geq 1$ introduced in (12), in the sense that both families forbid cycles over a vertex i ($C = \{i\}$). Contardo also defined the weaker k-Cycle Elimination Cut that only forbid cycles over i of size k or less. A lm-1SRC is a different kind of weakening, it forbids cycles over i contained in the set M. Of course, all these cuts can only be useful when the Ω set contains non-elementary routes.
- The cuts where $|C| = 4$ and $p = 2/3$ are 4SRCs, expressed as $y_C^2 + 2y_C^3 + 2y_C^4 + 3y_C^5 + 4y_C^6 + \ldots \ldots \leq 2$. This value of p may only produce states 0, 2/3, and 1/3 in Function α. Therefore, the state of a 4-SRC can be represented by a ternary digit.
- There are two interesting families of cuts with $|C| = 5$. Those with $p = 1/3$ will be called 5,1SRCs, $y_C^3 + y_C^4 + y_C^5 + 2y_C^6 \ldots \leq 1$; whereas those with $p = 1/2$ are 5,2SRCs, having the format $y_C^2 + y_C^3 + 2y_C^4 + 2y_C^5 + 3y_C^6 \ldots \leq 2$. The latter family was already used in (12).

6.4 Other Elements

6.4.1 Variable Fixing by Reduced Costs

The variable fixing procedure aims to eliminate x_{ij}^d variables by proving that they cannot assume positive values in any integral solution that improves upon the current best known integral solution. Consider an optimal solution of DWM and let z_{LB} be the value of this solution. Let $\bar{C}_{ij}^q$ be the minimum reduced cost route passing by arc $(i,j)^q \in A_Q$. Let z_{UP} be a valid upper bound for the problem. Variable x_{ij}^d can be fixed to zero if the following condition holds:

$$z_{LB} + \bar{C}_{ij}^q \geq z_{UB}. \tag{6.13}$$

For instances with integer costs, we can strengthen 6.13 using the stronger quantity $z_{UB} - 1 + \epsilon$ (where $\epsilon > 0$ is a small tolerance) instead of z_{UB}. In other words, if $\bar{C}_{ij}^q$ is larger than the integrality gap, given by $z_{UB} - z_{LB}$ (or $z_{UB} - 1 + \epsilon - z_{LB}$, for integer costs) then x_{ij}^d can be fixed to zero.

A similar procedure was also proposed by Irnich et al. (31), but it is weaker because it only removes an arc $(i,j) \in A$, if a single particular solution allows removing $(i,j)^q$ for all values of q. On the other hand, as our BCP already works on the arc-load formulation, individual arcs $(i,j)^q$ can be naturally removed. For instance, it is quite typical that, at a certain point of the BCP, 95% of the arc-load variables were already fixed to zero, while the fixing on arcs would not achieve 80%.

Variable fixing have a great potential to reduce the number of labels to be treated by the exact pricing (which is the most critical element of the BCP) at subsequent iterations. This happens because many fruitless labels (i.e., those that would only lead to routes that can not belong to an optimal solution) are not created during the pricing due to the fact that some arcs are fixed. As expected, deeper nodes of the BCP will have smaller gaps, then the fixing will be more effective on these nodes. Since half of the nodes in a search tree are leaves, the overall gain can be substantial. As pointed in (49), there are other potential benefits for the BCP. For instance, the column generation convergence can be improved because the number of routes (columns) is reduced. Furthermore, as some of the routes discarded could be part of the optimal solution of the linear relaxation, the desirable effect of an improved lower bound can be observed.

Section 7.2 shows how our pricing algorithms are used to fix variables by reduced costs.

6.4.2 Route Enumeration

Baldacci et al. (3) introduced a route enumeration based approach in order to close the duality gap after the root node. An elementary route can only be part of a solution that improves the best known upper bound if its reduced cost is smaller than the gap. The enumeration of elementary routes may be performed by a label setting algorithm, producing a set partitioning problem that is given to a general MIP solver. This may work very well if the size of the set R of enumerated routes is not too large. If $|R| < 20,000$ the set partitioning is usually solved in less than 1 minute in a modern machine. Larger values of $|R|$ may cause the MIP solver to take too much time, $|R| > 100,000$ is often unpractical.

Contardo (12) proposed another strategy in order to better profit from the route enumeration. The enumeration is performed even if the resulting R has a few million routes, which are stored in a pool. The column and cut generation proceeds. However, instead of using a labeling algorithm, the pricing starts to be performed by inspection in the pool. Now, the non-robust cuts have little impact in the pricing complexity. Therefore, they may be separated in a very aggressive way. This usually increases substantially the lower bounds, allowing reductions in the pool size by fixing variables (that now are routes) by reduced costs. For example, he reported that the enumeration of instance M-n151-k12 with gap of 5 units produced a pool with 4M routes. The separation of non-robust cuts then reduced the gap to 1.5 and the final pool only had 13K routes. The resulting set partitioning was easily solved, finishing the instance.

In this work we used Contardo-style enumeration, allowing pools with up to 50M routes. After enumeration, SRCs and also clique cuts (separated using the routines from (57)) are added. Anyway, since the enumeration of so many routes may be very time-consuming, the implementation of the labeling algorithms used for that purpose becomes critical. Section 7.3 presents our algorithms for elementary route enumeration.

6.4.3 Non-Robustness Control

Even with all the previously described enhancements, the addition of too many lm-SRCs will still cause an exponential explosion of the number of labels in the pricing algorithms. Worse, it is not possible to predict when the

explosion will occur. We have seen examples of runs where several hundreds such cuts are being normally handled by the pricing algorithm and then, at some node deep in the tree, the separation of a few dozen additional lm-SRCs makes this algorithm 100 times or even 1000 times slower. In some cases the BCP crashed due to memory overflow.

The first strategy tried for avoiding such failures was setting a conservative limit on nS and stopping the separation of lm-SRCs after it is reached. A much better strategy is to handle the lm-SRCs dynamically. In the beginning of the root node, when no lm-SRCs were added, the pricing still has a pseudo-polynomial complexity (assuming that the size of the ng-sets is fixed). Those robust runs of the pricing are used to establish a baseline BL on the number of labels. The algorithm proceeds by separating rounds of lm-SRCs. The number of labels in the pricing is likely to increase, values up to $5BL$ are always acceptable, larger values may be tolerated if the rate of lower bound improvement is still good. This mechanism determines, for each node, when separation will stop and enumeration/branching will be called. However, if the number of labels in a call of the pricing exceeds $50BL$, the BCP concludes that an exponential explosion is occurring. The pricing is aborted and the node is rolled back to its previous state before the last round of cuts. This means that the lm-SRCs added in that round, even if they are active, are removed.

6.4.4 Strong Branching

It is clear that route enumeration is an important element in some of the best performing algorithms for the CVRP, being able of drastically reducing the running times of some instances. Nevertheless, it is disturbing to consider that route enumeration, no matter how cleverly done, is an inherently exponential space procedure (it would remain exponential even if P=NP) that is bound to fail on larger/harder instances. However, one does not need to take a radical stance on completely avoiding branching. The hybrid strategy used in (47) and (50) performs route enumeration after solving each node. If a limit on the number of routes is reached, the enumeration is aborted and the BCP proceeds by traditional branching. Of course, since deeper nodes will have smaller gaps, at some point the enumeration will work.

The branching in our BCP is performed over sets $S \subseteq V_+$, imposing the disjunction $(\sum_{a\in\delta^-(S)} x_a = 1)\vee(\sum_{a\in\delta^-(S)} x_a \geq 2)$. Those branching constraints are robust and can translated into arc reduced cuts for the pricing in both child nodes. The BCP proposed by Fukasawa et al. (24) adopted a similar branching over sets (already used in (38)). In order to better choose the branching set,

that BCP used strong branching. Each set in a collection containing from 5 to 10 candidate sets was heuristically evaluated by applying a small number of column generation iterations in its children nodes. Remark that the exact evaluation of each candidate, performing full column generation (perhaps also cut generation) in both children nodes, would be too expensive.

The recent work by Røpke (56) showed that is possible to obtain major improvements in a BCP performance by performing a more sophisticated and aggressive strong branching:

- The simpler branching over individual arcs was used.
- The procedure starts by performing a quick evaluation of 30 candidate arcs, producing a ranking. The best ranked candidate is then fully evaluated and becomes the incumbent winner. Then, other candidates with good ranking are better evaluated, but only while they have a reasonable chance of beating the incumbent winner.
- It is possible to collect statistics along the enumeration tree to help on choosing the candidates. The previous evaluations of an arc are good predictors of future evaluations.

His extensive experiments were performed both on CVRP and VRPTW instances. This strong branching was the key algorithmic element that allowed his BCP to solve the hard instance M-n151-k12, with optimum 1015, starting from the rather modest root lower bound of 1001.5. This BCP was the first algorithm to solve all the 56 VRPTW Solomon instances with 100 customers.

Inspired by those good results, we devised a hierarchical strong branching procedure for our BCP:

- The Phase Zero performs the first selection of candidate sets. In the root node, this is a collection of $\min\{n, 300\}$ sets, chosen by the proximity of $\sum_{a \in \delta^-(S)} \bar{x}_a$ to 1.5 in the current fractional solution. For a non-root node v, the collection has cardinality between 50 and 10, depending on $TS(v)$, the estimative of the size of the subtree rooted in v. Half of the candidates are chosen based on the history of previous calls to the strong branching procedure. In contrast, the choice of the remaining candidates favors fresh sets, that were never evaluated before.
- The Phase One performs a rough evaluation of each candidate by solving the current restricted Master LP twice, adding the constraint corresponding to each child node. Column and cut generation are not performed. The resulting improvements in the lower bounds are usually overvaluations of the true improvements if that candidate is selected. The

candidates are ranked by the product rule (1) and the best candidates (between 20 and 3) candidates go to Phase Two. If $TS(v)$ is very small, Phase Two is skipped and the branching is performed with the best candidate of Phase One.

– Phase Two performs quite precise evaluations. Column and cut generation are performed, the only difference to the standard solving is that only heuristic pricing is used. Actually, if $TS(v)$ large, even the exact pricing may be called. The results of Phase Two are not only used to select the candidate to perform the current branching, they are stored in tables (the branching history) for subsequent use in the Phase Zero.

The whole procedure is guided by the principle that the strong branching effort in a node should depend on the expected subtree size. The logic behind this is the following. If $TS(v)$ is large, even a small improvement in that branching will pay the computational cost of the precise evaluation of several candidates. On the other hand, if $TS(v)$ is small, the branching should be fast, relying on the historical data and on the rough evaluations of Phase One. The estimation $TS(v)$ is calculated from the node gap (the upper bound minus the lower bound of v) and from the values of Δ_l and Δ_r, the average historical lower bound improvement in a left child (corresponding to a constraint $= 1$) and in a right child (constraint ≥ 2). It is typical that Δ_l is larger than Δ_r, reflecting a quite unbalanced tree. In the first nodes, the evaluations of the candidates performed by strong branching itself are used as proxies for Δ_l and Δ_r.

Combining Strong Branching and Route Enumeration

As mentioned before, our BCP uses a hybrid strategy. Enumeration is tried after solving each node. However, the strong branching can still be performed after a successful enumeration. This happens when the final set of routes R is too large (we use 20,000 as the limit) for the MIP solver.

7
Labeling Algorithms

This chapter presents the algorithms used by our BCP for pricing ng-routes, for fixing arc-load variables by reduced costs, and for enumerating elementary routes. Those algorithms should take into account the following points:

1. the reduced cost of an arc, defined as (6.6), depends on its load q;
2. some variables x_a^q may be already fixed to zero; and
3. non-robust lm-SRCs may be present.

7.1
Pricing Algorithms

7.1.1
Forward Labeling

The forward dynamic programming labeling algorithm for the pricing problem represents an ng-feasible partial forward path $P = (0, \ldots, i)$, $i \in V$, as a *label* $L(P) = (\bar{c}(P), v(P) = i, q(P), \Pi(P), S(P), pred(P))$ storing its reduced cost, end vertex, load, set of vertices forbidden as immediate extensions due to ng-sets, vector of states corresponding to the nS lm-SRCs with non-zero dual variables in the current Master LP solution, and a pointer to its predecessor label. Each $(i, q) \in V_Q$ defines a *bucket* $F(i, q)$. A label $L(P)$ is stored in bucket $F(v(P), q(P))$. A label $L(P_1)$ dominates a label $L(P_2)$ if every feasible completion of P_2 yields a route with reduced cost not smaller than the feasible route obtained by applying the same completion into P_1. Sufficient conditions for that are:

$$(i)\ v(P_1) = v(P_2), \quad (ii)\ q(P_1) = q(P_2), \quad (iii)\ \Pi(P_1) \subseteq \Pi(P_2), \text{ and}$$

$$(iv)\ \bar{c}(P_1) \leq \bar{c}(P_2) + \sum_{1 \leq s \leq nS : S(P_1)[s] > S(P_2)[s]} \sigma_s,$$

where $\sigma_s < 0$ is the dual variable associated to lm-SRC s. Remark that the reduced costs depending on q or the fixing of some x_a^q variables to zero

prevent (ii) to be strengthened to $q(P_1) \leq q(P_2)$. This happens because, if $q(P_1) \neq q(P_2)$, the reduced cost of a completion for P_2 may differ from the reduced cost of the same completion for P_1. In fact, due to the fixing, a feasible completion for P_2 may be infeasible for P_1. Note also that the second term in the right-hand side of (iv) is an upper bound on what a completion of P_2 can gain over the same completion of P_1, by avoiding the penalizations of revisiting the lm-SRCs s in which $S(P_1)[s] > S(P_2)[s]$. Only non-dominated labels are kept in the buckets. To accelerate the checking for dominated labels, it is convenient to keep labels of the same bucket ordered by reduced cost, since $L(P_1)$ dominates $L(P_2)$ only if $\bar{c}(P_1) \leq \bar{c}(P_2)$. The base set, multiplier and memory set of a lm-SRC s are denoted by $C(s)$, $p(s)$, and $M(s)$, respectively. Consider $NG(0)$ as $\{0\}$. Algorithm 7.1 presents the pseudocode of the Forward Labeling procedure. In the end of the algorithm, each non-

Algorithm 7.1 Forward Dynamic Programming Labeling

```
1: procedure FORWARD LABELING
2:     F(i,q) ← ∅, ∀(i,q) ∈ V_Q
3:     F(0,0) ← {(0,0,0,∅,0,nil)}, nFL ← 1
4:     for q = 1 to Q do
5:         for all i such that (i,q) ∈ V_Q do          ▷ Process buckets F with load q
6:             for all j such that (i,j)^q ∈ A_Q and x_ij^q is not fixed to 0 do
7:                 for all L_1 = (c̄_1, i, q, Π_1, S_1, _) ∈ F(i,q) do
8:                     if j ∉ Π_1 then
9:                         c̄_2 ← c̄_1 + c̄_ij^q , S_2 ← S_1
10:                        for s = 1 to nS do
11:                            if j ∉ M(s) then S_2[s] ← 0
12:                            else if j ∈ C(s) then
13:                                S_2[s] ← S_2[s] + p(s)
14:                                if S_2[s] ≥ 1 then c̄_2 ← c̄_2 − σ_s , S_2[s] ← S_2[s] − 1
15:                        L_2 ← (c̄_2, j, q + d_j, (Π_1 ∩ NG(j)) ∪ {j}, S_2, pointer to L_1)
16:                        insert ← true
17:                        for all L ∈ F(j, q + d_j) do
18:                            if L_2 dominates L then delete L, nFL ← nFL − 1
19:                            else if L dominates L_2 then insert ← false, break
20:                        if insert then
21:                            F(j, q + d_j) ← F(j, q + d_j) ∪ {L_2}, nFL ← nFL + 1
```

empty bucket $F(0,q)$, $1 \leq q \leq Q$, will contain only one label, representing the minimum reduced cost route with load q.

Algorithm 7.1 assumes that the dominance between labels is transitive, i.e., given labels L_1, L_2 and L_3, if L_1 dominates L_2 and L_2 dominates L_3, then L_1 dominates L_3. Transitivity is not obvious due to condition (iv). We prove this property because the algorithm removes (in line 18) a label $\mathcal{L}$ dominated by a label L_2, without knowing whether L_2 will not be latter removed by being dominated by another label L_3.

Proposition 2 *If $L(P_1)$ dominates $L(P_2)$ and $L(P_2)$ dominates $L(P_3)$, then $L(P_1)$ dominates $L(P_3)$.*

Proof 2 *Conditions (i), (ii) and (iii) are obviously transitive. Consider condition (iv), if $L(P_1)$ dominates $L(P_2)$ and $L(P_2)$ dominates $L(P_3)$, then $\bar{c}(P_1) \leq \bar{c}(P_2) + \sum_{s \in D_{1,2}} \sigma_s$, where $D_{1,2} = \{s \mid 1 \leq s \leq nS, S(P_1)[s] > S(P_2)[s]\}$, and $\bar{c}(P_2) \leq \bar{c}(P_3) + \sum_{s \in D_{2,3}} \sigma_s$, where $D_{2,3} = \{s \mid 1 \leq s \leq nS, S(P_2)[s] > S(P_3)[s]\}$. Therefore, $\bar{c}(P_1) \leq (\bar{c}(P_3) + \sum_{s \in D_{2,3}} \sigma_s) + \sum_{s \in D_{1,2}} \sigma_s \leq \bar{c}(P_3) + \sum_{s \in (D_{1,2} \cup D_{2,3})} \sigma_s$.*

If $s \in D_{1,3} = \{s \mid 1 \leq s \leq nS, S(P_1)[s] > S(P_3)[s]\}$ then $S(P_3)[s] < S(P_1)[s]$, which implies that $S(P_2)[s] < S(P_1)[s]$ or $S(P_2)[s] > S(P_3)[s]$. In other words, if $s \in D_{1,3}$ then $s \in (D_{1,2} \cup D_{2,3})$. Therefore, $D_{1,3} \subseteq (D_{1,2} \cup D_{2,3})$. Since the dual variables σ_s are negative, we can state that $\bar{c}(P_1) \leq (\bar{c}(P_3) + \sum_{s \in D_{2,3}} \sigma_s) + \sum_{s \in D_{1,2}} \sigma_s \leq \bar{c}(P_3) + \sum_{s \in (D_{1,2} \cup D_{2,3})} \sigma_s \leq \bar{c}(P_3) + \sum_{s \in D_{1,3}} \sigma_s$, which proofs that $L(P_1)$ dominates $L(P_3)$.

Now, it is possible to explain why the lm-SRCs have a reduced impact in the pricing when their memory sets are small. There are $O(nQ)$ buckets in the labeling algorithm. In a rough analysis, the time spent processing each bucket grows quadratically with the average number of non-dominated labels in it and linearly with nS.

- A first observation is that the state of an lm-SRC s needs only to be explicitly present in labels corresponding to partial paths ending in $M(s)$. For the remaining labels, this state is zero by definition, which means that σ_s will not play any role in the processing of their buckets. In other words, only the vertices in $M(s)$ need to know about the existence of that cut. This allows the acceleration of the algorithm by a factor of $\theta(n/M_{avg})$, where M_{avg} is the average size of the memory sets.
- However, the crucial point for the improved algorithm efficiency is related to the dominance. If there are no SRCs, the maximum number of non-dominated labels in a bucket $F(i, q)$ is bounded by $2^{|NG(i)|-1}$, as follows from dominance conditions (iii) and (iv). If the cardinality of the ng-sets is small (we used 8 in this work), the pricing is guaranteed to be reasonably fast (unless Q is very large). However, if a traditional SRC is added, its dual variable may make condition (iv) *weaker in all buckets*. As other SRCs are separated, this may quickly result in an exponential proliferation of non-dominated labels. In practice, this severely limits the number of SRCs that can be used. In contrast, a lm-SRC s with a small memory has much less impact because *it can only weaken the*

dominance in the buckets of $M(s)$. In practice, many more lm-SRCs can be separated before the exponential proliferation of labels is observed. This is exemplified in Figure 7.1. Let P_1 be the solid path and P_2 the dashed one, both paths ending in vertex i and having load q. A 3SRC with base set $C = \{1, 2, 3\}$ may prevent $L(P_1)$ from dominating $L(P_2)$, even though no good completion of P_2 visits C. On the other hand, a lm-3SRC over the same C, having the memory set represented in the figure by filled circles, would not interfere with that dominance.

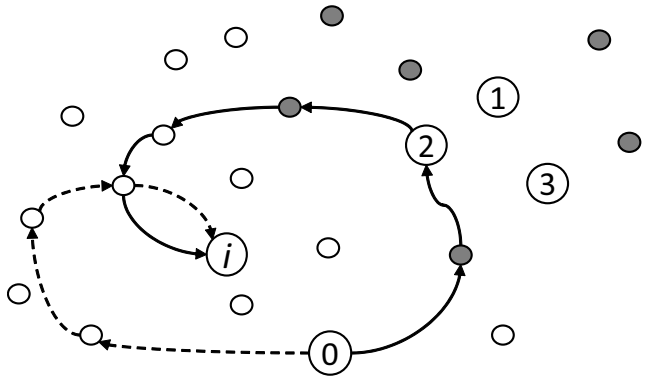

Figure 7.1: Example illustrating the performance gain in the pricing when using lm-SRCs.

7.1.2 Bidirectional Labeling

The labeling algorithm for the pricing can also be performed backwards. In that case, the labels represent ng-feasible partial backward paths $P = (i, \ldots, 0)$, $i \in V_+$. The initializing labels are put in buckets $B(0, q)$, $1 \leq q \leq Q$, and the algorithm proceeds in a reversed way, until the label corresponding to the route with minimum reduced cost is found in bucket $B(0, 0)$, as shown in Algorithm 7.2:

The forward and backward variants of the labeling are equivalent in terms of computational cost. However, as pointed by Righini and Salani (53), when forward labeling is used, most of the computational effort is spent in buckets with larger values of q, close to Q. This happens by combinatorial reasons, there are many more possible paths converging into a bucket $F(i, q)$ if q is larger. In a similar way, when backward labeling is used, most of the computational effort is spent in buckets with small values of q. Therefore, it is often advantageous to perform bidirectional search: use the forward labeling for filling the buckets $F(i, q)$ with $q \leq Q/2$ and backward labeling for filling the buckets $B(i, q)$ with $q > Q/2$. The minimum reduced cost paths can obtained by an additional concatenation step. After implementing this bidirectional

Algorithm 7.2 Backward Dynamic Programming Labeling

```
1: procedure BACKWARD LABELING
2:     B(i,q) ← ∅, ∀(i,q) ∈ V_Q, nBL ← 0
3:     for q = Q to 1 do
4:         B(0,q) ← {(0,0,q,∅,0,nil)}, nBL ← nBL + 1
5:         for all j such that (j,q) ∈ V_Q do        ▷ Process buckets B with load q
6:             for all i such that (i,j)^{q-d_j} ∈ A_Q and x_ij^{q-d_j} is not fixed to 0 do
7:                 for all L_1 = (c̄_1, j, q, Π_1, S_1, _) ∈ B(j,q) do
8:                     if i ∉ Π_1 then
9:                         c̄_2 ← c̄_1 + c̄_ij^{q-d_j} , S_2 ← S_1
10:                        for s = 1 to nS do
11:                            if i ∉ M(s) then S_2[s] ← 0
12:                            else if i ∈ C(s) then
13:                                S_2[s] ← S_2[s] + p(s)
14:                                if S_2[s] ≥ 1 then c̄_2 ← c̄_2 − σ_s , S_2[s] ← S_2[s] − 1
15:                        L_2 ← (c̄_2, i, q − d_j, (Π_1 ∩ NG(i)) ∪ {i}, S_2, pointer to L_1)
16:                        insert ← true
17:                        for all L ∈ B(i, q − d_j) do
18:                            if L_2 dominates L then delete L, nBL ← nBL − 1
19:                            else if L dominates L_2 then insert ← false, break
20:                        if insert then
21:                            B(i, q − d_j) ← B(i, q − d_j) ∪ {L_2}, nBL ← nBL + 1
```

algorithm, we realized that the number of labels in the backward part was consistently larger (3 to 10 times more is typical) than in the forward part. This happens because the backward labeling has more starting labels. Therefore, the algorithm performance can be improved by better balancing both parts. This means that the concatenation will occur at a value (dynamically determined) larger than $Q/2$.

Algorithm 7.3 presents the pseudocode of the Bidirectional Labeling procedure. The algorithm starts by running the Forward Labeling and Backward Labeling in an alternated way. If the current number of forward labels is smaller than the current number of backward labels, then the buckets F with load qf are processed and qf is incremented. Otherwise, the buckets B with load qb are processed and qb is decremented. The process ends when $qf = qb - 1$. The minimum reduced cost route with load $1 \leq q \leq qf$ is obtained from bucket $F(0, q)$, which contains at most one label. However, routes with load $q > qf$ must be obtained from the concatenation of forward and backward labels, which is a potentially costly operation. We use the fact that the labels are sorted in increasing reduced cost order (IRCO) inside the buckets. Therefore, many concatenations that would not yield negative reduced cost routes are quickly discarded. This is possible because the lm-SRCs only penalize the concatenation of labels, since their duals are negative. Procedure *Save* saves

pointers to pairs of labels corresponding to the routes with negative reduced cost, for subsequent use in the column generation. Note that the backward labeling procedure must be explicitly performed by the bidirectional algorithm because the backward paths can not be derived from the forward paths, as in Section 5.4.2. This happens because the inclusion of extended cuts or the fixing of arc-load variables may cause an asymmetry on the graph where pricing is performed.

Algorithm 7.3 Bidirectional Dynamic Programming Labeling

procedure BIDIRECTIONAL LABELING
  $F(i,q) \leftarrow \emptyset, \forall (i,q) \in V_Q$
  $F(0,0) \leftarrow \{(0,0,0,\emptyset,\mathbf{0},nil)\}$, $nFL \leftarrow 1$
  $B(i,q) \leftarrow \emptyset, \forall (i,q) \in V_Q$, $nBL \leftarrow 0$
  $qf \leftarrow 1$, $qb \leftarrow Q$
  while $qb - qf > 1$ **do**
    if $nFL < nBL$ **then**
      Process buckets F with load qf, $qf \leftarrow qf + 1$
    else
      Process buckets B with load qb, $qb \leftarrow qb - 1$
  for all $(i,j)^q \in A_Q$ such that x_{ij}^q is not fixed to 0, $q \leq qf$, $q + d_j \geq qb$ **do**
    for all $L_1 = (\bar{c}_1, i, q, \Pi_1, S_1, \_) \in F(i,q)$ in IRCO **do**
      for all $L_2 = (\bar{c}_2, j, q + d_i, \Pi_2, S_2, \_) \in B(j, q + d_i)$ in IRCO **do**
        $c_r \leftarrow c_1 + \bar{c}_{ij}^q + c_2$
        if $c_r \geq 0$ **then break**
        if $\Pi_1 \cap \Pi_2 = \emptyset$ **then**
          for $s := 1, \ldots, nS$ **do**
            if $S_1[s] + S_2[s] \geq 1$ **then**
              $c_r \leftarrow c_r - \sigma_s$
          if $c_r \leq 0$ **then**
            $Save$(pointer to L_1, pointer to L_2)

7.1.3 Completion Bounds

Completion bounds can be obtained by runs of the labeling algorithms that only consider the dimensions of nS' out of the nS the active lm-SRCs. Since the effect of a lm-SRC in the pricing is penalizing the reduced cost of some routes, this corresponds to a relaxation of the original problem. However, as proposed by Contardo (12), better completion bounds can be obtained by incorporating part of the effect of the remaining $nS - nS'$ SRCs into the reduced costs:

– For a 3-SRC or a 5,2SRC s, the value $\sigma_s/2$ can be subtracted from the reduced cost of the arcs inside $C(s)$.

– For a 4-SRC s, the value $2\sigma_s/3$ can be subtracted from the reduced cost of the arcs inside $C(s)$.

However, it is not possible to transfer part of the dual variables corresponding to 1-SRCs or to 5,1SRCs into arc reduced costs.

The completion bounds for the forward labeling are obtained by first running the relaxation defined above of the backward labeling. Let $\widehat{F}(i,q)$ be the minimum reduced cost of a label in a bucket $B'(i,q)$, $(i,q) \in V_Q$, filled by that relaxation. Then, the extension of label $L(P) = (\bar{c}(P), i, q(P), \_, \_, \_)$ to a customer j in Algorithm 7.1 is avoided if the following condition holds:

$$\bar{c}(P) + \bar{c}^{q}_{ij} + \widehat{F}(j, q(P) + d_j) \geq 0. \tag{7.1}$$

Analogously, the backward labeling uses completion bounds obtained by running the relaxation of the forward labeling. The extension of label $L(P) = (\bar{c}(P), j, q(P), \_, \_, \_)$ to a customer i in Algorithm 7.2 is avoided if:

$$\bar{c}(P) + \bar{c}^{q-d_j}_{ij} + \widehat{B}(i, q(P) - d_j) \geq 0, \tag{7.2}$$

where $\widehat{B}(i,q)$ is the minimum reduced cost of a label in a bucket $F'(i,q)$, $(i,q) \in V_Q$, filled by the relaxation. Therefore, Algorithm 7.3 uses both forward and backward completion bounds.

Algorithm 7.3 can use completions bounds obtained in a way that resembles the Decremental State-Space Relaxation (DSSR) strategy (54). Since the forward labeling usually runs faster than the backward, the first iteration starts by running the forward labeling without any lm-SRCs. Next, the backward labeling can be performed considering, for example, only the lm-SRCs with the nS' most negative dual variable. This last round is accelerated with the backward completions bounds obtained from the first forward running. The second iteration starts by running the forward labeling over the $2nS'$ lm-SRCs with most negative dual variables. Analogously, this round is accelerated by the forward completions bounds obtained from the previous running of the backward labeling and so on. The remaining lm-SRCs not considered in the computation may have their dual variables underestimated as described above.

When computing completions bounds, a possible acceleration is not filling buckets with high capacity in the forward labeling, say buckets $F(i,q)$ in which $q > 0.85Q$, or buckets with low capacity in the backward labeling, say buckets $B(i,q)$ in which $q < 0.15Q$. The reason is that for the forward (resp., backward) labeling the major computational effort is spent filling buckets with high (resp., low) capacity. Furthermore, it is reasonable to expect that these bounds will not be effective, since "short paths" are rarely eliminated by completions bounds.

7.1.4 Heuristic Pricing

Even with the use of the accelerating techniques mentioned in this section, the exact pricing algorithm can still be quite time-consuming. Therefore, the column generation can be accelerated by also using faster pricing heuristics. In fact, the exact pricing may be only called when the pricing heuristics can not find routes with negative reduced cost. Effective and simple such heuristics can be obtained by modifying the label setting algorithms. The heuristic used in this work is quite similar to the heuristic described in Section 5.4.6. The only difference is that the dual variables of the lm-SRCs must be taken into account in the calculation of the reduced cost of the paths.

7.2 Variable Fixing

The labeling algorithms are also employed in a key part of the BCP: the elimination of variables by Lagrangean bounds. A full separated run of both forward and backward labeling should be performed. The minimum reduced cost of a route passing by an arc $(i,j)^q \in A_Q$ can be obtained by concatenating the labels in $F(i,q)$ from the forward run with the labels in $B(j,q+d_j)$ from the backward run. As mentioned in Section 6.4.1, if $\bar{C}^q_{ij}$ is larger than the integrality gap then x^d_{ij} can be fixed to zero. Algorithm 7.4 shows the pseudocode for variable fixing. It is quite similar to that shown in Algorithm 7.3, the Bidirectional Labeling procedure. Note that the concatenation here also relies on the fact that labels are ordered by reduced cost inside buckets.

Algorithm 7.4 Procedure Variable Fixing

```
1: Run Forward Labeling
2: Run Backward Labeling
3: for all (i,j)^q ∈ A_Q such that x^q_ij is not fixed to 0 do
4:     Fix ← true
5:     for all L_1 = (c̄_1, i, q, Π_1, S_1, _) ∈ F(i,q) in RC order do
6:         for all L_2 = (c̄_2, j, q + d_j, Π_2, S_2, _) ∈ B(j, q + d_j) in RC order do
7:             c̄_r ← c̄_1 + c̄^q_ij + c̄_2
8:             if c̄_r > gap then break
9:             if Π_1 ∩ Π_2 = ∅ then
10:                for s = 1, ..., nS do
11:                    if S_1[s] + S_2[s] ≥ 1 then
12:                        c̄_r ← c̄_r − σ_s
13:                if c̄_r ≤ gap then Fix ← false, break
14:        if not Fix then break
15:    if Fix then Fix variable x^q_ij to 0
```

7.3 Route Enumeration

The forward route enumeration algorithm represents a feasible (elementary) path $P = (0, \ldots, i)$, $i \in V$, as a label $L(P) = (c(P), \bar{c}(P), v(P) = i, q(P), V(P), S(P), h(P), pred(P))$ storing its original edge costs, reduced cost, end vertex, load, set of visited vertices, vector of states corresponding to the nS lm-SRCs with non-zero dual variables in the current Master LP solution, a hash value depending on $V(P)$, and a pointer to its predecessor label. A label $L(P_1)$ dominates a label $L(P_2)$ if:

$$(i)\ v(P_1) = v(P_2), \quad (ii)\ V(P_1) = V(P_2)$$

$$(iii)\ c(P_1) \leq c(P_2).$$

It is very important to also use completion bounds to eliminate labels that can not be extended to routes with reduced cost smaller than the gap. Let gap be the current gap and let $\widehat{F}$ be the forward completion bounds obtained running the Backward Labeling procedure with respect to the current values of $\bar{c}$ and $\bar{S}$. The forward route enumeration extends a label $L(P)$ with $v(P) = i$ to a customer j if $j \notin V(P)$ if:

$$\bar{c}(P) + \bar{c}^q_{ij} + \widehat{F}(j, q(P) + d_i) < gap.$$

Remark that all the nS lm-SRCs are taken into account in the calculation of $\widehat{F}$, the relaxation is allowing ng-routes instead of only elementary routes. To accelerate the label dominance checking, the labels are stored in a hash table H having $|H|$ buckets, where $|H|$ is a power of 2. Each vertex i in V receives two random numbers between 0 and $|H| - 1$, $r_1(i)$ and $r_2(i)$. A label $L(P)$ is stored in bucket calculated by taking the bitwise exclusive or of the numbers $r_1(i)$, $i \in V(P)$, and $r_2(v(P))$. This ensures that labels that may have a dominance relationship will be in the same bucket. The pseudocode of the Forward Enumeration procedure is shown in Algorithm 7.5. The algorithm also keeps a list $\mathcal{Q}$ of unprocessed labels, initially containing only the starting label. In the end of the algorithm, the labels $L(P)$ with $v(P) = 0$ represent all elementary routes with reduced costs not greater than the duality gap.

A similar enumeration procedure can also be built by extending labels backwards. For this purpose, the list $\mathcal{Q}$ is initialized with starting labels at the depot carrying all possible loads $1 \leq q \leq Q$. The backward enumeration in itself has no advantage. However, as happens with the pricing, it can be used as a component of a significantly faster bidirectional enumerator. In this

Algorithm 7.5 Forward Route Enumeration Algorithm

```
1: procedure FORWARD ENUMERATION
2:     Run Backward Labeling for computing F̂ completion bounds
3:     H(0) ← {L_0 = (0, 0, 0, 0, ∅, 0, 0, nil)}
4:     Insert(Q, L_0)
5:     while Q is not empty do
6:         L_1 = (c_1, c̄_1, i, q, V_1, S_1, h_1, _) ← Remove(Q)
7:         for all (i, j)^q ∈ A_Q such that x^q_ij is not fixed to 0 do
8:             if j ∉ V_1 then
9:                 if c̄_1 + c̄^q_ij + F̂(j, q + d_j) ≤ gap then
10:                    c_2 ← c_1 + c_ij , c̄_2 ← c̄_1 + c̄^q_ij , S_2 ← S_1, h_2 ← h_1 ⊕ r_1(j)
11:                    for s := 1, ..., nS do
12:                        if j ∈ C(s) then
13:                            S_2[s] ← S_2[s] + p(s)
14:                            if S_2[s] ≥ 1 then
15:                                c̄_2 ← c̄_2 − σ_s , S_2[s] ← S_2[s] − 1
16:                    L_2 = (c_2, c̄_2, j, q + d_j, V_1 ∪ {j}, S_2, h_2, pointer to L_1)
17:                    insertLabel ← true
18:                    index ← h_2 ⊕ r_2(j)
19:                    for all L ∈ H(index) do
20:                        if L_2 dominates L then
21:                            Remove(Q, L), H(index) ← H(index) \ {L}, break
22:                        else if L dominates L_2 then insertLabel ← false, break
23:                    if insertLabel then
24:                        H(index) ← H(index) ∪ {L_2}
25:                        Insert(Q, L_2)
```

case, the forward and backward enumeration algorithms are run up to a point around $Q/2$ and a concatenation phase is used for retrieving the elementary paths with reduced cost smaller than the gap.

Algorithm 7.6 shows the pseudocode of the Backward Enumeration procedure and Algorithm 7.7 shows the pseudocode of the Bidirectional Enumeration. The latter algorithm enumerates all forward labels with load $q \leq Q/2$. All these labels are then put in buckets $F(i, q)$. Next, it enumerates all backward labels with load $q > Q/2$. These labels are put in buckets $B(i, q)$. The elementary routes are then obtained by concatenating forward and backward labels. This concatenation is accelerated because the labels inside the buckets are ordered by reduced cost. Procedure $newLabel(L_1, L_2)$ creates a new label that represents the route that is obtained by concatenating the labels L_1 and L_2.

Algorithm 7.6 Backward Route Enumeration Algorithm

procedure BACKWARD ENUMERATION
  Run Forward Labeling for computing $\widehat{B}$ completion bounds
  for $q := 1, \ldots, Q$ **do**
    $H(0) \leftarrow \{L_q = (0, 0, 0, q, \emptyset, \mathbf{0}, 0, nil)\}$
    $Insert(\mathcal{Q}, L_q)$
  while $\mathcal{Q}$ is not empty **do**
    $L_1 = (c_1, \bar{c}_1, j, q, V_1, S_1, h_1, \_) \leftarrow Remove(\mathcal{Q})$
    for all $(i, j)^{q-d_j} \in A_Q$ such that $x_{ij}^{q-d_j}$ is not fixed to 0 **do**
      if $i \notin V_1$ **then**
        if $\bar{c}_1 + \bar{c}_{ij}^{q-d_j} + \widehat{B}(i, q - d_j) \leq$ gap **then**
          $c_2 \leftarrow c_1 + c_{ij}$, $\bar{c}_2 \leftarrow \bar{c}_1 + \bar{c}_{ij}^{q-d_j}$, $S_2 \leftarrow S_1$, $h_2 \leftarrow h_1 \oplus r_1(i)$
          for $s := 1, \ldots, nS$ **do**
            if $i \in C(s)$ **then**
              $S_2[s] \leftarrow S_2[s] + p(s)$
              if $S_2[s] \geq 1$ **then**
                $\bar{c}_2 \leftarrow \bar{c}_2 - \sigma_s$, $S_2[s] \leftarrow S_2[s] - 1$
          $L_2 = (c_2, \bar{c}_2, j, q - d_j, V_1 \cup \{i\}, S_2, h_2,$ pointer to $L_1)$
          $insertLabel \leftarrow$ **true**
          $index \leftarrow h_2 \oplus r_2(i)$
          for all $\mathcal{L} \in H(index)$ **do**
            if L_2 dominates $\mathcal{L}$ **then** $Remove(\mathcal{Q}, \mathcal{L})$, delete $\mathcal{L}$, **break**
            else if $\mathcal{L}$ dominates L_2 **then** $insertLabel \leftarrow$ **false**, **break**
          if $insertLabel$ **then**
            $H(index) \leftarrow H(index) \cup \{L_2\}$
            $Insert(\mathcal{Q}, L_2)$

Algorithm 7.7 Bidirectional Route Enumeration Algorithm

procedure BIDIRECTIONAL ENUMERATION
  Run Forward Labeling for computing $\widehat{B}$ completion bounds
  Run Backward Labeling for computing $\widehat{F}$ completion bounds
  clear(F), clear(B)
  Run Forward Enumeration until all labels with load $q \leq Q/2$ are processed
  for $k := 0, \ldots, |H| - 1$ **do**
    for all $\mathcal{L} = (\_, \_, i, q, \_, \_, \_, \_) \in H(k)$ **do**
      $F(i, q) \leftarrow F(i, q) \cup \mathcal{L}$
  clear(H)
  Run Backward Enumeration until all labels with load $q > Q/2$ are processed
  for $k := 0, \ldots, |H| - 1$ **do**
    for all $\mathcal{L} = (\_, \_, i, q, \_, \_, \_, \_) \in H(k)$ **do**
      $B(i, q) \leftarrow B(i, q) \cup \mathcal{L}$
  clear(H)
  for all $(i, j)^q \in A_Q$ such that x_{ij}^q is not fixed to 0, $q \leq Q/2$, $q + d_j > Q/2$ **do**
    for all $L_1 = (c_1, \bar{c}_1, i, q, V_1, S_1, h_1, \_) \in F(i, q)$ in IRCO **do**
      for all $L_2 = (c_2, \bar{c}_2, j, q + d_i, V_2, S_2, h_2, \_) \in B(j, q + d_i)$ in IRCO **do**
        $\bar{c}_r \leftarrow \bar{c}_1 + \bar{c}_{ij}^q + \bar{c}_2$
        if $\bar{c}_r > 0$ **then break**
        if $V_1 \cap V_2 = \emptyset$ **then**
          for $s := 1, \ldots, nS$ **do**
            if $S_1[s] + S_2[s] \geq 1$ **then**
              $\bar{c}_r \leftarrow \bar{c}_r - \sigma_s$
          if $\bar{c}_r \leq 0$ **then**
            $L_3 \leftarrow newLabel(L_1, L_2)$
            $insert \leftarrow$ **true**
            $index \leftarrow (h_1 \oplus h_2) \bmod |H|$
            for all $\mathcal{L} \in H(index)$ **do**
              if L_3 dominates $\mathcal{L}$ **then** delete $\mathcal{L}$, **break**
              else if $\mathcal{L}$ dominates L_3 **then** $insert \leftarrow$ **false**, **break**
            if $insert$ **then**
              $H(index) \leftarrow H(index) \cup \{L_3\}$

8
Computational Results

In this chapter, we report detailed computational results for a large set of CVRP instances, comparing the proposed BCP with some of the recent exact methods described in Section 1.1. Additional computational results include a comparison of five variants of the pricing presented in Section 7.1. Moreover, some techniques used in the BCP to better handle special instances with a high degree of symmetry are presented.

Our BCP was coded in C++ using Microsoft Visual C++ 2010 Express. IBM ILOG CPLEX Optimizer 12.5 was used as the LP solver and MIP solver in the exact method. The experiments were conducted on an Intel Core i7-3960X 3.30GHz with 64GB RAM running Linux Ubuntu Server 12.04 LTS.

8.1
Problem Instances

The BCP was run over the standard classes of instances (`A`, `B`, `E`, `F`, `M`, and `P`, available at www.branchandcut.org) used for testing exact methods for the CVRP. The name of an instance from this benchmark follows the general format X-nY-kZ, where X denotes the class it belongs, Y corresponds to the number of vertices and Z refers to the *fixed* number of routes (K) required for any feasible solution. As usual in the literature of exact methods, the cost matrix for these instances is calculated from depot and customers coordinates, following the TSPLIB convention of rounding the euclidean distances to the nearest integer. Moreover, in order to better compare our results with those presented in (4), our BCP uses for each of these instances the same initial upper bound they used.

We also report results over instances traditionally used in the literature on heuristic methods, proposed by (9), (10) (instances with initial letter "C") and (28) (instances with initial letter "G"). In those cases, we follow the convention of *not rounding* the Euclidean distances and *not fixing* the number of routes. The initial upper bound used for each of these instances corresponds to the best solution value available in the literature.

8.2 Summary Results for the CVRP

Table 8.1 summarizes the performance of the new BCP over the classical benchmark, comparing with the recent exact algorithms for the CVRP. As usual in the literature where similar tables appear, classes E and M are grouped together. Columns **Opt** indicate the number of instances solved to optimality. Columns **Gap** and **Time** are the average gap in the root node and average times in seconds (over the indicated machines), computed only over the solved instances. The labels **FLL+06**, **BCM08**, **BMR11**, **Con12** and **Rop12** refer to the algorithms proposed in (38), (24), (3), (4), (12) and (56), respectively. Label **This BCP** refers to the proposed BCP.

The newly proposed method has a good performance and could solve all of those instances to optimality. In particular, the hard instances M-n200-k17 and M-n200-k16 were solved for the first time. On the latter it showed that the previous best known solution of value 1278 was not optimal.

We should remark that instances F-n72-k4 ($n/K = 17.75$), P-n101-k4 ($n/K = 25$), and F-n135-k7 ($n/K \approx 19.14$) are still better solved by a pure branch-and-cut algorithm, like (38). In fact column generation is still problematic for these instances. The number of variables in the SPF is roughly proportional to $\binom{n}{n/K}$, it is quite expected that the number of pricing iterations until convergence depends significantly on the average number of customers per route (see, for example, the number of exact pricing iterations in the tables shown in Section 5.6). Furthermore, as also shown in Section 5.6 for the instance M-n121-k7 ($n/K \approx 17.14$), the exact pricing tends to be slower for large n/K, specially with elementary or near-elementary routes. Anyway, faster pricing heuristics can be used to accelerate the column generation. The relatively expensive exact pricing is only called when the heuristics is not able to find routes with negative reduced cost. Unfortunately, our pricing heuristics are based on the bucket pruning technique (24). Such heuristics are not good for large vehicle capacity, since their resulting complexity is $\mathcal{O}(n^2Q)$ (instance F-n135-k7 has $Q = 2210$ and instance F-n72-k4 has $Q = 30000$, whereas the capacity of most instances of the standard dataset ranges from 100 to 200). Faster heuristics can be obtained using the *scaling* and/or *sparsification* techniques (24). The exact pricing can also be improved to better handle instances with large capacity, as discussed in Section 9.1. However, even with such improvements, it is still possible that branch-and-cut remains as the best approach for these instances, at least for F-n135-k7 and F-n72-k4.

As in (24), we have built a hybrid method that is able to automatically switch to a branch-and-cut after severe problems with column generation

Table 8.1: Summary results for the CVRP.

		FLL+06 (24)			BCM08 (3)			BMR11 (4)		
Class	NP	Opt	Gap	Time	Opt	Gap	Time	Opt	Gap	Time
A	22	22	0.81	1961	22	0.2	118	22	0.13	30
B	20	20	0.47	4763	20	0.16	417	20	0.06	67
E-M	12	9	1.19	126987	8	0.69	1025	9	0.49	303
F	3	3	0.14	2398				2	0.11	164
P	24	24	0.76	2892	22	0.28	187	24	0.23	85
Total	81	78			72			77		
Processor		Pentium 4 2.4GHz			Pentium 4 2.6GHz			X7350 2.93GHz		
		Con12 (12)			Rop12 (56)			This BCP		
Class	NP	Opt	Gap	Time	Opt	Gap	Time	Opt	Gap	Time
A	22	22	0.07	59	22	0.57	53	22	0.03	5.6
B	20	20	0.05	89	20	0.25	208	20	0.04	6.2
E-M	12	10	0.3	2807	10	0.96	44295	12	0.19	3669
F	3	2	0.06	3	3	0.25	2163	3	0.00	3679
P	24	24	0.13	43	24	0.69	280	24	0.07	32.7
Total	81	78			79			81		
Processor		E5462 2.8GHz			i7-2620M 2.7GHz			i7-3960X 3.3GHz		

convergence are found. However, in order to stick to the BCP paradigm, we prefer to report the results of a version where convergence problems triggers a dual stabilization strategy, similar to the one described in (48). In fact, without dual stabilization F-n135-k7 can not be solved in reasonable time. It is noteworthy to mention that the algorithms in (4) and (12) use a fast pre-processing stage based on cut generation which gives an initial lower bound close to the optimum value for instances F-n45-K4 and F-n72-k4. The robust BCP in (56) uses ng-sets of cardinality 10 for all instances, except for instance M-n151-k12 (ng-sets have cardinality 15) and instances F-n72-k4 and F-n135-k7 (uses the less computational expensive q-routes without 2-cycles).

Table 8.2 presents detailed information on the resolution of a selected set of larger instances. Besides M-n151-k12, M-n200-k16, and M-n200-k17, it includes results on 2 instances from (10) and 7 instances from (28). In those cases, the number of costumers and the number of routes in the optimal solution (if it is found) are displayed in parenthesis. For each instance and method, column **IUB** present the initial upper bound used by the method. Columns **RLB1**, **ER1**, **RLB2**, **ER2**, **RT(s)** are root node information. They are the lower bound obtained before enumeration (**RLB1**), the number of routes enumerated (if the method performs it and if the enumeration succeeds), (**ER1**), the improved root node lower bound after route enumeration, obtained by adding additional non-robust cuts (if Contardo style enumeration is performed) (**RLB2**), the number of remaining enumerated routes after that (**ER2**) and the total root node computing time (**RT(s)**). The final lower bound given by **FLB**, which is in bold when optimal, the number of nodes in the search tree denoted by **Nodes** and the total computational time in seconds **TT(s)** complete the table columns.

Table 8.2: Comparison of algorithms over hard instances.

Ins	Q	Alg	IUB	RLB1	ER1	RLB2	ER2	RT(s)	FLB	Nodes	TT(s)
M-n151-k12	200	BMR11	1015	1004.3	-			380	1004.3	1	380
		Con12	1015	1008.9	4.0M	1012.5	13K	19041	**1015**	1	19699
		Rop12	1015	1001.5					**1015**	5268	417146
		BCP	1015	1011.7	59K	1012.8	8K	178	**1015**	1	212
		B-BCP	1015	1001.5				109	**1015**	2537	7958
M-n200-k16	200	BMR11		1256.6	-			319	1256.6	1	319
		Con12	1278	1263.0	-	-	-	265589	1263.0	1	265589
		Rop12	1278	1253.0					1258.2	106	7200
		BCP	1278	1266.5	-	-	-	956	**1274**	97	39869
M-n200-k17	200	BMR11	1275	1258.7	-			436	1258.7	1	436
		Con12	1275	1265.1	-	-	-	34351	1265.1	1	34351
		Rop12	1276	1255.3					1261.4	144	7200
		BCP	1275	1268.7	-	-	-	537	**1275**	15	3581
C4 (150,12)	200	BCP	1028.42	1025.1	500k	1026.25	62k	430	**1028.42**	3	783
C5 (199,16)	200	BCP	1291.29	1284.14	-	-	-	595	**1291.29**	59	18690
G17 (240, 22)	20	BCP	707.76	705.54	-	-	-	1010	**707.76**	13	25203
G13 (252,-)	1000	BCP	857.19	851.97	-	-	-	21749	851.97	1	21749
G9 (255,-)	1000	BCP	579.71	576.88	-	-	-	9363	576.88	1	9363
G18 (300,27)	20	BCP	995.13	993.42	-	-	-	1030	**995.13**	15	25690
G14 (320,30)	1000	BCP	1080.55	1076.03	-	-	-	6330	**1080.55**	$\approx$ 3K	$\approx$ 36 days
G10 (323,-)	1000	BCP	736.26	731.13	-	-	-	16021	731.13	1	16021
G19 (360, 33)	20	BCP	1365.60	1363.10	-	-	-	1264	**1365.60**	239	260345

On instance M-n151-k12, Table 8.2 also includes results of Basic BCP (B-BCP), a stripped-down version of the BCP, without SRCs and route enumeration. In order to allow a direct comparison with the robust BCP in (56), Basic BCP only separates cuts from the edge formulation and the ng-sets have cardinality 15. The algorithm still has a relatively good performance. This shows the importance of some "minor elements", like strong branching or the fixing of variables, in the BCP implementation. Nevertheless, the Basic BCP can not solve the instances with 199 customers in reasonable time. The SRCs are essential for that. The performance of the full BCP over the larger instances shows the power of carefully aggregating the elements described in this work. This allowed more than doubling the size of the largest instance proved optimal to date.

8.3 Detailed Computational Results

Tables 8.3, 8.4, 8.5, 8.6 and 8.7 present detailed results for all instances from the datasets `A`, `B`, `E-M`, `F` and `P`, respectively. For each instance, column `Ins` gives the name of the instance, `UB` presents the initial upper bound used and `OPT` shows the value of the optimal solution found (values in bold indicate proven optimality for the first time). The following 10 columns are root node information. The first of these columns indicates the lower bound obtained using only robust cuts (`RLB`) and the next four columns report information immediately before enumeration: the improved lower bound due the addition

of non-robust lm-SRCs (`NRLB`), the number of active lm-SRCs (i.e., those with non-zero dual variables) in the Master LP (`nS`), the percentage of arcs $(i,j)^q \in A_Q$ fixed to zero (`Fix`) and the accumulated time up to this point (`T1`). Then, the number of routes enumerated (the word "limit" indicates that the enumeration was unsuccessful), (`|R|i`), the improved root node lower bound after route enumeration, obtained by adding additional non-robust SRCs and clique cuts (`NLB`), the number of remaining enumerated routes after that (`|R|f`), the time spent with route enumeration plus the time to go from lower bound `NRLB` to lower bound `NLB` (or only the time spent with enumeration, if it is aborted) (`T2`) and, finally, the time spent solving the resulting MIP (`TMIP`) complete the information of the root. Column `Nds` gives the number of nodes in the branch-and-bound tree and `TT` gives the total time in seconds. Additionally, there are other three columns: `BMR11`, `Con12` and `Rop12`. They are, respectively, the time in seconds reported by (4), (12) (rounded to the nearest integer) and (56). The average gaps for the lower bounds `RLB`, `NRLB` and `NLB` are also presented.

Table 8.8 shows detailed results for selected instances proposed by (9) and (10) and Table 8.9 presents detailed results for instances with up to 360 customers proposed by (28). For each instance, two new columns indicate the number of customers and the number of routes in the optimal solution found. These tables do not include columns of competing exact methods because for most of these instances there is no such results reported in the literature.

Figures 8.1, 8.2 and 8.3 illustrate the optimal solution found for instances M-n200-k16, G14 and G19.

Table 8.3: Detailed results on CVRP instances of classes A.

Ins	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BMR11	Con12	Rop12
A-n37-k5	669	669	667.50	669	12	100	0.70						1	0.70	6	8	3
A-n37-k6	949	949	937.06	946.02	64	98.1	2.77	182	946.25	138	0.31	0.02	1	3.10	8	21	17
A-n38-k5	730	730	723.40	730	52	100	1.76						1	1.76	7	13	4
A-n39-k5	822	822	817.83	822	53	100	1.34						1	1.34	7	19	5
A-n39-k6	831	831	824.51	831	56	100	1.29						1	1.29	7	13	6
A-n44-k6	937	937	934.90	937	8	100	0.72						1	0.72	5	11	3
A-n45-k6	944	944	941.26	944	39	100	1.19						1	1.19	12	30	11
A-n45-k7	1146	1146	1140.53	1146	59	100	1.50						1	1.50	11	28	10
A-n46-k7	914	914	914	914	0	100	0.56						1	0.56	8	7	1
A-n48-k7	1073	1073	1071.61	1073	74	100	1.28						1	1.28	15	24	7
A-n53-k7	1010	1010	1003.75	1010	61	100	3.01						1	3.01	24	30	29
A-n54-k7	1167	1167	1155.80	1167	140	100	8.22						1	8.22	35	95	30
A-n55-k9	1073	1073	1068.55	1073	49	100	2.04						1	2.04	19	18	6
A-n60-k9	1354	1354	1344.95	1354	153	100	9.45						1	9.45	46	59	84
A-n61-k9	1034	1034	1023.60	1034	103	100	5.30						1	5.30	37	61	69
A-n62-k8	1290	1288	1280.79	1287.15	121	97.3	18.89	1330	1287.27	950	1.4	0.39	1	20.68	96	105	80
A-n63-k9	1616	1616	1608.60	1616	58	100	4.12						1	4.12	40	78	40
A-n63-k10	1315	1314	1302.42	1310.61	137	97	11.56	1262	1310.75	978	1.17	0.34	1	13.07	51	74	42
A-n64-k9	1402	1401	1387.78	1394.55	100	94.6	11.28	7572	1398.83	667	7.79	0.30	1	19.37	63	224	379
A-n65-k9	1174	1174	1168.63	1174	55	100	3.16						1	3.16	32	47	21
A-n69-k9	1159	1159	1143.81	1159	126	100	7.58						1	7.58	55	90	135
A-n80-k10	1763	1763	1756.46	1763	127	100	13.98						1	13.98	82	248	186
Avg			**0.61**	**0.05**					**0.03**					**5.6**	**30**	**59**	**53**
Solved														**22**	**22**	**22**	**22**

Table 8.4: Detailed results on CVRP instances of classes B.

Ins	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BMR11	Con12	Rop12
B-n38-k6	805	805	805	805	0	100	0.36						1	0.36	4	4	2
B-n39-k5	549	549	549	549	0	100	0.72						1	0.72	6	0	13
B-n41-k6	829	829	829	829	0	100	0.48						1	0.48	4	5	4
B-n43-k6	742	742	737.21	742	127	100	6.10						1	6.1	23	25	9
B-n44-k7	909	909	909	909	0	100	0.48						1	0.48	3	6	2
B-n45-k5	751	751	751	751	0	100	1.23						1	1.23	9	21	19
B-n45-k6	678	678	678	678	0	100	0.77						1	0.77	10	21	11
B-n50-k7	741	741	741	741	0	100	0.98						1	0.98	4	3	2
B-n50-k8	1312	1312	1303.48	1309.68	81	95	10.09	1534	1309.73	1232	0.67	0.65	1	11.41	147	144	51
B-n51-k7	1032	1032	1026.83	1032	114	99	5.43						1	5.43	34	30	42
B-n52-k7	747	747	747	747	0	100	1.31						1	1.31	8	3	12
B-n56-k7	707	707	705	705	0	97.7	1.26	308	705	118	0.08	0.01	1	1.35	11	27	22
B-n57-k7	1153	1153	1153	1153	0	100	2.03						1	2.03	25	123	32
B-n57-k9	1598	1598	1596	1598	26	100	1.78						1	1.78	19	37	14
B-n63-k10	1496	1496	1487.15	1496	49	100	3.24						1	3.24	55	90	27
B-n64-k9	861	861	861	861	0	100	1.96						1	1.96	15	44	26
B-n66-k9	1316	1316	1308.93	1316	136	100	8.81						1	8.81	292	203	43
B-n67-k10	1032	1032	1027.60	1032	118	100	8.08						1	8.08	43	61	49
B-n68-k9	1275	1272	1263.74	1267.45	98	83.2	23.48	233945	1268.77	52517	22.92		3	62.14	526	763	3671
B-n78-k10	1221	1221	1217.20	1221	83	96.1	5.59						1	5.59	97	170	107
Avg			**0.25**	**0.04**					**0.04**					**6.2**	**67**	**89**	**208**
Solved														**20**	**20**	**20**	**20**

Table 8.5: Detailed results on CVRP instances of classes E and M.

Ins	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BMR11	Con12	Rop12
E-n51-k5	521	521	519.28	521	78	100	2.99						1	2.99	8	14	9
E-n76-k7	682	682	671.10	682	230	100	60.76						1	60.76	152	379	1819
E-n76-k8	735	735	726.97	735	183	100	20.54						1	20.54	82	172	767
E-n76-k10	830	830	817.34	828.20	208	99.1	21.50	582	828.41	406	2.84	0.13	1	24.47	114	184	1666
E-n76-k14	1021	1021	1006.94	1016.24	158	97.3	9.28	2038	1016.27	1870	0.72	0.64	1	10.64	52	141	1429
E-n101-k8	815	815	804.38	815	213	100	97.10						1	97.10	579	1092	6611
E-n101-k14	1071	1067	1053.69	1063.91	265	92.8	53.41	176295	1063.97	150216	11.55		3	95.61	453	662	4325
M-n101-k10	820	820	820	820	0	100	4.24						1	4.24	35	13	34
M-n121-k7	1034	1034	1032.48	1034	62	100	47.43						1	47.43	1249	5713	11140
M-n151-k12	1015	1015	1000.43	1011.74	232	98.1	178.38	58944	1012.81	8312	24.77	8.88	1	212.03	-	19699	417146
M-n200-k16	1278	**1274**	1252.54	1266.53	397	86.6	948.64	limit	1266.53		7.85		97	39868.87	-	-	-
M-n200-k17	1275	**1275**	1255.04	1268.71	395	94.0	527.36	limit	1268.71		9.45		5	3580.51	-	-	-
Avg			**1.11**	**0.20**					**0.19**					**3669**	**303**	**2807**	**44295**
Solved														**12**	**9**	**10**	**10**

Table 8.6: Detailed results on CVRP instances of classes F.

Ins	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BMR11	Con12	Rop12
F-n45-k4	724	724	724	724	0	100	41.83						1	41.83	23	1	13
F-n72-k4	237	237	235.58	235.58	0	100	3581.36	525	237	0	2009.52		1	5590.88	304	4	3349
F-n135-k7	1162	1162	1160.46	1162	79	97.1	5405.40						1	5405.38	-	-	3126
Avg			**0.24**	**0.20**					**0**					**3679**	**164**	**3**	**2163**
Solved														**3**	**2**	**2**	**3**

Table 8.7: Detailed results on CVRP instances of classes P.

Ins	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BMR11	Con12	Rop12
P-n16-k8	450	450	448	448	0	93.5	0.01	23	450	0	0.05		1	0.06	1		0
P-n19-k2	212	212	212	212	0	100	0.19						1	0.19	1		1
P-n20-k2	216	216	213	216	5	100	0.19						1	0.19	1		1
P-n21-k2	211	211	211	211	0	100	0.21						1	0.21	1		1
P-n22-k2	216	216	216	216	0	100	0.26						1	0.26	1		2
P-n22-k8	603	603	603	603	0	100	0.06						1	0.06	1		0
P-n23-k8	529	529	529	529	0	100	0.06						1	0.06	1		0
P-n40-k5	458	458	458	458	0	100	0.68						1	0.68	2	4	3
P-n45-k5	510	510	506.96	510	49	100	1.60						1	1.60	6	12	13
P-n50-k7	554	554	551.59	554	72	100	1.72						1	1.72	9	19	6
P-n50-k8	631	631	617.15	628.65	172	98.6	12.29	347	628.71	323	0.65	0.06	1	13.00	68	41	127
P-n50-k10	696	696	689.36	696	50	100	1.01						1	1.01	9	19	6
P-n51-k10	741	741	736.19	741	35	100	0.70						1	0.70	11	20	6
P-n55-k7	568	568	559.20	566.02	122	99.1	10.48	375	566.91	106	3.29	0.01	1	13.78	17	48	98
P-n55-k8	588	588	580.62	586.76	140	99.6	10.92	134	587.41	0	0.11		1	11.03	18	17	36
P-n55-k10	699	694	681.79	690.19	128	91.5	9.34	8361	690.23	7455	0.82	0.92	1	11.08	29	39	110
P-n55-k15	993	989	972.54	987.02	66	97.5	3.52	355	987.11	343	0.32	0.03	1	3.87	27	86	16
P-n60-k10	756	744	738.90	743.68	84	87.3	5.70	31448	744	25537	2.08	1.11	1	8.89	30	28	15
P-n60-k15	1033	968	963.47	963.47	0	15.2	0.67	1172814	968	0	7.71		1	8.38	73	30	4
P-n65-k10	792	792	787.09	792	65	100	1.72						1	1.72	14	42	14
P-n70-k10	834	827	814.33	823.89	171	89.6	21.70	64906	824.64	43486	7.65		3	44.81	166	133	534
P-n76-k4	593	593	589.22	593	133	100	65.71						1	65.71	118	363	497
P-n76-k5	627	627	617.74	627	193	100	169.14						1	169.14	282	1331	3888
P-n101-k4	681	681	678.57	681	113	100	402.25						1	402.25	1155	5793	1343
Avg			**0.78**	**0.13**					**0.07**					**32.7**	**85**	**43**	**280**
Solved														**24**	**24**	**24**	**24**

Table 8.8: Detailed results on instances from (9) and (10).

Ins	n	Ksol	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT
C1	50	5	524.61	524.61	522.98	524.61	77	100	3.72						1	3.72
C2	75	10	835.26	**835.26**	822.40	833.26	222	98.2	35.27	2055	833.37	1657	2.26	1.00	1	38.53
C3	100	8	826.14	**826.14**	815.02	824.02	163	96.6	137.60	63051	826.14	427	17.63	0.09	1	155.32
C12	100	10	819.56	**819.56**	819.56	819.56	0	100	4.07						1	4.07
C11	120	7	1042.12	**1042.12**	1041.97	1042.12	73	100	41.07						1	41.07
C4	150	12	1028.42	**1028.42**	1013.35	1025.10	277	96.6	330.56	500642	1026.25	62317	99.43		3	782.61
C5	199	16	1291.29	**1291.29**	1270.09	1284.14	364	91.5	586.01	limit			9.40		59	18690.20
Avg					**0.90**	**0.20**					**0.14**					**2816**
Solved																**7**

Table 8.9: Detailed results on instances from (28).

Ins	n	Ksol	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT
G17	240	22	707.76	**707.76**	700.91	705.54	833	92.2	993.44	limit			16.96		13	25202.88
G13	252		857.19		845.80	851.97	705	94.6	21564.95	limit			184.41		1	21749.36
G9	255		579.71		575.67	576.88	408	94.3	9276.06	limit			87.06		1	9363.12
G18	300	27	995.13	**995.13**	988.36	993.42	1031	96.1	1012.44	limit			17.76		15	25690.27
G14	320	30	1080.55	**1080.55**	1070.53	1076.03	702	96.1	6210.38	limit			120.09		≈ 3K	≈ 36 days
G10	323		736.26		729.86	731.13	469	87.8	15942.88	limit			78.31		1	16021.19
G19	360	33	1365.60	**1365.60**	1356.18	1363.10	930	95.1	1248.40	limit			15.20		239	260344.52
Avg					**0.64**	**0.17**										≈ **9.9** days
Solved																**4**

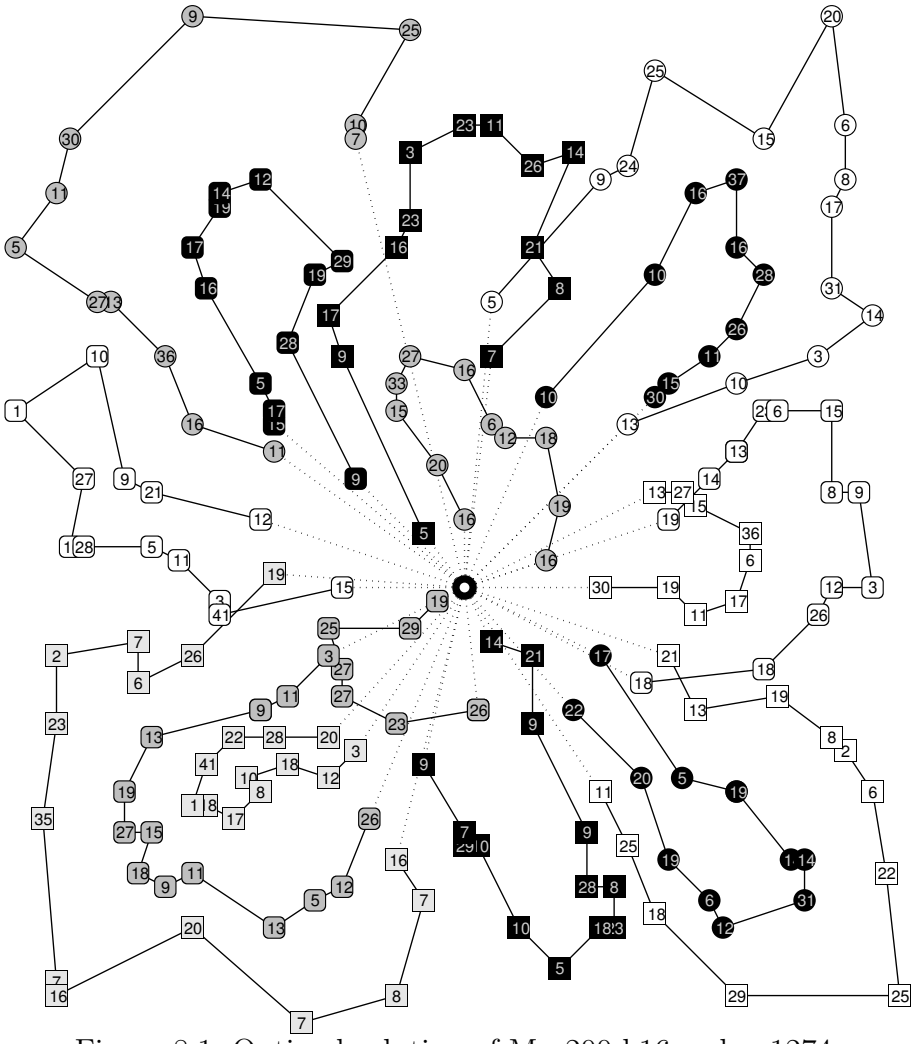

Figure 8.1: Optimal solution of M-n200-k16, value 1274.

8.3.1 Results for the Pricing

Table 8.10 shows detailed results over a number of selected instances of five variants of the pricing: 1 - Forward Labeling (Algorithm 7.1); 2 - Bidirectional Labeling with concatenation at $Q/2$; 3 - Bidirectional Labeling with dynamic determination of the concatenation point (Algorithm 7.3); 4 - The later algorithm with mild use of completion bounds (n'_S is small); 5 - The same algorithm with aggressive use of completion bounds (n'_S is larger). For each instance, there are five rows, each giving the results for a pricing variant. The name of the columns are: **Ins** - The name of the instance; **UB** - The initial upper bound; **LB** - The root lower bound obtained; **Fix** - the percentage of arcs $(i,j)^q \in A_Q$ fixed at the end of the root node; **Alg** - The pricing variant; **nS** - the number of lm-SRCs with non-zero duals; **nFL** - the number of forward labels; **FT** - the time to run the Forward Labeling; **nBL** - the number of backward labels; **BT** - the time to run the Backward Labeling; **CT** - the time

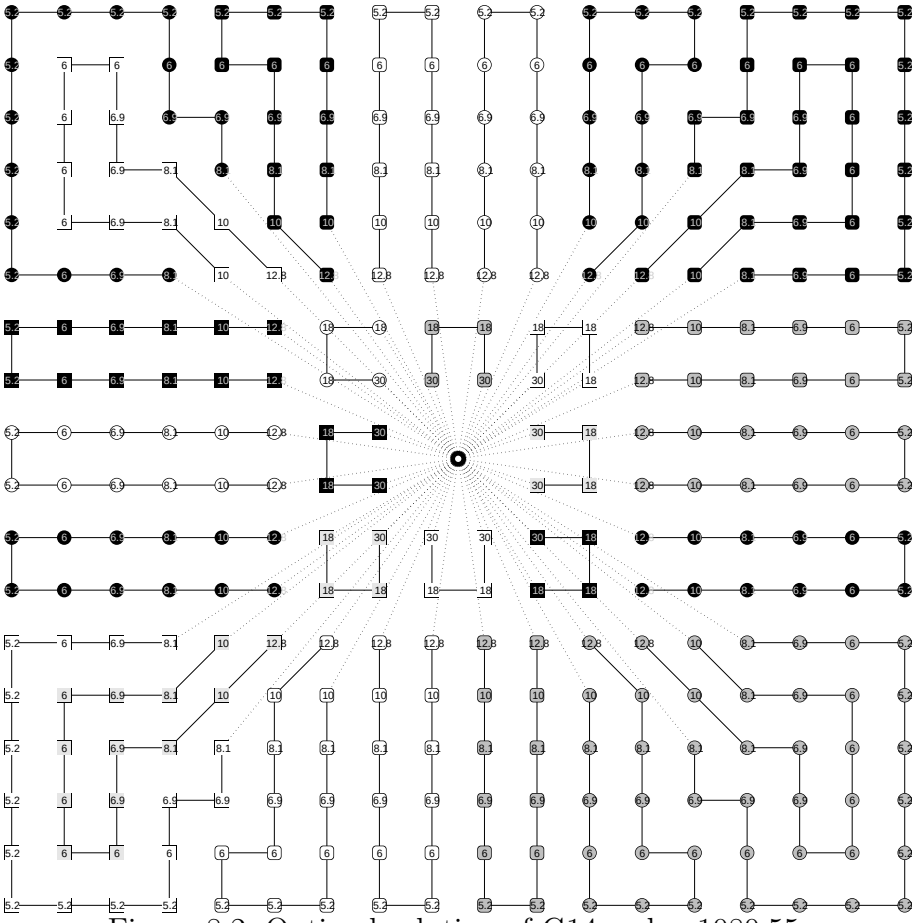

Figure 8.2: Optimal solution of G14, value 1080.55.

to run the concatenation; **CBT** - the time to calculate completion bounds; **TT** - the total time of the pricing; and **qf** - the concatenation point. The number of customers is indicated in parentheses for the instances proposed in (28). Note that for variant 1, $\mathbf{qf} = Q$.

The results (columns from **nS** to **qf**) are an average of the rounds of the exact pricing performed in the end of the root node, after the last separation of lm-SRCs. In order to ensure a fair comparison, in this particular experiment, all the pricing variants are run "in parallel" with exactly the same input, the reduced costs. We only use columns found by Variant 5, the columns from the other variants are discarded.

Table 8.10: Results for 5 variants of the pricing.

Ins	UB	LB	Fix	Alg	nS	nFL	FT	nBL	BT	CT	CBT	TT	qf
B-n68-k9	1272	1268.43	93.0	1	206	681,215	162.13					162.13	100
				2		29,305	1.10	123,533	9.36	0.76		11.21	50
				3		56,400	3.10	69,091	3.60	0.77		7.46	57
				4		56,886	3.40	66,073	3.53	0.81	0.06	7.80	57
				5		55,000	3.13	64,386	3.54	0.81	4.83	12.31	57
E-n101-k8	815	814.39	98.3	1	190	81,238	0.33					0.33	200
				2		11,539	0.03	54,223	0.19	0.03		0.25	100
				3		24,802	0.08	30,123	0.08	0.03		0.19	123
				4		24,165	0.08	29,837	0.08	0.03	0.04	0.23	123
				5		24,577	0.08	28,792	0.08	0.03	0.14	0.33	123

M-n151-k12	1015	1012.54	98.7	1	349	300,434	3.12					3.12	200
				2		53,043	0.28	213,691	1.95	0.14		2.37	100
				3		110,990	0.79	111,542	0.79	0.16		1.73	122
				4		111,731	0.78	107,555	0.77	0.16	0.11	1.83	123
				5		107,585	0.72	104,803	0.71	0.16	0.63	2.23	122
M-n200-k16	1278	1266.13	86.0	1	362	824,411	30.13					30.13	200
				2		205,676	4.31	445,318	10.60	0.20		15.10	100
				3		290,721	7.10	322,890	6.98	0.21		14.29	113
				4		282,208	2.86	264,891	2.86	0.21	1.00	6.92	118
				5		164,252	1.63	293,159	1.63	0.17	2.85	6.28	96
M-n200-k17	1275	1268.53	93.9	1	362	713,344	15.80					15.80	200
				2		182,203	2.24	442,487	7.16	0.16		9.56	100
				3		275,242	4.15	296,893	4.15	0.17		8.46	116
				4		265,776	2.12	240,002	2.15	0.17	0.64	5.08	121
				5		183,267	1.38	239,635	1.36	0.15	2.26	5.16	104
G17 (240)	707.76	705.7	92.3	1	1015	574,434	52.56					52.56	20
				2		142,898	8.41	184,784	12.47	1.61		22.48	10
				3		142,898	8.58	184,784	12.75	1.59		22.93	10
				4		175,048	7.27	137,401	6.60	1.73	0.14	15.74	11
				5		136,400	5.41	127,199	3.96	1.60	2.52	13.48	10
G13 (242)	857.19	851.74	94.1	1	598	8,945,606	711.68					711.68	1000
				2		691,079	24.46	5,939,668	397.29	1.52		423.26	500
				3		2,063,475	105.74	2,378,698	106.14	1.32		213.20	635
				4		2,018,630	60.62	1,974,895	61.01	1.22	3.69	126.54	651
				5		1,009,351	26.79	1,648,510	26.84	1.56	21.63	76.83	548
G9 (255)	579.71	576.84	94.2	1	371	4,721,819	185.30					185.30	1000
				2		1,868,427	58.49	3,812,497	142.65	1.10		202.24	500
				3		2,589,046	90.78	2,644,367	90.84	0.86		182.48	568
				4		2,556,700	51.12	2,172,705	51.30	0.63	5.10	108.14	589
				5		818,102	14.46	1,629,212	14.40	0.11	24.64	53.60	579
G14 (320)	1080.55	1076.12	96.1	1	712	6,856,892	338.01					338.01	1000
				2		1,062,545	28.18	5,346,066	234.80	0.41		263.39	500
				3		2,393,118	87.77	2,601,846	87.88	0.34		175.99	624
				4		2,397,374	39.57	1,948,837	39.74	0.31	5.06	84.67	652
				5		1,181,803	17.68	1,863,797	17.74	0.41	20.05	55.88	523

The results in Table 8.10 show that Variant 5 is the best pricing algorithm, at least for very hard instances, those with more than 199 customers. Therefore, it was the variant chosen for the proposed BCP.

8.3.2 Handling Symmetrical Instances

A high level of symmetry is observed in most instances from (28). This impacts negatively the BCP performance, making the cutting and branching operations less efficient than usual. In fact, whenever a fractional solution is cut, it is likely that a symmetric solution with the same cost will appear in its place. The lower bounds can only move after all symmetric solutions are cut. We implemented some special techniques in our BCP in order to mitigate this negative impact.

The proposed BCP has a procedure that automatically detects, for instances that use non-rounded euclidean distances calculated from the customer

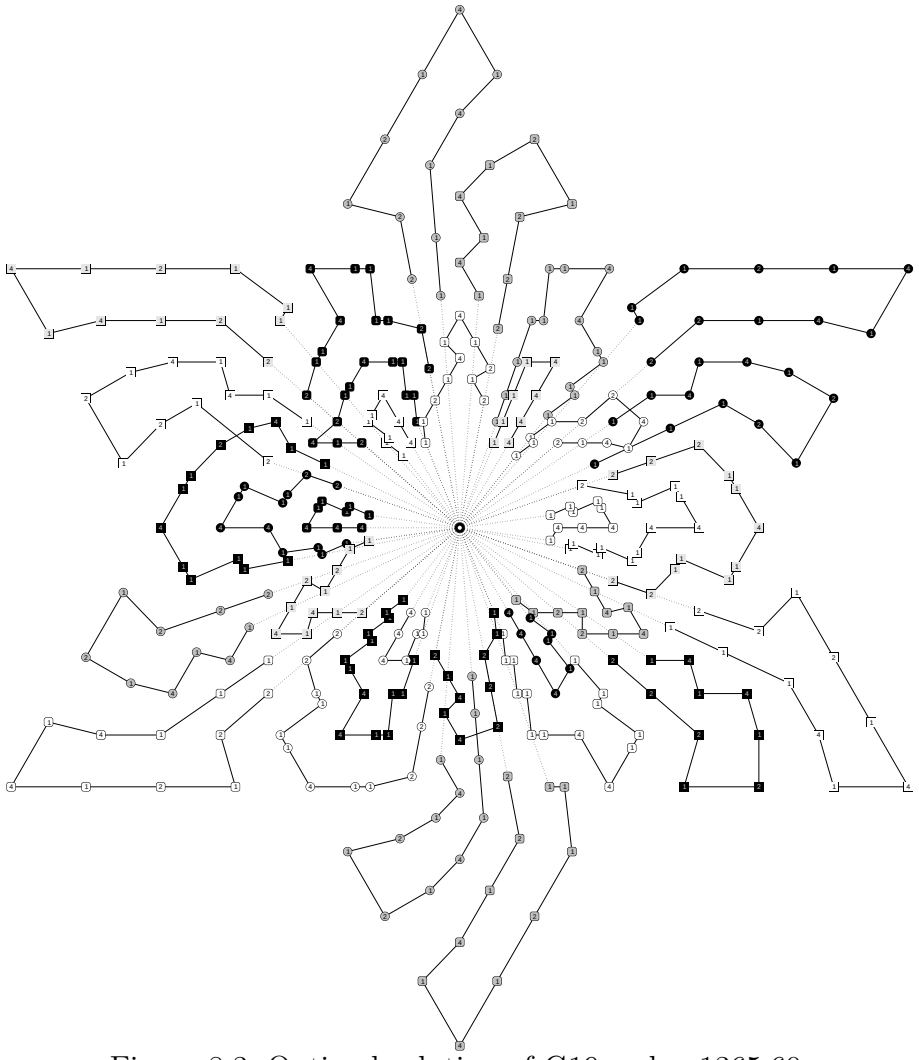

Figure 8.3: Optimal solution of G19, value 1365.60.

and depot coordinates, angles of rotation and reflection with respect to the depot that result in equivalent instances. For example, the instance G14 (see Figure 8.2) remains the same if it is rotated around the depot in an angle of 90 degrees or reflected over a horizontal line passing through the depot. As a result, for every feasible solution for this instance (fractional or integral), there are 8 symmetric solutions with the same cost. For the instance G19, shown in Figure 8.3, the number of symmetric solutions is 12 because the rotation angle is 60 degrees. More precisely, the procedure uses the rotation and reflection operations mentioned before to partition the set of arcs into symmetry groups. In instances G9-G12, each group has 2 members, in instances G13-G19, each group has 8 members, in instances G17-G20, the groups have size 12. Instances G1-G8 are also very symmetric, but they are not considered in this CVRP work because they have additional distance constraints.

The BCP benefits from the symmetry detection in two ways.

1. For every variable x_a^q fixed to zero by reduced cost, all variables $x_{a'}^q$ where

$a' \in Sym(a)$, the symmetry group of a, can also be fixed to zero. This is valid because the same argument that was used to fix a can be turned into an argument to fix a', by only switching indices.

2. Consider the simple disjunction $(x_a = 1) \vee (x_a = 0)$ on the branching. The stronger disjunction $(x_a = 1) \vee (\sum_{a \in Sym(a)} x_a = 0)$ is also valid. This can be explained as follows: if all optimal solutions use at least one arc in $Sym(a)$, there is an optimal solution that uses a. It is important to remark that the left node (corresponding to $x_a = 1$) from this branching is not completely symmetrical anymore. This is good, but also means that the symmetry related tricks can not be used in the corresponding subtree. On the other hand, the right node (corresponding to $\sum_{a \in Sym(a)} x_a = 0$) is still symmetrical.

   Now, consider a disjunction $(\sum_{a \in \delta^-(S)} x_a = 1) \vee (\sum_{a \in \delta^-(S)} x_a \geq 2)$ on the branching. Let X be a non-empty proper subset of the arcs in A. We say that the subset $X' \subset A$, $|X'| = |X|$, belongs to $Sym(X)$ if every arc in X can be mapped into an arc in X' by the same combination of rotation and reflection operations. It can be seen that all possible sets $Sym(X)$ have the same cardinality $|Sym|$, which is also the cardinality of the groups of symmetrical arcs. We claim that the right side of disjunction can be strengthened to $(\wedge_{X' \in Sym(\delta^-(S))} \sum_{a \in X'} x_a \geq 2)$. In other words, $|Sym|$ inequalities can be imposed on the right node. This is valid because if in all optimal solutions there is a set $X' \in Sym(\delta^-(S))$ such that $\sum_{a \in X'} x_a = 1$, there is an optimal solution where $\sum_{a \in \delta^-(S)} x_a = 1$. Using this improved disjunction, the symmetry is partially broken in the left child. But is maintained in the right child, and thus can be exploited again.

The use of those techniques makes a lot a difference in the BCP performance on highly symmetrical instances. For example, instance G17 takes more than 3 days to be solved without them.

9
Conclusions

This thesis addressed the Capacitated Vehicle Routing Problem (CVRP), the problem of designing optimal vehicle routes to serve a set of customers using a fleet of homogeneous vehicles stationed at a central depot. The CVRP is one of the most studied generalizations of the Traveling Salesman Problem. The idea is that state-of-the-art exact algorithms for it can be adapted to to successfully solve other VRP variants.

Although the focus of this thesis is the CVRP, the first contribution presented is related to Shortest Path Problem with Resource Constraint (SPPRC) and the Elementary SPPRC. These interesting problems are useful to solve the VRPs via column generation and branch-and-price. Efficient exact algorithms for the (E)SPPRC were proposed. Some of them combine the most recent techniques for these problems, such as Decremental State-Space Relaxation (DSSR) (54), *ng*-routes (3), bidirectional search (53) and completion bounds. These algorithms were embedded in a column and cut generation algorithm to compute lower bounds for the CVRP. Extensive computational results are reported showing that the best performing algorithms are capable of pricing *ng*-routes with *ng*-set sizes up to sixty-four for hard CVRP instances. Moreover, it is shown that the pricing algorithms are superior than the best algorithm proposed in (54) for the ESPPRC. Our algorithm starts with empty *ng*-sets. At each DSSR iteration, it identifies one or more cycles on the best solutions and include the repeated vertex in the *ng*-sets of all vertices which compose the cycle. This approach performs better than the regular DSSR for elementary routes because it does not add the vertex restrictions globally, thus keeping the dominance rule easier to be evaluated.

Moreover, important contributions are presented for the exact solution of the CVRP. The newly Branch-Cut-and-Price was the result of a deliberate effort of testing, improving and combining ideas proposed by several authors, such as: extended formulation, *ng*-routes, strong branching, robust and non-robust cuts (including Subset Row Cuts), bidirectional pricing, variable fixing and route enumeration. A new form of SRC is presented, the limited memory SRCs (lm-SRCs). These cuts represent a weaker version of SRCs carefully

designed to be efficiently used in the context of a column generation algorithm. These new cuts were decisive on the exact resolution of the last two open instances from the standard benchmark dataset, instances M-n200-k17 and M-n200-k16. Moreover, very hard instances with up to 360 customers, never tackled before by exact methods, could be solved to optimality. Detailed and extensive computational results on a large set of instances are reported, including experiments with a number of pricing variants.

We conclude by stating more personal views on what general BCP algorithm construction can learn from that CVRP experience:

- The separation of non-robust cuts should be as much integrated with the pricing as possible. More precisely, besides taking classical polyhedral considerations into account, the non-robust cuts should be designed in order to minimize their impact on the specific kind of algorithm used in the pricing. In the BCP presented in this paper, the limited memory SRCs has a quite odd algorithmic definition that only makes sense in the context of the labeling algorithm. For example, suppose an alternative BCP for the CVRP where a MIP model is used to price elementary routes. The limited memory SRCs would not fit in that algorithm, they would probably cause more negative impact in the pricing than ordinary SRCs.
- When designing non-robust cuts, it is desirable to have a parameter that allows a smooth control on cut strength vs impact in the pricing. In the case of limited memory SRCs, the parameter M has that role.
- Even if designed and separated in a careful way, at some point the non-robust cuts can indeed make the pricing intractable, halting the BCP algorithm. Therefore, it is advisable to have suitable escape mechanisms, like the rollback introduced in this work.
- Fixing variables from the original formulation by reduced costs can help a lot. But this fixing can be more effective if the original formulation is chosen in order to match the algorithm using in the pricing. In fact, in the arc-load formulation, the vertices in V_Q have a one-to-one correspondence with the buckets in the labeling algorithm.
- The idea of finishing a BCP by enumerating all columns that may be part of the optimal solution into a pool and perform pricing by inspection after that and eventually sending a much reduced problem to a MIP solver can be quite effective in some instances. However, for consistency reasons, it should be hybridized with traditional branching.

– Strong branching is now a standard technique for improving branch-and-bound or branch-and-cut performance. Our experience confirms the statement by (56) that it can also improve a lot the performance of BCP algorithms.

9.1 Future Work and Extensions

The results over F instances from the standard benchmark showed that the performance of the BCP is greatly affected by the vehicle capacity. On those kind of instances, it is possible that the Two-Index Formulation (TIF) may be a better underlying formulation than the Arc-Load Formulation (ALF). The fixing by reduced costs or cuts over the x_{ij}^d variables have at least two drawbacks:

– the symmetry of the CVRP can not be exploited by the bidirectional labeling algorithm for exact pricing, route enumeration and the algorithm for variable fixing, a backward labeling procedure must be explicitly performed due the asymmetry of ALF; and
– the dominance rule in the labeling algorithm for the exact pricing is weakened, it can only be checked for labels within the same bucket;

For instances with large Q, this weakened dominance may be more evident, paths using (a bit) less load are more likely to dominate others using higher load when Q is large. Therefore, it may be the case that the improved fixing (or cuts) over the variable from ALF does not compensate these drawbacks.

As for future work, we suggest: (i) an improved BCP capable to dynamically choose the most appropriate underlying formulation for the instance; (ii) implement fast pricing heuristics less sensitive to the value of the vehicle capacity; and, more importantly (iii) extend the BCP to other VRP variants that may include time windows and/or other likely additional constraints.

Remark that most of the techniques implemented by the BCP are potentially adaptable to other VRP variants. For instance, the lm-SRCs are cuts for the SPF, which is a general formulation for VRPs. The strong branching is also quite generic. It is worth mentioning that, even some CVRP instances can take advantage of the symmetry, the work done on the extended formulation has great potential benefits. Because most of the VRPs are asymmetric, this means that the code is much more generic, the labeling algorithms for pricing, fixing and route enumeration perform backward labeling procedures. We believe that with minor changes our BCP would run for the Asymmetric CVRP, the Time Dependent Traveling Salesman Problem

(TDTSP), the Time Dependent Vehicle Routing Problem (TDVRP) and the "green" VRP, where the cost of an arc (CO_2 emission) depends on the load. Moreover, there are other important variants such as the HFVRP and parallel machine scheduling where we already know that extended cuts works well.

Anyway, each variant has its own particularities, which means that the task of extending the BCP for some of them may be far from trivial, as we aspect for the VRPTW.

Bibliography

[1] ACHTERBERG, T. **Constraint Integer Programming**. 2007. PhD thesis - Technische Universitat Berlin.

[2] AGARWAL, Y.; MATHUR, K. ; SALKIN, H. **Networks**. A set-partitioning based exact algorithm for the vehicle routing problem, journal, v.19, p. 731–739, 1989.

[3] BALDACCI, R.; CHRISTOFIDES, N. ; MINGOZZI, A. **Mathematical Programming**. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts, journal, v.115, n.2, p. 351–385, 2008.

[4] BALDACCI, R.; MINGOZZI, A. ; ROBERTI, R. **Operations Research**. New route relaxation and pricing strategies for the vehicle routing problem, journal, v.59, n.5, p. 1269–1283, 2011.

[5] BALINSKI, M.; QUANDT, R. **Operations Research**. On an integer program for a delivery problem, journal, v.12, n.2, p. 300–304, 1964.

[6] BARTOLINI, E.; CORDEAU, J.-F. ; LAPORTE, G. **Mathematical Programming**. Improved lower bounds and exact algorithm for the capacitated arc routing problem., journal, v.137, n.1-2, p. 409–452, 2013.

[7] BOLAND, N.; DETHRIDGE, J. ; DUMITRESCU, I. **Operations Research Letters**. Accelerated label setting algorithms for the elementary resource constrained shortest path problem, journal, v.34, n.1, p. 58–68, 2006.

[8] CHABRIER, A. **Computers & Operations Research**. Vehicle routing problem with elementary shortest path based column generation, journal, v.33, n.10, p. 2972–2990, Oct. 2006.

[9] CHRISTOFIDES, N.; EILON, S. **Operational Research Quarterly**. An algorithm for the vehicle-dispatching problem, journal, v.20, p. 309–318, 1969.

[10] CHRISTOFIDES, N.; MINGOZZI, A. ; TOTH, P. **The vehicle routing problem**. In: Christofides, N.; Mingozzi, A.; Toth, P. ; Sandi, C., editors, COMBINATORIAL OPTIMIZATION, volume 1, p. 315–338. Wiley Interscience, 1979.

[11] CHRISTOFIDES, N.; MINGOZZI, A. ; TOTH, P. **Mathematical Programming**. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations, journal, v.20, p. 255–282, 1981.

[12] CONTARDO, C. **A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints**. Technical report, Archipel-UQAM 5078, Université du Québec à Montréal, Canada, 2012.

[13] CONTARDO, C.; CORDEAU, J.-F. ; GENDRON, B. **A branch-and-cut-and-price algorithm for the capacitated location-routing problem**. Technical report, CIRRELT-2011-44, Université de Montréal, 2011.

[14] DANTZIG, G.; RAMSER, R. **Management Science**. The truck dispatching problem, journal, v.6, p. 80–91, 1959.

[15] DANTZIG, G.; WOLFE, P. **Operations Research**. Decomposition principle for linear programs, journal, v.8, p. 101–111, 1960.

[16] DESAULNIERS, G.; DESROSIERS, J. ; SOLOMON, J. **Column Generation**. Springer, 2005.

[17] DESROCHERS, M.; DESROSIERS, J. ; SOLOMON, M. **Operations research**. A new optimization algorithm for the vehicle routing problem with time windows, journal, v.40, n.2, p. 342–354, 1992.

[18] DESROSIERS, J.; SOUMIS, F. ; DESROCHERS, M. **Networks**. Routing with time windows by column generation, journal, v.14, p. 545–565, 1984.

[19] DI PUGLIA PUGLIESE, L.; GUERRIERO, F. **Networks**. A survey of resource constrained shortest path problems: Exact solution approaches, journal, v.62, n.3, p. 183–200, 2013.

[20] DROR, M. **Operations Research**. Note on the complexity of the shortest path models for column generation in VRPTW, journal, v.42, n.5, p. 977–978, 1994.

[21] FEILLET, D.; DEJAX, P.; GENDREAU, M. ; GUEGUEN, C. **Networks**. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems, journal, v.44, n.3, p. 216–229, 2004.

[22] FISHER, M. **Operations Research**. Optimal solution of vehicle routing problem using minimum k-trees, journal, v.42, p. 626–642, 1994.

[23] FORD, L. R., J.; FULKERSON, D. R. **Management Science**. A suggested computation for maximal multi-commodity network flows, journal, v.5, n.1, p. pp. 97–101, 1958.

[24] FUKASAWA, R.; LONGO, H.; LYSGAARD, J.; POGGI DE ARAGÃO, M.; REIS, M.; UCHOA, E. ; WERNECK, R. F. **Mathematical Programming**. Robust branch-and-cut-and-price for the capacitated vehicle routing problem, journal, v.106, n.3, p. 491–511, 2006.

[25] GILMORE, P.; GOMORY, R. **Operations Research**. A linear programming approach to the cutting stock problem, journal, v.9, p. 849–859, 1961.

[26] GILMORE, P. C.; GOMORY, R. E. **Operations Research**. A linear programming approach to the cutting stock problem-part II, journal, v.11, n.6, p. 863–888, 1963.

[27] GODINHO, M. T.; GOUVEIA, L.; MAGNANTI, T.; PESNEAU, P. ; PIRES, J. **On time-dependent models for unit demand vehicle routing problems**. In: INTERNATIONAL NETWORK OPTIMIZATION CONFERENCE (INOC), 2007.

[28] GOLDEN, B. L.; WASIL, E. A.; KELLY, J. P. ; CHAO, I. **The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results**. In: FLEET MANAGEMENT AND LOGISTICS, p. 33–56. Springer, 1998.

[29] HADJICONSTANTINOU, E.; CHRISTOFIDES, N. ; MINGOZZI, A. **Annals of Operations Research**. A new exact algorithm from the vehicle routing problem based on q-paths and k-shortest paths relaxations, journal, v.61, n.1, p. 21–43, 1995.

[30] HOUCK, D.; PICARD, J.; QUEYRANNE, M. ; VEMUGANTI, R. **Opsearch**. The travelling salesman problem as a constrained shortest path problem, journal, v.17, p. 93–109, 1980.

[31] IRNICH, S.; DESAULNIERS, G.; DESROSIERS, J. ; HADJAR, A. **INFORMS Journal on Computing**. Path-reduced costs for eliminating arcs in routing and scheduling, journal, v.22, n.2, p. 297–313, 2010.

[32] IRNICH, S.; VILLENEUVE, D. **INFORMS Journal on Computing**. The shortest-path problem with resource constraints and k-cycle elimination for k $\geq$ 3, journal, v.18, n.3, p. 391–406, 2006.

[33] JEPSEN, M.; PETERSEN, B.; SPOORENDONK, S. ; PISINGER, D. **Operations Research**. Subset-row inequalities applied to the vehicle-routing problem with time windows, journal, v.56, n.2, p. 497–511, 2008.

[34] LAPORTE, G.; NOBERT, Y. **OR Spectrum**. A branch and bound algorithm for the capacitated vehicle routing problem, journal, v.5, n.2, p. 77–85, 1983.

[35] LETCHFORD, A.; SALAZAR-GONZÁLEZ, J. **Mathematical Programming**. Projection results for vehicle routing, journal, v.105, n.2, p. 251–274, 2006.

[36] LÜBBECKE, M.; DESROSIERS, J. **Operations Research**. Selected topics in column generation, journal, v.53, p. 1007–1023, 2004.

[37] LYSGAARD, J. **CVRPSEP: A package of separation routines for the capacitated vehicle routing problem**, 2003. Available at `www.asb.dk/~lys`.

[38] LYSGAARD, J.; LETCHFORD, A. N. ; EGLESE, R. W. **Mathematical Programming**. A new branch-and-cut algorithm for the capacitated vehicle routing problem, journal, v.100, n.2, p. 423–445, June 2004.

[39] MARTINELLI, R. **Exact Algorithms for Arc and Node Routing Problems**. 2012. PhD thesis - Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio).

[40] MARTINELLI, R.; PECIN, D. ; POGGI, M. **European Journal of Operational Research**. Efficient elementary and restricted non-elementary route pricing, journal, v.239, p. 102–111, 2014.

[41] MARTINHON, C.; LUCENA, A. ; MACULAN, N. **European Journal of Operational Research**. Stronger k-tree relaxations for the vehicle routing problem, journal, v.158, n.1, p. 56–71, 2004.

[42] MILLER, D. **ORSA Journal on Computing**. A matching based exact algorithm for capacitated vehicle routing problems, journal, v.7, p. 1–9, 1995.

[43] NADDEF, D.; RINALDI, G. **Branch-and-cut algorithms for the capacitated VRP**. In: Toth, P.; Vigo, D., editors, THE VEHICLE ROUTING PROBLEM, chapter 3, p. 53–84. SIAM, 2002.

[44] NEMHAUSER, G. L.; WOLSEY, L. A. **Integer and Combinatorial Optimization**. New York, NY, USA: Wiley-Interscience, 1988.

[45] PECIN, D.; PESSOA, A.; POGGI, M. ; UCHOA, E. **Improved branch-cut-and-price for capacitated vehicle routing**. In: INTEGER PROGRAMMING AND COMBINATORIAL OPTIMIZATION, p. 393–403. Springer, 2014.

[46] PECIN, D. G. **Uso de Rotas Elementares no CVRP**. 2010. Master's thesis - Instituto de Informática, Universidade Federal de Goiás. in portuguese.

[47] PESSOA, A.; POGGI, M. ; BARBOZA, E. U. **Robust branch-cut-and-price algorithms for vehicle routing problems**. In: THE VEHICLE ROUTING PROBLEM: LATEST ADVANCES AND NEW CHALLENGES, p. 297–325. Springer, Berlin, 2008.

[48] PESSOA, A.; SADYKOV, R.; UCHOA, E. ; VANDERBECK, F. **In-out separation and column generation stabilization by dual price smoothing**. In: SYMPOSIUM ON EXPERIMENTAL ALGORITHMS, p. 354–365. Springer, 2013.

[49] PESSOA, A.; UCHOA, E.; DE ARAGÃO, M. P. ; RODRIGUES, R. **Mathematical Programming Computation**. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems, journal, v.2, n.3-4, p. 259–290, 2010.

[50] PESSOA, A.; UCHOA, E. ; POGGI DE ARAGÃO, M. **Networks**. A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem, journal, v.54, n.4, p. 167–177, 2009.

[51] POGGI DE ARAGÃO, M.; UCHOA, E. **Integer program reformulation for robust branch-and-cut-and-price algorithms**. In: IN PROCEEDINGS OF THE CONFERENCE MATHEMATICAL PROGRAM IN RIO: A CONFERENCE IN HONOUR OF NELSON MACULAN, p. 56–61, 2003.

[52] PUGLIESE, L. D. P.; GUERRIERO, F. **Annals of Operations Research**. A computational study of solution approaches for the resource constrained elementary shortest path problem, journal, v.201, n.1, p. 131–157, 2012.

[53] RIGHINI, G.; SALANI, M. **Discrete Optimization**. Symmetry helps: Bounded bidirectional dynamic programming for the elementary shortest path problem with resource constraints, journal, v.3, n.3, p. 255–273, Sept. 2006.

[54] RIGHINI, G.; SALANI, M. **Networks**. New dynamic programming algorithms for the resource constrained elementary shortest path problem, journal, v.51, n.3, p. 155–170, 2008.

[55] ROBERTI, R.; MINGOZZI, A. **Transportation Science**. Dynamic ng-path relaxation for the delivery man problem, journal, 2013. to appear.

[56] RØPKE, S. **Presentation in Column Generation 2012**. Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems, journal, 2012.

[57] SANTOS, H.; TOFFOLO, T.; GOMES, R. ; RIBAS, S. **Annals of Operations Research**. Integer programming techniques for the nurse rostering problem, journal, 2014. To appear.

[58] TOTH, P.; VIGO, D. **Discrete Applied Mathematics**. Models, relaxations and exact approaches for the capacitated vehicle routing problem, journal, v.123, p. 487–512, 2002.

[59] TOTH, P.; VIGO, D. **The Vehicle Routing Problem**. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, 2002.

[60] VANDERBECK, F. **Operations Research**. On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm, journal, v.48, n.1, p. pp. 111–128, 2000.

[61] VANDERBECK, F.; SAVELSBERGH, M. W. P. **Operations Research Letters**. A generic view of dantzig-wolfe decomposition for integer programming, journal, v.34, p. 296–306, May 2006.

[62] WOLSEY, L. A. **Integer Programming**. New York, NY, USA: Wiley-Interscience, 1998.