UNIVERSIDADE FEDERAL FLUMINENSE

TEOBALDO LEITE BULHÕES JÚNIOR

Column Generation over Set Partitioning Formulations: Theory and Practice

NITERÓI

2017

UNIVERSIDADE FEDERAL FLUMINENSE

TEOBALDO LEITE BULHÕES JÚNIOR

Column Generation over Set Partitioning Formulations: Theory and Practice

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science. Topic Area: Algorithms and Optimization.

Advisor:
Fábio Protti

Co-Advisor:
Eduardo Uchoa

NITERÓI

2017

TEOBALDO LEITE BULHÕES JÚNIOR

Column Generation over Set Partitioning Formulations: Theory and Practice

Thesis presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Doctor of Science. Topic Area: Algorithms and Optimization.

Approved by:

Prof. D.Sc. Fábio Protti / IC-UFF (President)

Prof. D.Sc. Eduardo Uchoa / TEP-UFF

Prof. D.Sc. Yuri Frota / IC-UFF

Prof. D.Sc. Luiz Satoru Ochi / IC-UFF

Prof. D.Sc. Artur Pessoa / TEP-UFF

Prof. D.Sc. Rafael Martinelli / DEI-PUC-Rio

Prof. Dr. Ricardo Corrêa / IM-UFRRJ

Niterói, December 19, 2017

To my parents.

Acknowledgments

I would like to express my sincere gratitude to:

- My family, in special to my parents and to my aunt Gracinha, for the support in my whole life.
- My dear girlfriend, Ana, for her love and friendship, and for understanding the reason for my absence.
- My advisors, Prof. Fábio Protti and Prof. Eduardo Uchoa, whom I admire both intellectually and personally. Thank you for the great opportunity of working with you and for countless valuable discussions. In special, I would like to thank Prof. Fábio Protti for being so kind in accepting integer programming as my major research field.
- Ruslan Sadykov, for being my unofficial advisor during the research reported in Chapter 3, as well as in other ongoing works. I also would like to thank Ruslan for the hospitality during my two stays in Bordeaux. I am in debt to Prof. Eduardo Uchoa who has provided me the great opportunity of visiting Bordeaux.
- Prof. Artur Pessoa, for being my unofficial advisor during the research reported in Chapter 2 and for joining me in countless "cafézinhos na School".
- Prof. Marcos Roboredo, for the friendship and for joining me in countless "cafézinhos na School".
- Prof. Anand Subramanian, Prof. Lucídio Cabral and Prof. Gilberto Farias, for advising me during my years at UFPB. In special, I would like to thank Prof. Anand Subramanian for being always around to help me and for introducing me to Prof. Eduardo Uchoa.
- My fellows at "República Ditatorial de Eyder Rios" (known in LOGIS as "Consulado da Paraíba"): Hugo Kramer (Mulo), Gilberto Farias (Chimba, Mamãe), Eduardo Queiroga (Edu), Thiago Gouveia (Gov boy), Eyder Rios (Papai) and Thaylon Guedes (Gigante guerreiro). Thank you for your friendship!

- Prof. Luiz Satoru, for the constant collaborations with Prof. Lucídio Cabral that opened doors to many students from UFPB, including me, Anand, Gilberto, Thiago and Eduardo.

- Thibaut Vidal and Renatha Capua, for the friendship. I also would like to thank Thibaut for the fruitful research collaborations.

- The Brazilian agency CAPES, for the scholarship grant.

Abstract

This thesis presents theoretical and practical contributions on the use of column generation over set partitioning formulations. On the theoretical side, the set packing and set partitioning polytopes associated with a binary n-row matrix having all possible $2^n - 1$ columns are analyzed. We show the precise relation between those polytopes: with very few exceptions, every facet-inducing inequality for the former polytope is also facet-inducing for the latter polytope, and vice-versa. Moreover, we characterize the multipliers that induce Chvátal-Gomory rank 1 clique facets and give several families of multipliers that yield other facets for arbitrarily large dimensions. On the practical side, we present a novel branch-and-price algorithm over a set partitioning formulation for the Minimum Latency Problem (MLP), a variant of the Traveling Salesman Problem. The proposed algorithm strongly relies on new features for the *ng*-path relaxation, the current best-performing path relaxation employed by many exact algorithms for routing problems. Such new features are not built over any strong assumption on the MLP and can be potentially used in several other problems. Computational experiments over TSPLIB instances are reported, showing that several instances not solved by the current state-of-the-art method can now be solved.

Keywords: Set Partitioning, Column Generation, Rank 1 Cuts, Polyhedral Combinatorics, Routing, *ng*-paths.

Resumo

Esta tese apresenta resultados teóricos e práticos a respeito do uso de geração de colunas a partir de formulações de particionamento de conjuntos. Na parte teórica, são estudados os poliedros de particionamento e empacotamento de conjuntos associados a uma matriz binária com n linhas e $2^n - 1$ colunas. Um forte relação entre esses poliedros é provada: com algumas poucas exceções, as inequações que induzem facetas desses poliedros são iguais. Além disso, caracterizam-se os multiplicadores que induzem as inequações de *clique* que podem ser obtidas por uma única aplicação do procedimento de arredondamento de Chvátal-Gomory (cortes de posto 1) , bem como apresentam-se famílias de multiplicadores que induzem facetas para dimensões arbitrárias (não necessariamente *cliques*). Na parte prática, é apresentado um algoritmo *branch-and-price* para o Problema da Latência Mínima (PLM), uma variante do Problema do Caixeiro Viajante. O algoritmo proposto utiliza uma formulação de particionamento de conjuntos e novas ideias para melhorar o desempenho da relaxação de caminho conhecida como *ng-path*. Essas novas ideias não são baseadas em nenhuma hipótese forte em relação ao PLM, tendo o potencial de serem aplicadas em outros problemas. Experimentos computacionais mostram que o algoritmo proposto supera o estado da arte, sendo capaz de resolver várias instâncias nunca antes resolvidas.

Palavras-chave: Particionamento de Conjuntos, Geração de Colunas, Cortes de posto 1, Combinatória Poliédrica, Roteamento, *ng-paths*.

List of Figures

List of Tables

Glossary

COP	Combinatorial Optimization Problem
VRP	Vehicle Routing Problem
CVRP	Capacitated Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows
MLP	Minimum Latency Problem
TSP	Traveling Salesman Problem
TSPLIB	Traveling Salesman Problem Library
TDTSP	Time Dependent Traveling Salesman Problem
ERCSPP	Elementary Resource Constrained Shortest Path Problem
RCSPP	Resource Constrained Shortest Path Problem
MPLD	Multiple Partial Label Dominance
DSSR	Decremental State-Space Relaxation
BP	Branch-and-Price
BKS	Best Known Solution

Contents

Chapter 1

Introduction

The general definition of many combinatorial optimization problems (COPs) can be stated as follows. We are given a finite set $\mathcal{N}$, a cost function $c : \mathcal{N} \to \mathbb{R}$ and a collection $\mathcal{F}$ of subsets of $\mathcal{N}$. Each set $F \in \mathcal{F}$ is composed of some elements $j \in \mathcal{N}$ that, together, satisfy a number of constraints and represents a feasible solution for the problem. The goal is to find a set $F \in \mathcal{F}$ whose total cost $\sum_{j \in F} c(j)$ is maximum (or minimum). For example, the classical COP of finding a minimum-cost spanning tree of a graph $G = (V, E)$ is modeled with $\mathcal{N} = E$ and $\mathcal{F} = \{F \subseteq E : F \text{ induces a spanning tree of } G\}$. Remark that in meaningful COPs brute force is not an option because $|\mathcal{F}|$ is often exponential on $|\mathcal{N}|$.

Another classical COP is the set partitioning problem, in which set $\mathcal{N}$ is composed of m binary n-dimensional columns and set $\mathcal{F}$ contains the subsets $F \subseteq \mathcal{N}$ such that each row i, $1 \leq 1 \leq n$, is covered by exactly one column in F. Let $A \in \{0,1\}^{n \times m}$ be a matrix whose columns are precisely those of $\mathcal{N}$ (in an arbitrary order). Therefore, set partitioning is equivalent to the following integer linear program:

$$\min \sum_{j=1}^{m} c_j x_j \tag{1.1}$$

$$\text{s.t.} \quad Ax = \mathbb{1}, \tag{1.2}$$

$$x \in \{0,1\}^m, \tag{1.3}$$

where $\mathbb{1}$ represents the n-dimensional all-ones vector and $c_j \in \mathbb{R}$ is the cost of the jth column of A. A closely related COP is the set packing problem, where each row is covered

at most once:

$$\min \sum_{j=1}^{m} c_j x_j \tag{1.4}$$

$$\text{s.t.} \quad Ax \leq \mathbb{1}, \tag{1.5}$$

$$x \in \{0,1\}^m. \tag{1.6}$$

It is well-known in the literature that problem (1.4)-(1.6) can be reduced to (1.1)-(1.3) by adding slack variables.

The set partitioning and set packing structures ($Ax = \mathbb{1}$ and $Ax \leq \mathbb{1}$) are among the most frequent ones in integer programming, and for many COPs they are almost enough for a complete formulation. For example, for solving vehicle routing or scheduling problems, columns may represent routes or schedules, respectively; for graph coloring problems, columns may represent stable sets; for bin packing problems, columns may represent the possible assignments of items to bins; and so on. In fact, the state-of-the-art exact methods for many COPs are based on set partitioning integer programming formulations — see, for instance, (Pecin et al., 2017a; Pecin et al., 2017c; Pessoa et al., 2010; Roberti and Mingozzi, 2014; Archetti et al., 2011; Desaulniers, 2010). As the number of variables in those formulations is very large, the linear programs in those algorithms are typically solved by column generation, a technique that implicitly considers the variables of the problem (see (Chen et al., 2009) for further details on integer programming and column generation).

This thesis is mainly motivated by vehicle routing problems (VRPs), a class of problems that model many practical scenarios in freight distribution and collection, transportation, garbage collection, newspaper delivery, etc. (Toth et al., 2014). In a VRP, one looks for a minimum-cost set of vehicle routes that serve a set $\mathcal{C}$ of geographically dispersed customers. A vehicle route (route, for short) starts in a depot, visits a sequence of customers, and then returns to a depot. For example, in the classical Capacitated VRP (CVRP), each customer $i \in \mathcal{C}$ has a demand $d_i \in \mathbb{Z}_+^*$ and a homogeneous fleet of K vehicles with capacity Q is based at a single depot. In addition, a cost $c_{ij} \in \mathbb{Z}_+$ is incurred whenever a vehicle traverses an arc (i, j) (from the depot to a customer, from a customer to the depot or from a customer to another customer). The goal is to find up to K routes such that:

- Each customer $i \in \mathcal{C}$ is visited exactly once.

- The total demand of the customers visited in a route does not exceed the vehicle capacity Q.
- The total cost, defined as the sum of the costs of the traversed arcs, is minimized.

Therefore, the only operational constraint modeled in CVRP is the vehicle capacity (see Figure 1.1). In another classical variant, the VRP with Time Windows (VRPTW), a vehicle is allowed to visit a customer $i \in \mathcal{C}$ only within a specific time window. Many other VRP variants abound in the literature, modeling different scenarios found in practice. See (Toth et al., 2014) for further details on VRP variants, applications and solution methods.

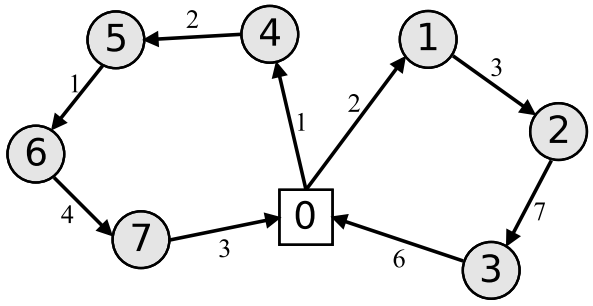


Figure 1.1: A CVRP solution with two routes. Vertex 0 represents the depot, while the other vertices represent the customers. The vehicle capacity is $Q = 20$ and the demands are $d_1 = 8$, $d_2 = 5$, $d_3 = 5$, $d_4 = 6$, $d_5 = 5$, $d_6 = 5$ and $d_7 = 3$. The cost of a route is given by the sum of the costs of the traversed arcs. Thus, the total cost of the routes are 11 and 18, and the solution cost is 29.

This thesis presents theoretical and practical contributions on the use of column generation over set partitioning formulations:

- Chapter 2 studies the complete set packing and set partitioning polytopes, which are both associated with a binary n-row matrix having all possible columns. Chvátal-Gomory rank 1 cuts for the latter polytope play a central role in recent column generation based algorithms for some COPs, in special VRPs. We show the precise relation between those polytopes, characterize the multipliers that induce rank 1 clique facets and give several families of multipliers that yield other facets for arbitrarily large dimensions. The results derived in that chapter are general and can be potentially used in several exact algorithms over set partitioning formulations.
- Chapter 3 presents a novel branch-and-price algorithm over a set partitioning formulation for the Minimum Latency Problem, a variant of the Traveling Salesman

Problem. The proposed algorithm strongly relies on new features for the *ng*-path relaxation, the route relaxation currently employed by many column generation based algorithms for VRPs. The new features are not built over any strong assumption about the Minimum Latency Problem and can be potentially used in several other COPs, specially in other VRPs.

- Chapter 4 presents the concluding remarks of this work.

Chapter 2

On the Complete Set Packing and Set Partitioning Polytopes

2.1 Introduction

Let A be a binary matrix with n rows and m columns. The set packing polytope associated with A, denoted as $SPP_{\leq}(A)$, is defined as the convex hull of the incidence vectors of integer solutions to the system $Ax \leq \mathbb{1}$, where $\mathbb{1}$ represents the n-dimensional all-ones vector and $x \in \{0,1\}^m$. Such solutions are equivalent to stable sets of the intersection graph G derived from A, i.e., a graph whose vertices represent columns of A and whose edges indicate non-orthogonality between those columns. Therefore, we also denote this polytope as $SPP_{\leq}(G)$. A closely related polytope is the set partitioning polytope, defined analogously with respect to the system $Ax = \mathbb{1}$.

Since the seventies, many authors have studied $SPP_{\leq}(G)$, proposing facet-inducing inequalities associated with specific graphs: cliques (Padberg, 1973), odd holes (Padberg, 1973), odd anti-holes (Nemhauser and Trotter, 1974), webs (Trotter, 1975), anti-webs (Trotter, 1975), $K_{1,3}$-free graphs (Giles and Trotter, 1981), wheels (Cheng and Cunningham, 1997), antiweb-wheels (Cheng and Vries, 2002) and grilles (Cánovas et al., 2000). For some graphs, such as perfect graphs, series-parallel graphs and graphs that do not have a 4-wheel as a minor, complete characterizations of $SPP_{\leq}(G)$ are known (Chvátal, 1975; Mahjoub, 1988; Barahona and Mahjoub, 1994a; Barahona and Mahjoub, 1994b). Facet-inducing inequalities with binary coefficients were studied in (Chvátal, 1975; Balas and Zemel, 1977b). Facet-generating procedures for $SPP_{\leq}(G)$ were described in (Nemhauser and Trotter, 1974; Chvátal, 1975; Wolsey, 1976; Padberg, 1977; Balas and Zemel, 1977a; Barahona and Mahjoub, 1994a; Cánovas et al., 2000; Rossi and Smriglio, 2001; Reben-

nack et al., 2011; S. Xavier and Campêlo, 2011; Corrêa et al., 2017). In contrast, the set partitioning polytope is seldom studied in the literature (see for instance (Balas and Padberg, 1976; Balas, 1977; Sherali and Lee, 1996)) due to its more complex structure — even computing its dimension is an NP-Hard problem.

In this chapter, we study the complete set packing and set partitioning polytopes, respectively $CSPP_{\leq}(n)$ and $CSPP_{=}(n)$, which are both associated with a binary n-row matrix A having all possible $(2^n - 1)$ non-zero columns. While $CSPP_{=}(n)$ has already been defined in (Pecin et al., 2017b), as far as we know, $CSPP_{\leq}(n)$ is studied for the first time in this work. The definitions for both polytopes are formalized as follows.

$$CSPP_{=}(n) = \text{Conv}\left\{\sum_{j=1}^{2^n-1} b^j \lambda_j = \mathbb{1}, \lambda \in \{0,1\}^{2^n-1}\right\}$$

$$CSPP_{\leq}(n) = \text{Conv}\left\{\sum_{j=1}^{2^n-1} b^j \lambda_j \leq \mathbb{1}, \lambda \in \{0,1\}^{2^n-1}\right\}$$

where b^j is the column associated with the binary representation of j. For example, if $n = 3$, then $b^1 = (1,0,0)^T$, $b^2 = (0,1,0)^T$, $b^3 = (1,1,0)^T$, etc. A column $e^i = b^{2^{i-1}}$, $1 \leq i \leq n$, is the *singleton* column associated to row i. Let $\bar{b}^j = \mathbb{1} - b^j = b^{2^n-1-j}$ be the complement of b^j. Also, we define $B = \bigcup_{j=1}^{2^n-1} b^j$, the set of all columns, and we denote a solution to the complete set partitioning (set packing) problem by a subset s of columns whose incidence vector belongs to $CSPP_{=}(n)$ ($CSPP_{\leq}(n)$). Moreover, we use $\hat{e}^j$ to denote the incidence vector of solution $\{b^j\}$.

On the practical side, our study of those polytopes is motivated by the many applications that can be modeled as set packing/partitioning problems with a very large number of columns, where explicit representation of the coefficient matrix is unpractical and the linear relaxations have to be solved by column generation. In that context, any cutting plane should have a well-defined coefficient for every possible column, since it is not possible to predict which columns will be generated. In other words, cuts should be valid for $CSPP_{=}(n)/CSPP_{\leq}(n)$. On the theoretical side, the "complete" structure facilitates the study of those polytopes, allowing strong properties to be derived. In this regard, it is worth mentioning the works of Araóz, (1974) and Dash et al., (2010) on the master knapsack polyhedron $K(r)$ (or generalizations of it), another polyhedron with "complete" structure:

$$K(r) = \text{Conv}\left\{x \in \mathbb{Z}_+^r : \sum_{i=1}^{r} i x_i = r\right\}$$

where r is a positive integer.

A typical application modeled as a set partitioning problem is the VRP, where one looks for a minimum-cost set of routes that serve a set of customers $\mathcal{C}$. Those routes should respect operational constraints, that vary according to the considered variant. Let Ω, c_r and a_i^r denote, respectively, the set of feasible routes, the cost of route r, and the number of times route r visits customer i. The set partitioning formulation of the VRP follows:

$$\min \sum_{r \in \Omega} c_r \lambda_r \tag{2.1}$$

$$\text{s.t.} \quad \sum_{r \in \Omega} a_i^r \lambda_r = 1, \qquad \forall i \in \mathcal{C}, \tag{2.2}$$

$$\lambda_r \in \{0, 1\}, \qquad \forall r \in \Omega. \tag{2.3}$$

State-of-the-art exact algorithms for many VRPs, including its most classical variants, the CVRP the VRPTW, are based on a combination of column and cut generation over the above formulation. However, the addition of general cuts for $CSPP_{=}(n)$ has the serious drawback of complicating a lot the pricing problem (i.e., the column generation subproblem), making the algorithm unpractical. Jepsen et al., (2008) realized that some cuts with Chvátal-Gomory rank 1 could be better treated in the pricing. Recently, Pecin et al., (2017a) introduced the so-called limited-memory technique, for further minimizing the impact in the pricing of rank 1 cuts. This lead to big improvements in the performance of exact algorithms for CVRP (Pecin et al., 2017a) and VRPTW (Pecin et al., 2017c), more than doubling the size of the instances that can be solved. This motivated Pecin et al., (2017b) to determine computationally 9 sets of rational multipliers that are capable of generating all cuts of rank 1 that induce facets of $CSPP_{=}(n)$, for $n \leq 5$. The authors argued that the new multipliers contributed decisively for solving a previously open CVRP instance with 420 customers, the largest classical instance ever solved. However, that computational "brute force" approach breaks for $n > 5$.

This chapter is a theoretical analysis of $CSPP_{=}(n)$, aimed at finding infinite families of multipliers that produce facets for arbitrarily large values of n. First, we prove a very strong relationship between $CSPP_{=}(n)$ and $CSPP_{\leq}(n)$ (Section 2.3). More specifically, we show that, with very few exceptions, every facet-inducing inequality for the former polytope is also facet-inducing for the latter polytope, and vice-versa. This essentially means that the study of $CSPP_{=}(n)$ can be reduced to the study of the simpler $CSPP_{\leq}(n)$ polytope. Second, we characterize a set of multipliers that can induce all

rank 1 clique inequalities for those polytopes (Section 2.4.1). Finally, we propose 7 families of multipliers that generalize the 9 sets of multipliers found by Pecin et al., (2017b) and prove that they are facet-defining (Section 2.4.2). However, before proceeding to the results, we briefly discuss rank 1 cuts in the next section.

2.2 Chvátal-Gomory Rank 1 Cuts

Given an integer polyhedron $P_I = \text{Conv}\{x \in \mathbb{Z}^q \mid Hx \leq h\}$ with $H \in \mathbb{Z}^{p \times q}$ and $h \in \mathbb{Z}^p$, a Chvátal-Gomory rank 1 cut, introduced in (Chvátal, 1973) and (Gomory, 1963), is a valid cut for P_I of the form $\lfloor u^T H \rfloor x \leq \lfloor u^T h \rfloor$, where $u \in \mathbb{R}^p_+$ and $\lfloor \rfloor$ denotes a component-wise round-down operator when its argument is a vector. Although any $u \in R^p_+$ is valid, it is well-known in the literature that rational multipliers lying in $[0, 1)^p$ are sufficient to generate any non-dominated rank 1 cut (Caprara and Fischetti, 1996). The rank 1 closure of the relaxed polyhedron $P_R = \{x \in \mathbb{R}^q \mid Hx \leq h\}$ is the polyhedron $P_{R1} = \{x \in P_R \mid \lfloor u^T H \rfloor x \leq \lfloor u^T h \rfloor, \forall u \in [0, 1)^p\}$. Clearly, $P_I \subseteq P_{R1} \subseteq P_R$.

Some complexity results concerning rank 1 cuts have been proved in the last decades. Eisenbrand, (1999) proved that, in general, it is NP-Hard to separate them. Earlier, Caprara and Fischetti, (1996) had already shown that if one restricts the multipliers to values in $\{0, \frac{1}{2}\}$, the general separation problem of the so-called $\{0, \frac{1}{2}\}$-cuts is still NP-Hard, but there are some polynomially solvable cases. Later, Letchford et al., (2011) showed that $\{0, \frac{1}{2}\}$-cuts separation remains NP-Hard in case matrix H is binary. Also, the authors pointed out that their proof can be directly extended to the set packing case (H binary, h all-ones vector). Finally, Dunkel and Schulz, (2013) proved that the separation of rank 1 cuts with binary coefficients is NP-Hard.

Recently, rank 1 cuts have been extensively used in exact algorithms over the VRP formulation (2.1)-(2.3). This formulation is often obtained by applying a Dantzig-Wolfe decomposition to a flow formulation defined over arc variables. In this case, the pricing subproblem corresponds to finding a path in a network where arc costs are reduced costs. According to the classification proposed in (Poggi de Aragão and Uchoa, 2003), cuts over the arc variables are *robust*, since they can be translated into reduced costs, not changing the pricing structure. On the other hand, cuts defined over the path variables λ, such as rank 1 cuts, qualify as *non-robust*: each additional cut of this type requires new dimensions in the dynamic programming labeling algorithm that is typically used to solve the pricing subproblem.

Even though they are classified as non-robust, rank 1 cuts are attractive in practice because a single additional dimension is required for each cut added. Let $P = (v_0, v_1, \ldots, v_p)$ be a non-dominated partial path stored by the labeling algorithm and consider the rank 1 cut induced by multipliers $\frac{\alpha_i}{\beta}, i \in \mathcal{C}$. Remark that the coefficient of P in this cut is given by $\lfloor \sum_{i=1}^{p} \frac{\alpha_{v_i}}{\beta} \rfloor$ (vertex v_0 is ignored because it represents the depot). The new dimension keeps track of the *state* of P in the cut, which is zero for an empty path. When a new visit to a customer v_i is performed, α_{v_i} is added to the state of P. If the new state reaches a value greater than or equal to β, the coefficient of P in the cut is increased by one unit and the state of P is decreased by β units. In other words, the state of P equals the remainder of the integer division $\frac{\sum_{i=1}^{p} \alpha_{v_i}}{\beta}$. Another attractive feature of rank 1 cuts is that they are characterized solely by the multipliers, so in an implementation there is no need for explicitly storing the coefficients of the columns.

The limited-memory rank 1 cuts proposed by Pecin et al., (2017a) is a rank 1 cut with an associated memory $S \subseteq \mathcal{C}$. The idea is to set the state to zero if the new visit is outside the memory S (see Algorithm 1). The smaller the memory, the smaller the impact of the cut on the labeling algorithm, but also the smaller the coefficient of P in the cut. Hence, those cuts are a weakening of the original rank 1 cuts (those later cuts are obtained with a full memory $S = \mathcal{C}$). However, after some rounds of cut separation, the memories get adjusted and eventually the same bounds are obtained (Pecin et al., 2017a). The impact on the labeling algorithm can be further reduced by defining the memories in terms of arcs instead of nodes, at the expense of a good convergence to strong bounds (Pecin et al., 2017c).

Algorithm 1 Computing the coefficient of a partial path in a limited-memory rank 1 cut

```
1: procedure(P = (v_0, v_1, ..., v_p), α, β, S)
2:     state ← 0, coeff ← 0
3:     for i ← 1 to p do
4:         if v_i ∉ S then
5:             state ← 0
6:         else
7:             state ← state + α_{v_i}
8:             if state ≥ β then
9:                 state ← state − β, coeff ← coeff + 1
10: return coeff
```

2.3 Properties

In this section we show some properties of the polytopes at stake. The main property is presented in Theorem 2.1, which states that, with very few exceptions, every facet-inducing inequality for $CSPP_{=}(n)$ is also facet-inducing for $CSPP_{\leq}(n)$, and vice-versa. However, we first show some building blocks of this theorem.

Definition 2.3.1. *The intersection graph associated with $CSPP_{=}(n)$ and $CSPP_{\leq}(n)$, denoted as G_n, is a graph where each vertex represents a column b^j and an edge exists iff the columns represented by its endpoints are non-orthogonal (see Figure 2.1).*

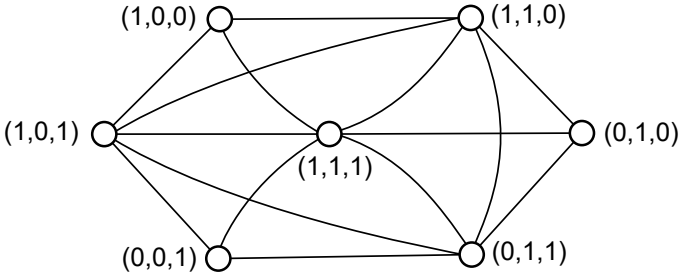


Figure 2.1: Graph G_3.

For a set of columns $S \subseteq B$, we define $\phi(S) \subseteq B$ as the set of all singleton columns that are orthogonal to all columns in S, and $\varphi(b^j) \subseteq B$ as the set of the singleton columns that are non-orthogonal to b^j. Let also $\mathcal{K}_{=}$ be the set of the n maximal cliques of G_n that contain singleton columns and let $\mathcal{K}_{\leq}$ be the set of all other maximal cliques of G_n.

Lemmas 1 to 3 state some basic properties of $CSPP_{=}(n)$ and $CSPP_{\leq}(n)$.

Lemma 1. *The dimension of $CSPP_{=}(n)$ is $2^n - n - 1$.*

Proof. The incidence vectors of the following $2^n - n$ solutions are clearly linearly independent:

- $s_j = \{b^j\} \cup \phi(\{b^j\})$, $1 \leq j \leq 2^n - 1$ and b^j is not a singleton column,
- $s = \bigcup_{k=1}^{n} \{e^k\}$.

□

Lemma 2. *Let $a^T\lambda \leq a_0$ be facet-inducing for $CSPP_{=}(n)$. For every column b^j holds that $a_j \geq \sum_{b^i \in \varphi(b^j)} a_i$.*

Proof. Since $a^T\lambda \leq a_0$ induces a facet, there should be a set of columns S such that $\{b^j\} \cup S$ is a feasible solution and $a_j + \sum_{b^i \in S} a_i = a_0$, otherwise a_j could be increased without cutting off any feasible solution, which is a contradiction to the facetness of $a^T\lambda \leq a_0$. Hence, if $a_j < \sum_{b^i \in \varphi(b^j)} a_i$, the feasible solution $\varphi(b^j) \cup S$ would be cut off by $a^T\lambda \leq a_0$. □

Lemma 3. *Any inequality $a^T\lambda \leq a_0$ that is facet-inducing for $CSPP_{=}(n)$ can be rewritten as $\alpha^T\lambda \leq \alpha_0$ such that:*

- $\alpha_j = 0$ *for every singleton column* b^j,
- $\alpha_j \geq 0$ *for every column* b^j,
- $\alpha_0 \geq 0$.

Moreover, in this form, such a facet is a valid inequality for $CSPP_{\leq}(n)$.

Proof. This proof is algorithmic. Start with $\alpha_j = a_j$ for $j = 1, \ldots, 2^n - 1$. For each singleton column b^j, do the following. If $\alpha_j > 0$, subtract α_j times the equality containing b^j from $\alpha^T\lambda \leq \alpha_0$. If $\alpha_j < 0$, add $|\alpha_j|$ times the equality containing b^j to $\alpha^T\lambda \leq \alpha_0$. Now that $\alpha_j = 0$ for every singleton column b^j, it follows from Lemma 2 that $\alpha_i \geq 0$ for every column b^i. Moreover, $\alpha_0 \geq 0$, otherwise any feasible solution would be cut off.

Now, it remains to prove that $\alpha^T\lambda \leq \alpha_0$ is satisfied by any vertex of $CSPP_{\leq}(n)$. For that, let λ be a vertex of $CSPP_{\leq}(n)$, and λ' be the vertex of $CSPP_{=}(n)$ obtained from λ by increasing the coordinates of singleton columns until $\sum_{j \in K} \lambda'_j = 1$ for all $K \in \mathcal{K}_{=}$. Since $\alpha_j = 0$ for every singleton column b^j, we have that $\alpha^T\lambda = \alpha^T\lambda' \leq \alpha_0$. □

Next, we introduce some facet-inducing inequalities for $CSPP_{=}(n)$ that are necessary to establish the main property.

Lemma 4. *The non-negativity inequality $\lambda_j \geq 0$ defines a facet of $CSPP_{=}(n)$ iff b^j is not a singleton column.*

Proof. If b^j is not a singleton column, then the incidence vectors of the following $2^n - n - 1$ solutions satisfy $\lambda_j \geq 0$ at equality and are clearly linearly independent:

- $s_i = \{b^i\} \cup \phi(\{b^i\})$, $1 \leq i \leq 2^n - 1$, $b^i \neq b^j$ and b^i is not a singleton column,
- $s = \bigcup_{k=1}^{n} \{e^k\}$.

It remains to prove that $\lambda_j \geq 0$ does not define a facet when b^j is a singleton column. Let $K \in \mathcal{K}_=$ be the maximal clique of G_n that contains b^j, and K' be the clique obtained by replacing b^j by $\bar{b}^j = b^{2^n-1-j}$ in K. The sum of $-\sum_{v\in K} \lambda_v = -1$, $\sum_{v\in K'} \lambda_v \leq 1$ and $-\lambda_{(2^n-1-j)} \leq 0$ equals $\lambda_j \geq 0$. □

Lemma 5. *Let $K \in \mathcal{K}_\leq$ be a maximal clique of G_n. Then, the clique inequality*

$$\sum_{v\in K} \lambda_v \leq 1 \tag{2.4}$$

is valid and defines a facet of $CSPP_=(n)$.

Proof. For every non-singleton column b^j, consider the following solution:

$$s_j = \begin{cases} \{b^j\} \cup \phi(\{b^j\}) & \text{if } b^j \in K \\ \{b^j\} \cup \{b^i\} \cup \phi(\{b^j, b^i\}) & \text{if } b^j \notin K \end{cases}$$

where b^i is any column in K such that b^i and b^j are orthogonal. There exists at least one such column otherwise one could augment K by adding b^j to it. The incidence vectors of these $2^n - n - 1$ solutions satisfy (2.4) at equality and are clearly linearly independent. □

The following lemma presents an useful link between $CSPP_=(n)$ and $CSPP_\leq(n)$.

Lemma 6. *Let x and y be two points in $\mathbb{R}_+^{2^n-1}$ such that $x_i \geq y_i$, $1 \leq i \leq 2^n - 1$. If $x \in CSPP_=(n)$, then $y \in CSPP_\leq(n)$.*

Proof. Let $u^1, \ldots, u^p$ be the vertices of $CSPP_=(n)$. As $x \in CSPP_=(n)$, there exist $\alpha_1, \ldots, \alpha_p \in \mathbb{R}_+$ such that $x = \sum_{i=1}^p \alpha_i u^i$ and $\sum_{i=1}^p \alpha_i = 1$. Also, let $v^1, \ldots, v^p, v^{p+1}, \ldots, v^q$ be the vertices of $CSPP_\leq(n)$, where $v_i = u_i$ for $1 \leq i \leq p$. Algorithm 2 computes $\beta_1, \ldots, \beta_q \in \mathbb{R}_+$ such that $y = \sum_{i=1}^q \beta_i v^i$ and $\sum_{i=1}^q \beta_i = 1$. This algorithm starts with the vector z, defined as $\beta_1 v^1 + \ldots + \beta_q v^q$, equal to x (lines 1-3). Then, it iteratively decreases the value of each component $z_j > y_j$ without changing the other components of z (lines 5-11) as follows. First, it chooses arbitrarily a vertex v^r of $CSPP_\leq(n)$ with $\beta_r > 0$ that includes the column b^j (line 6). Let v^s be the vertex of $CSPP_\leq(n)$ that results from removing the column b^j from v^r (line 8). The value of z_j is decreased by moving a portion Δ of the coefficient β_r of v^r to the coefficient β_s of v^s (lines 9-11). The algorithm terminates in a finite number of steps because the value of Δ is calculated (line 7) in such a way that, in the next iteration of the while loop, either z_j becomes equal to y_j or the number of non-zero values decreases in the set $\{\beta_r \mid v^r \text{ includes the column } b^j\}$. Also, if all values in the latter set are equal to zero, then $z_j = y_j = 0$. □

Algorithm 2 Computing $(\beta_1, \ldots, \beta_q)$ from $(\alpha_1, \ldots, \alpha_p)$

$\beta_i \leftarrow \alpha_i, \forall i \in \{1, \ldots, p\}$
$\beta_i \leftarrow 0, \forall i \in \{p+1, \ldots, q\}$
$z \leftarrow x$
for $j \in \{1, \ldots, 2^n - 1\}$ **do**
  while $z_j > y_j$ **do**
    Choose v^r such that $v^r_j = 1$ and $\beta_r > 0$
    $\Delta \leftarrow \min\{z_j - y_j, \beta_r\}$
    Let v^s be the vertex $v^r - \hat{e}^j$
    $\beta_r \leftarrow \beta_r - \Delta$
    $\beta_s \leftarrow \beta_s + \Delta$
    $z \leftarrow \beta_1 v^1 + \ldots + \beta_q v^q$
return $(\beta_1, \ldots, \beta_q)$

Now we are ready to prove the main property presented in this section.

Theorem 2.1. *Let*

$$\sum_{v \in K} \lambda_v = 1, \forall K \in \mathcal{K}_=, \tag{2.5}$$

$$\sum_{v \in K} \lambda_v \leq 1, \forall K \in \mathcal{K}_\leq, \tag{2.6}$$

$$H\lambda \leq h, \tag{2.7}$$

$$\lambda_j \geq 0, \forall j \in \{1, \ldots, 2^n - 1\} \setminus \{2^0, \ldots, 2^{n-1}\}, \tag{2.8}$$

be a minimal description of $CSPP_=(n)$ where inequalities (2.7) *are of the form described in Lemma 3. Then,*

$$\sum_{v \in K} \lambda_v \leq 1, \forall K \in \mathcal{K}_= \cup \mathcal{K}_\leq, \tag{2.9}$$

$$H\lambda \leq h, \tag{2.10}$$

$$\lambda_j \geq 0, \forall j \in \{1, \ldots, 2^n - 1\}, \tag{2.11}$$

is a minimal description of $CSPP_\leq(n)$.

Proof. First, we prove that (2.9)-(2.11) suffices to describe $CSPP_\leq(n)$. Let $\lambda' \in \mathbb{R}^{2^n-1}$ be a point that satisfies (2.9)-(2.11). Also, let $\lambda'' \in \mathbb{R}^{2^n-1}$ be a point obtained from λ' by increasing the coordinates corresponding to singleton columns so as to guarantee that λ'' satisfies (2.5). Clearly, λ'' satisfies (2.8). For every inequality $a^T\lambda \leq a_0$ from (2.6)-(2.7), we know that $a_j = 0$ for every singleton column b^j. Thus, λ'' also satisfies (2.6)-(2.7) and we can conclude that $\lambda'' \in CSPP_=(n)$. Since $\lambda', \lambda'' \geq 0$, $\lambda'' \geq \lambda'$ and $\lambda'' \in CSPP_=(n)$, by Lemma 6 we have that $\lambda' \in CSPP_\leq(n)$. Therefore, (2.9)-(2.11) are sufficient to describe $CSPP_\leq(n)$.

Now, let us show that this description is minimal. First, note that (2.9) and (2.11) are necessary to the description because cliques and non-negativity inequalities define facets of $CSPP_{\leq}(n)$. Now suppose that an inequality $a^T\lambda \leq a_0$ from (2.10), which induces a facet of $CSPP_{=}(n)$ by definition, is redundant for $CSPP_{\leq}(n)$. By the facetness of $a^T\lambda \leq a_0$, there exists a point $\lambda' \in \mathbb{R}^{2^n-1}$ that is cut off from $CSPP_{=}(n)$ only by this inequality. Since $a^T\lambda \leq a_0$ is valid for $CSPP_{\leq}(n)$, we have that $\lambda' \notin CSPP_{\leq}(n)$. By the proof of Lemma 4, if $\lambda' \notin \mathbb{R}_{+}^{2^n-1}$ it would be cut off by (2.5), (2.6) or (2.8). As a result, $\lambda' \in \mathbb{R}_{+}^{2^n-1}$, and thus, is not cut off by (2.11). Moreover, it is not cut off by (2.9), since such inequalities represent a relaxation of (2.5)-(2.6). Therefore, λ' must be cut off from $CSPP_{\leq}(n)$ only by $a^T\lambda \leq a_0$, which is a contradiction to the redundancy of this inequality. □

Corollary 1. *Every facet-inducing inequality for $CSPP_{\leq}(n)$ is also facet-inducing for $CSPP_{=}(n)$, unless it is a clique inequality derived from $K_{=}$ or a non-negativity inequality for a singleton column.*

2.4 Rank 1 Facets

We now focus on the rank 1 facets of $CSPP_{\leq}(n)$ and $CSPP_{=}(n)$, which are very useful in practice because they can be handled (to some extent) efficiently in column generation algorithms.

2.4.1 Cliques

The following two lemmas characterize the multipliers that induce rank 1 clique inequalities.

Lemma 7. *Any set of multipliers $u = (\frac{\alpha_1}{\beta}, \frac{\alpha_2}{\beta}, \ldots, \frac{\alpha_n}{\beta})$, with $\alpha_1, \alpha_2, \ldots, \alpha_n$ and β integers, such that $\sum_{i=1}^{n} \alpha_i = 2\beta - 1$ induces a clique inequality.*

Proof. Since $\sum_{l=1}^{n} u_l < 2$, any two columns b^i and b^j such that $\sum_{l=1}^{n} u_l b_l^i \geq 1$ and $\sum_{l=1}^{n} u_l b_l^j \geq 1$ should be non-orthogonal. Therefore, the set of columns with a non-zero coefficient in the corresponding rank 1 cut induces a clique in the intersection graph. It remains to show that this clique is maximal. Let b^k be a column with a zero coefficient. Thus, $\sum_{l=1}^{n} \alpha_l b_l^k < \beta$, which implies $\sum_{l=1}^{n} \alpha_l (1 - b_l^k) \geq \beta$. In other words, the complement of b^k has a non-zero coefficient in the rank 1 cut. Thus, b^k cannot be added to the clique. □

Lemma 8. *Every rank 1 clique inequality $a^T\lambda \leq 1$ can be generated by multipliers $u = (\frac{\alpha_1}{\beta}, \frac{\alpha_2}{\beta}, \ldots, \frac{\alpha_n}{\beta})$, with $\alpha_1, \alpha_2, \ldots, \alpha_n$ and β integers, such that $\sum_{i=1}^{n} \alpha_i = 2\beta - 1$.*

Proof. It is well known that the rational multipliers suffice to generate all rank 1 cuts. Let $u' = (\frac{\alpha'_1}{\beta}, \frac{\alpha'_2}{\beta}, \ldots, \frac{\alpha'_n}{\beta})$ be a set of multipliers inducing a clique inequality $a^T\lambda \leq 1$ such that $\sum_{i=1}^{n} \alpha'_i < 2\beta - 1$, and let K be the maximal clique of G_n associated with $a^T\lambda \leq 1$. Now consider a set of multipliers $u'' = (\frac{\alpha''_1}{\beta}, \frac{\alpha''_2}{\beta}, \ldots, \frac{\alpha''_n}{\beta})$ such that $\alpha''_i \geq \alpha'_i$, $1 \leq i \leq n$, and $\sum_{i=1}^{n} \alpha''_i = 2\beta - 1$, and let $d^T\lambda \leq 1$ be the inequality induced by u''. We argue that $a = d$. Clearly, if $a_j = 1$, then $d_j = 1$ since $\alpha''_i \geq \alpha'_i$, $1 \leq i \leq n$. Also, if $a_j = 0$, then $d_j = 0$, otherwise K is a not a maximal clique of G_n. □

2.4.2 Generalization of Known Facets

Pecin et al., (2017b) obtained computationally the following set of multipliers that, together with their permutations, induces all rank 1 facets of $CSPP_{=}(n)$, with $n \leq 5$.

- $n = 3 : \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$
- $n = 4 : \left(\frac{2}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right), \left(0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$
- $n = 5 : \left(0, 0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right), \left(0, \frac{2}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right), \left(\frac{2}{4}, \frac{2}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right), \left(\frac{3}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right), \left(\frac{3}{5}, \frac{2}{5}, \frac{2}{5}, \frac{1}{5}, \frac{1}{5}\right), \left(\frac{2}{3}, \frac{2}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right), \left(\frac{3}{4}, \frac{3}{4}, \frac{2}{4}, \frac{2}{4}, \frac{1}{4}\right), \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right), \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$

In this section we generalize those multipliers to arbitrarily large n. The results will be shown for $CSPP_{\leq}(n)$, and then extended to $CSPP_{=}(n)$ through Theorem 2.1. The inequalities described in Part (a) of Theorem 2.2 were already considered in (Pecin et al., 2017c), where the authors showed that they dominate the elementary cuts proposed by Balas, (1977).

Theorem 2.2. *The following multipliers induce rank 1 facets of $CSPP_{\leq}(n)$.*
(a) $\left(\frac{n-2}{n-1}, \frac{1}{n-1}, \frac{1}{n-1}, \ldots, \frac{1}{n-1}\right), n \geq 4$
(b) $\left(\frac{n-3}{n-1}, \frac{2}{n-1}, \frac{1}{n-1}, \frac{1}{n-1}, \ldots, \frac{1}{n-1}\right), n \geq 5$
(c) $\left(\frac{n-2}{n}, \frac{2}{n}, \frac{2}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right), n \geq 5$
(d) $\left(\frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2}\right)$, *if n is odd*
(e) $\left(\frac{1}{3}, \frac{1}{3}, \ldots, \frac{1}{3}\right)$, *if $n = 3k + 2, k \in \mathbb{N}, k \geq 1$*
(f) $\left(\frac{n-3}{n-2}, \frac{n-3}{n-2}, \frac{2}{n-2}, \frac{1}{n-2}, \ldots, \frac{1}{n-2}\right), n \geq 5$

(g) $\left(\frac{n-2}{n-1}, \frac{n-2}{n-1}, \frac{2}{n-1}, \frac{2}{n-1}, \frac{1}{n-1}, \dots\right), n \geq 5$

In the proofs of Parts (d) and (e), we define $B_t \subseteq B$ as the set of columns with exactly t entries equal to 1. In the proofs of Parts (f) and (g), we define R_t as the set of columns with coefficient t in the corresponding rank 1 cut.

Proof. By Lemma 7, we know that (a), (b) and (c) induce clique inequalities, which are well-known facets of $CSPP_{\leq}(n)$. In the proofs of (d)-(g), we denote by $a^T\lambda \leq a_0$ the inequality induced by the corresponding vector of multipliers u and we define $F_a = \{\lambda' \in CSPP_{\leq}(n) \mid a^T\lambda' = a_0\}$. In addition, we define $\alpha^T\lambda \leq \alpha_0$ as a facet-inducing inequality for $CSPP_{\leq}(n)$ such that $F_a \subseteq F_\alpha$, where $F_\alpha = \{\lambda' \in CSPP_{\leq}(n) \mid \alpha^T\lambda' = \alpha_0\}$. The proofs show that $\alpha = \gamma a$ for some $\gamma \in \mathbb{R}$.

Part (d): Let $b^i, b^j \in B_2$ be distinct non-orthogonal columns. Define the solutions $s_1 = \{b^i\} \cup S$ and $s_2 = \{b^j\} \cup S$, where $S \subseteq B_2$ is a set composed of $\frac{n-3}{2}$ orthogonal columns. The incidence vectors λ_1 and λ_2 associated with s_1 and s_2 lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i = \alpha_j$. Suppose now that b^i and b^j are orthogonal. Define $S \subseteq B_2$ as a set composed of $\frac{n-5}{2}$ columns such that all columns in $S \cup \{b^i, b^j\}$ are orthogonal to each other. Remark that there is a single row p not covered by any column in $S \cup \{b^i, b^j\}$. Define $b^{i'}$ as a column in B_2 such that $b^{i'}_p = 1$ and $b^i \cdot b^{i'} = 1$, and $b^{j'}$ as a column in B_2 such that $b^{j'}_p = 1$ and $b^j \cdot b^{j'} = 1$. Therefore, $s_1 = \{b^i, b^{j'}\} \cup S$ and $s_2 = \{b^j, b^{i'}\} \cup S$ are feasible solutions. Also, the incidence vectors λ_1 and λ_2 associated with s_1 and s_2 lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i + \alpha_{j'} = \alpha_j + \alpha_{i'}$. Since $b^{j'} \cdot b^{i'} = 1$, we have that $\alpha_{j'} = \alpha_{i'}$, and we finally conclude that $\alpha_i = \alpha_j$. Therefore, there exists $\gamma \in \mathbb{R}$ such that $\alpha_j = \gamma$, for all $b^j \in B_2$.

Now let b^j be a column in B_t with $t \neq 2$. Define the solution $s_1 = \{b^j\} \cup S$, where S is a set composed of $\lfloor \frac{n-t}{2} \rfloor$ orthogonal columns in B_2. Also, let s_2 be the solution obtained from s_1 by replacing b^j by a set of $\lfloor \frac{t}{2} \rfloor$ orthogonal columns in B_2, all of them orthogonal to b^j. The incidence vectors λ_1 and λ_2 associated with s_1 and s_2 lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$. This implies that $\alpha_j = \gamma\lfloor \frac{t}{2} \rfloor$. Hence, we have shown that $\alpha = \gamma a$.

Part (e): First, we show that $\alpha_i = \alpha_j = \gamma$ for any two columns $b^i, b^j \in B_3$. We consider three cases.

(I) $b^i \cdot b^j = 2$. In this case, there are $n-4$ rows not covered by b^i or b^j. Since $n = 3k+2$ for some integer $k \geq 1$, $\lfloor \frac{n-4}{3} \rfloor = \lfloor \frac{n}{3} \rfloor - 1$. Therefore, it is possible to define a set

$S \subseteq B_3$ of $\lfloor \frac{n}{3} \rfloor - 1$ columns such that solutions $s_1 = S \cup \{b^i\}$ and $s_2 = S \cup \{b^j\}$ are feasible. Also, the incidence vectors λ_1 and λ_2 associated with s_1 and s_2 lie in F_a, and thus $\alpha^T \lambda_1 = \alpha^T \lambda_2$, which implies $\alpha_i = \alpha_j$.

(II) $b^i \cdot b^j = 1$. In this case, there are $n-5$ rows not covered by b^i or b^j. Since $n = 3k+2$ for some integer $k \geq 1$, $\lfloor \frac{n-5}{3} \rfloor = \lfloor \frac{n}{3} \rfloor - 1$. Therefore, it is possible to define a set $S \subseteq B_3$ of $\lfloor \frac{n}{3} \rfloor - 1$ columns such that solutions $s_1 = S \cup \{b^i\}$ and $s_2 = S \cup \{b^j\}$ are feasible. Also, the incidence vectors λ_1 and λ_2 associated with s_1 and s_2 lie in F_a, and thus $\alpha^T \lambda_1 = \alpha^T \lambda_2$, which implies $\alpha_i = \alpha_j$.

(III) $b^i \cdot b^j = 0$ (only if $n \geq 8$). In this case, there are $n-6$ rows not covered by b^i or b^j. Since $n = 3k+2$ for some integer $k \geq 2$, $\lfloor \frac{n-6}{3} \rfloor = \lfloor \frac{n}{3} \rfloor - 2$. Therefore, it is possible to define a set $S \subseteq B_3$ of $\lfloor \frac{n}{3} \rfloor - 2$ orthogonal columns that are also orthogonal to b^i and b^j. Remark that there are two rows p and q not covered by any column in $S \cup \{b^i, b^j\}$. Define $b^{i'}$ as a column in B_3 such that $b^{i'}_p = b^{i'}_q = 1$ and $b^i \cdot b^{i'} = 1$, and $b^{j'}$ as a column in B_3 such that $b^{j'}_p = b^{j'}_q = 1$ and $b^j \cdot b^{j'} = 1$. Thus, $s_1 = S \cup \{b^i, b^{j'}\}$ and $s_2 = S \cup \{b^j, b^{i'}\}$ are feasible solutions whose incidence vectors, λ_1 and λ_2, lie in F_a. Hence, $\alpha^T \lambda_1 = \alpha^T \lambda_2$, implying that $\alpha_i + \alpha_{j'} = \alpha_j + \alpha_{i'}$. However, according to Case (I), $\alpha_{j'} = \alpha_{i'}$ because $b^{j'} \cdot b^{i'} = 2$. Therefore, we finally have that $\alpha_i = \alpha_j$.

Now let b^j be a column in B_t with $t \neq 3$. Define the solution $s_1 = \{b^j\} \cup S$, where S is a set composed of $\lfloor \frac{n-t}{3} \rfloor$ orthogonal columns in B_3. Also, let s_2 be the solution obtained from s_1 by replacing b^j by a set of $\lfloor \frac{t}{3} \rfloor$ orthogonal columns in B_3, all of them orthogonal to b^j. The incidence vectors λ_1 and λ_2 associated with s_1 and s_2 lie in F_a, and thus $\alpha^T \lambda_1 = \alpha^T \lambda_2$. This implies that $\alpha_j = \gamma \lfloor \frac{t}{3} \rfloor$. Hence, we have shown that $\alpha = \gamma a$.

Part (f): Consider the partitioning of R^1 into non-empty sets $C_1, C_2, \ldots, C_8$, where:

- $C_1 = \{b^j : \sum_{k=1}^{2} b^j_k = 2, \sum_{k=3}^{n} b^j_k = 0\}$
- $C_2 = \{b^j : \sum_{k=1}^{2} b^j_k = 2, b^j_3 = 0, \sum_{k=4}^{n} b^j_k = 1\}$
- $C_3 = \{b^j : \sum_{k=1}^{2} b^j_k = 1, b^j_3 = 1, \sum_{k=4}^{n} b^j_k = 0\}$
- $C_4 = \{b^j : \sum_{k=1}^{2} b^j_k = 1, b^j_3 = 1, 1 \leq \sum_{k=4}^{n} b^j_k < n-3\}$
- $C_5 = \{b^j : \sum_{k=1}^{2} b^j_k = 1, b^j_3 = 0, \sum_{k=4}^{n} b^j_k = 1\}$

- $C_6 = \{b^j : \sum_{k=1}^{2} b_k^j = 1, b_3^j = 0, \sum_{k=4}^{n} b_k^j > 1\}$
- $C_7 = \{b^j : \sum_{k=1}^{2} b_k^j = 0, b_3^j = 1, \sum_{k=4}^{n} b_k^j = n-3\}$
- $C_8 = \{b^j : \sum_{k=1}^{2} b_k^j = 0, b_3^j = 1, \sum_{k=4}^{n} b_k^j = n-4\}$

For some pairs of sets (C_p, C_q), we will show that $\alpha_j = \gamma$ for any column $b^j \in C_p \cup C_q$. Then, by transitivity, we will conclude that $\alpha_j = \gamma$ for any column $b^j \in R^1$.

(I) Let $b^i \in C_7$ and $b^j \in C_8$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, \bar{b}^i\}$ and $s_2 = \{b^j, \bar{b}^i\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i = \alpha_j$.

(II) Let $b^i \in C_3$ and $b^j \in C_8$. Define $b^k \in R^1$ such that $b_p^k = 1 - b_p^i$, for $p \in \{1,2\}$, and $b_p^k = 1 - b_p^j$, for $p \in \{3,\ldots,n\}$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, b^k\}$ and $s_2 = \{b^j, b^k\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i = \alpha_j$.

(III) Let $b^i \in C_1$ and $b^j \in C_2$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, \bar{b}^j\}$ and $s_2 = \{b^j, \bar{b}^j\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i = \alpha_j$.

(IV) Let $b^i \in C_1$ and $b^j \in C_5$. Define $b^{j'} \in R^1$ such that $b_1^{j'} = b_2^{j'} = 0$ and $b_p^{j'} = 1 - b_p^j$, for $p \in \{3,\ldots,n\}$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, \bar{b}^i\}$ and $s_2 = \{b^j, b^{j'}\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i + \alpha_{(2^n-1-i)} = \alpha_j + \alpha_{j'}$. Since $\bar{b}^i \in C_7$ and $b^{j'} \in C_8$, $\alpha_{(2^n-1-i)} = \alpha_{j'}$, hence $\alpha_i = \alpha_j$.

(V) Let $b^i \in C_5$ and $b^j \in C_6$. Define $b^{i'} \in R^1$ such that $b_p^{i'} = 1 - b_p^i$, for $p \in \{1,2,3\}$, and $\sum_{p=4}^{n} b_p^{i'} = 0$. Also, define $b^{j'} \in R^1$ such that $b_p^{j'} = 1 - b_p^j$, for $p \in \{1,2,3\}$, and $\sum_{p=4}^{n} b_p^{j'} = 0$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, b^{i'}\}$ and $s_2 = \{b^j, b^{j'}\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i + \alpha_{i'} = \alpha_j + \alpha_{j'}$. Since $b^{i'}, b^{j'} \in C_3$, $\alpha_{i'} = \alpha_{j'}$, hence $\alpha_i = \alpha_j$.

(VI) Let $b^i \in C_3$ and $b^j \in C_4$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, \bar{b}^i\}$ and $s_2 = \{b^j, \bar{b}^j\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i + \alpha_{(2^n-1-i)} = \alpha_j + \alpha_{(2^n-1-j)}$. Since $\bar{b}^i \in C_6$ and $\bar{b}^j \in C_5 \cup C_6$, $\alpha_{(2^n-1-i)} = \alpha_{(2^n-1-j)}$, hence $\alpha_i = \alpha_j$.

(VII) Let $b^i \in C_8$ and $b^j \in C_5$. Define $b^{i'} \in R^1$ such that $\sum_{p=1}^{2} b_p^{i'} = 1$ and $b_p^{i'} = 1 - b_p^i$, for $p \in \{3, \ldots, n\}$. Also, define $b^{j'} \in R^1$ such that $b_p^{j'} = 1 - b_p^j$, for $p \in \{1, \ldots, n\} \setminus \{3\}$, and $b_3^{j'} = 0$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, b^{i'}\}$ and $s_2 = \{b^j, b^{j'}\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i + \alpha_{i'} = \alpha_j + \alpha_{j'}$. Since $b^{i'} \in C_5$ and $b^{j'} \in C_5 \cup C_6$, $\alpha_{i'} = \alpha_{j'}$, hence $\alpha_i = \alpha_j$.

The relationships between columns in R^1 proved in (I)-(VII) are shown in the graph depicted in Figure 2.2. In this graph, each node represents a set C_p, $p \in \{1, \ldots, 8\}$, and an edge indicates that columns in the corresponding sets have the same coefficient in the vector α. Since this graph is connected and sets C_p, $p \in \{1, \ldots, 8\}$, are non-empty, there exists $\gamma \in \mathbb{R}$ such that $\alpha_j = \gamma$, for all $b^j \in R^1$.

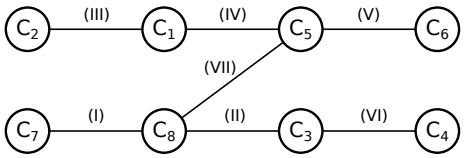


Figure 2.2: Proof of Part (f) of Theorem 2.2: a graph representing (I)-(VII).

Now let b^j be a column in R^2. Since the incidence vector of the solution $\{b_j\}$ lies in F_a, we have that $\alpha_j = \alpha_0$. However, solution $\{b^i, \bar{b}^i\}$ also lies in F_a, where b^i is any column in R^1. Therefore, $\alpha_j = \alpha_0 = 2\gamma$.

Finally, let b^j be a column in R^0. Notice that the incidence vectors of the solutions $s_1 = \{b^j, \bar{b}^j\}$ and $s_2 = \{\bar{b}^j\}$, λ_1 and λ_2, lie in F_a. Therefore, $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_j = 0$. Putting all together, we have shown that $\alpha = \gamma a$.

Part (g): Consider the partitioning of R^1 into non-empty sets $C_1, C_2, \ldots, C_8$, where:

- $C_1 = \{b^j : \sum_{k=1}^{2} b_k^j = 2, \sum_{k=3}^{n} b_k^j = 0\}$
- $C_2 = \{b^j : \sum_{k=1}^{2} b_k^j = 2, \sum_{k=3}^{4} b_k^j = 0, \sum_{k=5}^{n} b_k^j = 1\}$
- $C_3 = \{b^j : \sum_{k=1}^{2} b_k^j = 1, \sum_{k=3}^{4} b_k^j = 0, \sum_{k=5}^{n} b_k^j > 0\}$
- $C_4 = \{b^j : \sum_{k=1}^{2} b_k^j = 1, \sum_{k=3}^{4} b_k^j = 1, \sum_{k=5}^{n} b_k^j = 0\}$
- $C_5 = \{b^j : \sum_{k=1}^{2} b_k^j = 1, \sum_{k=3}^{4} b_k^j = 1, \sum_{k=5}^{n} b_k^j > 0\}$

- $C_6 = \{b^j : \sum_{k=1}^{2} b_k^j = 1, \sum_{k=3}^{4} b_k^j = 2, \sum_{k=5}^{n} b_k^j < n-4\}$
- $C_7 = \{b^j : \sum_{k=1}^{2} b_k^j = 0, \sum_{k=3}^{4} b_k^j = 2, \sum_{k=5}^{n} b_k^j = n-5\}$
- $C_8 = \{b^j : \sum_{k=1}^{2} b_k^j = 0, \sum_{k=3}^{4} b_k^j = 2, \sum_{k=5}^{n} b_k^j = n-4\}$

For some pairs of sets (C_p, C_q), we will show that $\alpha_j = \gamma$ for any column $b^j \in C_p \cup C_q$. Then, by transitivity, we will conclude that $\alpha_j = \gamma$ for any column $b^j \in R^1$.

(I) Let $b^i \in C_1$ and $b^j \in C_2$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, \bar{b}^j\}$ and $s_2 = \{b^j, \bar{b}^j\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i = \alpha_j$.

(II) Let $b^i \in C_7$ and $b^j \in C_8$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, \bar{b}^j\}$ and $s_2 = \{b^j, \bar{b}^j\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i = \alpha_j$.

(III) Let $b^i \in C_4$ and $b^j \in C_7$. Define $b^k \in R^1$ such that $b_p^k = 1 - b_p^i$, for $p \in \{1,2\}$, and $b_p^k = 1 - b_p^j$, for $p \in \{3,\ldots,n\}$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, b^k\}$ and $s_2 = \{b^j, b^k\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i = \alpha_j$.

(IV) Let $b^i \in C_3$ and $b^j \in C_4$. Define $b^{i'} \in R^1$ such that $b_p^{i'} = 1 - b_p^i$, for $p \in \{1,2\}$, $\sum_{p=3}^{4} b_p^{i'} = 1$ and $\sum_{p=5}^{n} b_p^{i'} = 0$. Also, define $b^{j'} \in R^1$ such that $b_p^{j'} = 1 - b_p^j$, for $p \in \{1,2,3,4\}$, and $\sum_{p=5}^{n} b_p^{j'} = 0$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, b^{i'}\}$ and $s_2 = \{b^j, b^{j'}\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i + \alpha_{i'} = \alpha_{j'} + \alpha_j$. Since $b^{i'}, b^{j'} \in C^4$, $\alpha_{i'} = \alpha_{j'}$, hence $\alpha_i = \alpha_j$.

(V) Let $b^i \in C_3$ and $b^j \in C_5$. Define $b^{i'} \in R^1$ such that $b_p^{i'} = 1 - b_p^i$, for $p \in \{1,2\}$, $\sum_{p=3}^{4} b_p^{i'} = 1$ and $\sum_{p=5}^{n} b_p^{i'} = 0$. Also, define $b^{j'} \in R^1$ such that $b_p^{j'} = 1 - b_p^j$, for $p \in \{1,2,3,4\}$, and $\sum_{p=5}^{n} b_p^{j'} = 0$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, b^{i'}\}$ and $s_2 = \{b^j, b^{j'}\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i + \alpha_{i'} = \alpha_{j'} + \alpha_j$. Since $b^{i'}, b^{j'} \in C^4$, $\alpha_{i'} = \alpha_{j'}$, hence $\alpha_i = \alpha_j$.

(VI) Let $b^i \in C_2$ and $b^j \in C_5$. Define $b^{j'} \in R^1$ such that $b_p^{j'} = 1 - b_p^j$, for $p \in \{1,2,3,4\}$, and $\sum_{p=5}^{n} b_p^{j'} = 0$. Therefore, the incidence vectors λ_1 and λ_2 associated with

solutions $s_1 = \{b^i, \bar{b}^i\}$ and $s_2 = \{b^j, b^{j'}\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i + \alpha_{(2^n-1-i)} = \alpha_{j'} + \alpha_j$. Since $\bar{b}^i \in C^7$ and $b^{j'} \in C^4$, $\alpha_{(2^n-1-i)} = \alpha_{j'}$, hence $\alpha_i = \alpha_j$.

(VII) Let $b^i \in C_5$ and $b^j \in C_6$. Define $b^{i'} \in R^1$ such that $b^{i'}_p = 1 - b^i_p$, for $p \in \{1, 2, 3, 4\}$, $\sum_{p=5}^{n} b^{i'}_p = 0$. Therefore, the incidence vectors λ_1 and λ_2 associated with solutions $s_1 = \{b^i, b^{i'}\}$ and $s_2 = \{b^j, \bar{b}^j\}$ lie in F_a, and thus $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_i + \alpha_{i'} = \alpha_{(2^n-1-j)} + \alpha_j$. Since $b^{i'} \in C^4$ and $\bar{b}^j \in C^3$, $\alpha_{i'} = \alpha_{(2^n-1-j)}$, hence $\alpha_i = \alpha_j$.

The relationships between columns in R^1 proved in (I)-(VII) are shown in the graph depicted in Figure 2.3. In this graph, each node represents a set C_p, $p \in \{1, \ldots, 8\}$, and an edge indicates that columns in the corresponding sets have the same coefficient in the vector α. Since this graph is connected and sets C_p, $p \in \{1, \ldots, 8\}$, are non-empty, there exists $\gamma \in \mathbb{R}$ such that $\alpha_j = \gamma$, for all $b^j \in R^1$.

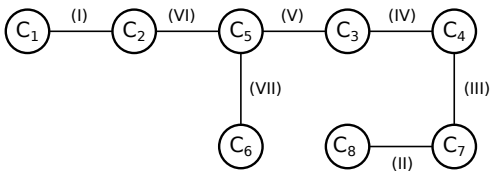


Figure 2.3: Proof of Part (g) of Theorem 2.2: a graph representing (I)-(VII).

Now let b^j be a column in R^2. Since the incidence vector of the solution $\{b_j\}$ lies in F_a, we have that $\alpha_j = \alpha_0$. However, solution $\{b^i, \bar{b}^i\}$ also lies in F_a, where b^i is any column in R^1. Therefore, $\alpha_j = \alpha_0 = 2\gamma$.

Finally, let b^j be a column in R^0. Notice that the incidence vectors of the solutions $s_1 = \{b^j, \bar{b}^j\}$ and $s_2 = \{\bar{b}^j\}$, λ_1 and λ_2, lie in F_a. Therefore, $\alpha^T\lambda_1 = \alpha^T\lambda_2$, which implies $\alpha_j = 0$. Putting all together, we have shown that $\alpha = \gamma a$. □

2.5 Conclusions

The close relation between the set packing and the set partitioning problems is very well-known and is ultimately derived from the fact that any set packing can be transformed into a set partitioning by just adding slack variables. However, the precise relation between the set packing and set partitioning polytopes for general matrices A is difficult to be determined, since even computing the dimension of $SPP_{=}(A)$ is NP-Hard. Theorem

2.1 shows that if A has all the $2^n - 1$ non-zero columns, there is a perfect one to one correspondence between the non-trivial facets of those polytopes. So, the study of $CSPP_{=}(n)$ can be reduced to the study of the "simpler" polytope $CSPP_{\leq}(n)$. This result is useful for proving that a given inequality induces a facet of $CSPP_{=}(n)$, since it is often easier to prove such a result for $CSPP_{\leq}(n)$. For example, before we realized this one to one correspondence, we failed to find a simple proof that, for n odd, the rank 1 cut induced by multipliers $\left(\frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2}\right)$ is facet-inducing for $CSPP_{=}(n)$. An interesting question left open by this work is whether Theorem 2.1 (or a modification of it) holds for a more general case — a promising case is when the matrix A contains all singleton columns. Another research avenue is to investigate the practical behavior of the new rank 1 facets found in this work.

Chapter 3

A Branch-and-Price Algorithm for the Minimum Latency Problem

This chapter deals with the Minimum Latency Problem (MLP), a variant of the well-known Traveling Salesman Problem in which the objective is to minimize the sum of waiting times of customers. This problem arises in many applications where customer satisfaction is more important than the total time spent by the server. This chapter presents a novel branch-and-price algorithm for MLP that strongly relies on new features for the *ng*-path relaxation, namely: (1) a new labeling algorithm with an enhanced dominance rule named multiple partial label dominance; (2) a generalized definition of *ng*-sets in terms of arcs, instead of nodes; and (3) a strategy for decreasing *ng*-set sizes when those sets are being dynamically chosen. Also, other elements of efficient exact algorithms for vehicle routing problems are incorporated into our method, such as reduced cost fixing, dual stabilization, route enumeration and strong branching. Computational experiments over TSPLIB instances are reported, showing that several instances not solved by the current state-of-the-art method can now be solved.

3.1 Introduction

In MLP, we are given a complete directed graph $G = (V, A)$ and a time t_{ij} for each arc $(i, j) \in A$. Set V is composed of $n + 1$ nodes: node 0, representing a depot, and nodes $1, \ldots, n$, representing n customers. The task is to find a Hamiltonian circuit ($i_0 = 0, i_1, \ldots, i_n, i_{n+1} = 0$), a.k.a. tour, in G that minimizes $\sum_{p=1}^{n+1} l(i_p)$, where the latency $l(i_p)$ is defined as the accumulated travel time from the depot to i_p. The MLP is related to the Time Dependent Traveling Salesman Problem (TDTSP), a generalization of the

Traveling Salesman Problem (TSP) in which the cost for traversing an arc depends on its position in the tour. More precisely, MLP can be viewed as the particular case of the TDTSP where the cost of an arc (i, j) in position $p, 0 \leq p \leq n$, is given by $(n - p + 1)t_{ij}$. A sample MLP instance is illustrated in Figure 3.1.

Figure 3.1: (a) Sample instance I with 4 customers. The graph is undirected because the travel times in I are symmetric. (b) An MLP solution for I with cost 33. The latencies are $l(1) = 1$, $l(2) = 5$, $l(3) = 6$, $l(4) = 7$ and $l(0) = 14$.

The MLP is also known in the literature as Delivery Man Problem (Roberti and Mingozzi, 2014), Traveling Repairman Problem (Afrati, Foto et al., 1986), Traveling Deliveryman Problem (Tsitsiklis, 1992) and Traveling Salesman Problem with Cumulative Costs (Bianco et al., 1993). Although MLP seems to be a simple variant of TSP, some important characteristics are very different in those problems. First, two different viewpoints of a distribution system are considered: TSP is server oriented, since one wants to minimize the total travel time; on the other hand, MLP is customer oriented because the objective is equivalent to minimizing the average waiting time of customers (Silva et al., 2012; Sitters, 2002; Archer and Williamson, 2003). Customer satisfaction is the main objective in many applications, such as home delivery services (Méndez-Díaz et al., 2008), and has attracted the attention of researchers, as reflected by the considerable number of MLP variants studied in the very last years (see, for instance, (Lysgaard and Wøhlk, 2014; Rivera et al., 2016; Nucamendi-Guillén et al., 2016; Sze et al., 2017)). Second, in contrast to what happens in TSP, simple local changes may affect globally an MLP solution because the latency of subsequent customers may change (Silva et al., 2012; Sitters, 2002). This can make it more difficult to solve MLP both exactly and heuristically. For example, current state-of-the-art exact methods for MLP are not capable of solving consistently instances with 150 customers, whereas TSP instances with thousands of customers are solved routinely (Abeledo et al., 2013).

Many complexity results for MLP have been obtained. The problem is NP-Hard for general metric spaces (Sahni and Gonzalez, 1976), and remains NP-Hard even if the times correspond to Euclidean distances (Afrati, Foto et al., 1986) or if they are obtained

from an underlying graph that is a tree (Sitters, 2002). On the other hand, the problem is polynomial if the underlying graph is a path (Afrati, Foto et al., 1986; García et al., 2002), a tree with equal weights or a tree with diameter at most 3 (Blum et al., 1994). The MLP with deadlines, i.e., with upper bounds on $l(i_p)$, is NP-Hard even for paths (Afrati, Foto et al., 1986). In terms of approximation, hardness results show that one should not expect to attain arbitrarily good approximation factors for MLP (Blum et al., 1994). However, 3.59 and 3.03 approximations are known for general metric spaces and general trees, respectively (Chaudhuri et al., 2003; Archer and Blasiak, 2010). Moreover, a constant factor approximation is not likely to exist if times do not satisfy the triangle inequality, just as for TSP (Blum et al., 1994).

The first integer programming formulations were given in (Picard and Queyranne, 1978), where the authors stated TDTSP as a machine scheduling problem and solved instances with up to 20 jobs by means of a branch-and-bound method over lagrangian bounds. A new formulation with n constraints was presented in (Fox et al., 1980), but the authors did not report any computational results. Lucena, (1990) and Bianco et al., (1993) followed the same approach as Picard and Queyranne, (1978) and employed langragian bounds in experiments over MLP instances with up to 30 and 60 vertices, respectively. The latter authors also developed a dynamic programming method capable of attesting that the bounds obtained for 60-vertex instances were within 3% from optimality. Then, a series of enumerative strategies based on new formulations was introduced in (Fischetti et al., 1993; Van Eijl, 1995; Méndez-Díaz et al., 2008; Bigras et al., 2008; Godinho et al., 2014), as well as cutting planes (Van Eijl, 1995; Méndez-Díaz et al., 2008; Bigras et al., 2008) and polyhedral studies (Méndez-Díaz et al., 2008). Instances with 60 vertices could already be solved by the algorithm of Fischetti et al., (1993). More recently, Abeledo et al., (2013) managed to solve almost all TSPLIB instances with up to 107 vertices using a branch-cut-and-price algorithm. The authors departed from a formulation by Picard and Queyranne, (1978) and proposed new inequalities, that are proved to be facet-inducing. Roberti and Mingozzi, (2014) implemented dual ascent and column generation techniques to compute a sequence of lower bounds associated with set partitioning formulations where a column represents an *ng*-path, which is a path relaxation introduced by Baldacci et al., (2011). An *ng*-path may contain cycles, but just those allowed by the so-called *ng*-sets. These sets are iteratively augmented so that less cycles are allowed and improved bounds are obtained. The final lower bound is used in a dynamic programming recursion to compute the optimal solution. This method could solve some larger TSPLIB instances, with up to 150 vertices, and currently holds the status of state-of-the-art exact method

for MLP. Finally, heuristic algorithms for MLP can be found in (Ngueveu et al., 2010; Salehipour et al., 2011; Silva et al., 2012; Mladenović et al., 2013).

This chapter presents a novel branch-and-price algorithm for MLP that strongly relies on *ng*-paths. Following the directions of Roberti and Mingozzi, (2014), our method works over a set partitioning formulation where columns represent *ng*-paths and the column generation bounds computed on each node of the tree are derived from dynamically defined *ng*-sets. However, we introduce the following improvements on the use of *ng*-paths.

- **Multiple Partial Label Dominance:** In the labeling algorithms used for pricing *ng*-paths, a partial path P is represented as a label $L(P)$. A key concept in this kind of algorithm is dominance. A label $L(P_1)$ dominates a label $L(P_2)$ if every completion P' of P_2 is also a feasible completion of P_1, and the cost of $P_1 + P'$ is not larger than the cost of $P_2 + P'$. In this case, $L(P_2)$ can be safely eliminated. In this chapter, we propose a stronger dominance rule by which some extensions for $L(P_2)$ can be avoided, even though this label cannot be completely disregarded according to the classical dominance rule. We briefly discuss two alternative implementations of this new dominance rule, where the best one typically speeds up the labeling algorithm by factors between 4 and 8.
- **Arc-Based *ng*-Path Relaxation:** *ng*-sets as originally defined by Baldacci et al., (2011) are a vertex-based memory mechanism. In this chapter, we provide a generalized definition of them in terms of arcs. We show that this new definition is particularly useful in the context of dynamically defined *ng*-sets, allowing strong bounds to be obtained in more controlled pricing times.
- **Fully Dynamic *ng*-Path Relaxation:** We improve the dynamic *ng*-path relaxation of Roberti and Mingozzi, (2014) by introducing a procedure for decreasing the *ng*-sets, without changing the current bounds. Such reductions are beneficial for the pricing time and also help to refine the choice of *ng*-sets.

Also, other well-known elements of efficient exact algorithms for many other variants of the vehicle routing problem (VRP) are incorporated into our method, namely reduced cost fixing, dual stabilization, route enumeration and strong branching. Computational experiments over MLP instances derived from TSPLIB were conducted to attest the effectiveness of the new branch-and-price algorithm. The results show that better bounds can be obtained in less computational time when compared to the state-of-the-art algorithm, specially because of the new features for the *ng*-path relaxation. In particu-

lar, the branch-and-price solved all the 9 instances with up to 150 vertices not solved in Roberti and Mingozzi, (2014). It could also solve 4 additional instances, with more than 150 vertices, never considered before by exact methods.

The remainder of this chapter is organized as follows. Section 3.2 discusses the *ng*-path relaxation and labeling algorithms. Section 3.3 introduces the new features for the *ng*-path relaxation. The proposed branch-and-price algorithm is described in Section 3.4, where we also give implementation details. Computational experiments are presented in Section 3.5. Finally, concluding remarks are drawn in the last section.

3.2 Route Relaxations and Labeling Algorithms

This section reviews the route relaxations and labeling algorithms that are related to current state-of-the-art exact algorithms for VRPs, such as Capacitated VRP (CVRP), VRP with time windows (VRPTW), and the MLP itself. Such algorithms are based on a combination of column and cut generation over the following set-partitioning formulation.

$$\min \sum_{R \in \Omega} c_R \lambda_R \tag{3.1}$$

$$\text{s.t.} \quad \sum_{R \in \Omega} a_R^i \lambda_R = 1, \qquad \forall i \in \mathcal{C}, \tag{3.2}$$

$$\lambda_R \in \{0,1\}, \qquad \forall R \in \Omega, \tag{3.3}$$

where $\mathcal{C}$, Ω, c_R and a_R^i denote, respectively, the set of customers, the set of feasible routes, the cost of route R, and the number of times route R visits customer i.

As the number of variables in Formulation (3.1)-(3.3) is exponential in $|\mathcal{C}|$, column generation is typically applied to solve its linear relaxation. The pricing subproblem depends on the considered variant, but it can often be modeled as the Elementary Resource Constrained Shortest Path Problem (ERCSPP). In ERCSPP, we are given a directed graph $G' = (V', A')$ with vertex set V' and arc set A'; source and sink nodes $s \in V'$ and $t \in V'$, respectively; and a set of resources $\mathcal{W}$. Moreover, for each $i \in V'$ and $r \in \mathcal{W}$, let $l_i^r \in \mathbb{R}$ and $u_i^r \in \mathbb{R}$ be, respectively, the minimum and the maximum consumption of resource r in any partial path from s to i. Each partial path $P = (i_0 = s, i_1, \ldots, i_p)$ has an associated vector of resource consumption $w(P) \in \mathbb{R}^{|\mathcal{W}|}$, which is computed with the help of resource extension functions (REFs) $f_{ij} : \mathbb{R}^{|\mathcal{W}|} \to \mathbb{R}^{|\mathcal{W}|}$, one for each $(i,j) \in A'$.

More precisely,

$$w(P) = f_{i_{p-1}i_p}(w(P' = (i_0, \ldots, i_{p-1}))), \tag{3.4}$$

if $p \geq 1$, or $w(P)$ is any vector that satisfies the bounds of the source node, if $p = 0$. That is, the initial resource consumption is a decision variable. The task is to find the least-cost $s - t$ path containing no cycles and satisfying the resource constraints, where the cost $c(P)$ of a partial path $P = (i_0 = s, i_1, \ldots, i_p)$ is computed with the help of cost extension functions (CEFs) $c_{ij} : \mathbb{R}^{|\mathcal{W}|} \to \mathbb{R}$, one for each arc $(i, j) \in A'$, as follows:

$$c(P) = \begin{cases} c(P' = (i_0, \ldots, i_{p-1})) + c_{i_{p-1}i_p}(w(P' = (i_0, \ldots, i_{p-1}))), \text{if } p \geq 1 \\ 0, \text{otherwise.} \end{cases} \tag{3.5}$$

See (Irnich, 2008; Irnich and Desaulniers, 2005) for further details on this very generic definition of ERCSPP. From now on, for ease of presentation and because *the new ideas for the pricing subproblem described in this chapter are only concerned with the ng-path relaxation*, we will assume a definition of the problem in which a single discrete resource is present and the REF $f_{ij} : \mathbb{Z} \to \mathbb{Z}$ for an arc $(i, j) \in A'$ is the function $f_{ij}(w) = w + w_{ij}$, where $w_{ij} \in \mathbb{Z}$ denotes the consumption of this single resource by the arc. The lower and upper bounds on resource consumption when reaching a vertex $i \in V'$ will be denoted as $l_i \in \mathbb{Z}$ and $u_i \in \mathbb{Z}$, respectively. In addition, there exists a CEF $c_{ij} : \mathbb{Z} \to \mathbb{Z}$ for each arc $(i, j) \in A'$. Finally, we will also assume that $V' = \mathcal{C} \cup \{s, t\}$. Remark that this simplified definition still covers the MLP case.

ERCSPP is NP-Hard in the strong sense (Dror, 1994) and also a difficult problem to solve in practice. The hard constraint is the one that imposes elementarity, since the Resource Constrained Shortest Path Problem (RCSPP) can be solved in pseudo-polynomial time for a fixed number of resources. RCSPP is a relaxation of ERCSPP where a vertex can be visited more than once in an optimal path, as long as the resource constraints are satisfied. In view of this, several state-of-the-art algorithms employ some route relaxation as an alternative to elementary routes. The idea is to replace set Ω by some set also containing non-elementary routes so as to make the pricing subproblem easier. An ideal relaxation would provide the elementary route bound while keeping the pricing subproblem tractable. It is worth mentioning that such a relaxation is mandatory in problems where the optimal solution has exactly one route (e.g., MLP), otherwise the pricing subproblem is as hard as the original problem, rendering column generation meaningless.

The first route relaxation is just to allow any non-elementary route, as long as it

satisfies the resource constraints. It was already observed in Christofides et al., (1981) that it is possible to eliminate routes with 2-cycles (subpaths like $i \to j \to i$) without increasing the complexity. The bounds obtained with 2-cycle elimination may be good in some cases, but are likely to be poor in other cases, specially in routing problems with many customers per route (Martinelli et al., 2014). This motivated Irnich and Villeneuve, (2006) to propose an algorithm to forbid cycles of an arbitrary maximum size k. The pricing subproblem now is referred to as RCSPP with k-cycle elimination. Theoretically, the proposed algorithm can be used for pricing elementary routes, but only small values of k can be efficiently used in practice. This is because the complexity of the algorithm grows factorially with k. Nevertheless, k-cycle elimination for small values of k proved to be useful at that time, improving column generation based algorithms for several VRP variants. For example, the branch-cut-and-price for MLP in (Abeledo et al., 2013) uses $k = 5$.

Later, Baldacci et al., (2011) introduced a new kind of elementarity relaxation, the so-called *ng*-paths (a.k.a. *ng*-routes). Extensive experiments on several VRP variants show that *ng*-routes are almost always more efficient than routes without k-cycles, in the sense of providing better bounds in less computational time. In contrast to previous relaxations, cycles eliminated by *ng*-routes are not distinguished by size. Instead, a cycle $H = (i_0, i_1, \ldots, i_p = i_0)$ is forbidden if all customers $i_1, \ldots, i_{p-1}$ are able to "remember" customer i_0. Formally speaking, we define an *ng*-set $N_i \subseteq \mathcal{C}$ for each customer $i \in \mathcal{C}$. We assume that $i \in N_i$. Typically, N_i contains the customers that are likely to appear close to customer i in low-cost paths, e.g., nearest customers to take advantage of a locality principle that is often present in VRPs. In case $j \in N_i$, we say that i remembers j or equivalently that j is remembered by i. Then, each path $P = (\ i_0, i_1, \ldots, i_p)$ has an associated set of forbidden extensions $\Pi(P)$ that is computed based on the *ng*-sets:

$$\Pi(P) = \left\{ i_m \in \{i_0, \ldots, i_{p-1}\} : i_m \in \bigcap_{q=m+1}^{p} N_{i_q} \right\} \cup \{i_p\} \tag{3.6}$$

In words, a customer $i_m \neq i_p$ belongs to $\Pi(P)$ if it is remembered by all customers i_q with $m < q \leq p$. Therefore, path P is *ng*-feasible if $i_m \notin \Pi(P_{m-1} = (i_0, i_1, \ldots, i_{m-1})), 1 \leq m \leq p$. Alternatively, one can define the *ng*-path relaxation in terms of the *ng*-memory $M_i \subseteq \mathcal{C}$ of a customer i, which is the set of customers that remember i, i.e., $M_i = \{j \in \mathcal{C} : i \in N_j\}$. In this case, the set of forbidden extensions for path P is computed as:

$$\Pi(P) = \left\{ i_m \in \{i_0, \ldots, i_{p-1}\} : i_q \in M_{i_m}, q = m+1, \ldots, p \right\} \cup \{i_p\} \tag{3.7}$$

and a path P is *ng*-feasible iff between two visits to a customer i, some customer $j \notin M_i$ is visited. In this chapter, both concepts (*ng*-sets and *ng*-memories) are used. Clearly, the greater the *ng*-memories, the closer to elementary are the *ng*-paths.

The RCSPP with *ng*-routes is commonly solved by means of a labeling algorithm. In this kind of algorithm, a label $L(P) = (c(P), w(P), v(P), \Pi(P))$ represents a partial path P in G' ending at vertex $v(P)$ with cost $c(P)$, resource consumption $w(P)$, and set $\Pi(P)$ of forbidden extensions. We define $N_s = N_t = \emptyset$ so as to guarantee that $\Pi(P)$ is well-defined for paths in G' (recall that we have assumed $V' = \mathcal{C} \cup \{s, t\}$). Initial labels representing paths defined only by the source vertex s are present in the beginning, one for each possible resource consumption in $[l_s, u_s]$, and new labels are generated by extending existing ones along all possible arcs. Labels are processed in increasing order of resource consumption and dominance rules are applied to eliminate labels not leading to optimal paths. The general scheme of labeling algorithms with *ng*-paths is shown in Algorithm 3, where L_j and U_j denote, respectively, the set of processed and unprocessed labels that correspond to paths ending at vertex j.

Algorithm 3 General Labeling Algorithm with *ng*-paths

1: Initialize L_j and U_j as empty sets, for all $j \in V'$
2: Add the initial labels to U_s
3: **while** $\bigcup_{j \in V'} U_j \neq \emptyset$ **do**
4: Choose a label $L(P) \in \bigcup_{j \in V'} U_j$ with minimum resource consumption and let $i = v(P)$
5: **for each** $(i, v) \in A'$ such that $P + v$ is feasible **do**
6: Define label $L(P + v) = (c(P) + c_{iv}(w(P)), w(P) + w_{iv}, v, (\Pi(P) \cap N_v) \cup \{v\})$
7: **if** $L(P + v)$ is dominated by any label in $L_v \cup U_v$ **then**
8: **continue**
9: **else**
10: Remove labels in U_v dominated by $L(P + v)$
11: $U_v \leftarrow U_v \cup \{L(P + v)\}$
12: $U_i \leftarrow U_i \setminus \{L(P)\}$
13: $L_i \leftarrow L_i \cup \{L(P)\}$
14: **return** best label in L_t

The feasibility test in Line 5 of Algorithm 3 takes into account the resource constraints and the extensions forbidden by *ng*-sets. More precisely, the new path $P + v$ is feasible if $l_v \leq w(P) + w_{iv} \leq u_v$ and $v \notin \Pi(P)$. Regarding the dominance checks performed in lines 7 and 10, the following conditions are sufficient to verify that a label $L(P')$ dominates a label $L(P)$.

(I) $v(P') = v(P)$

(II) $c(P') \leq c(P)$

(III) $w(P') = w(P)$

(IV) $\Pi(P') \subseteq \Pi(P)$

3.3 New Features for the *ng*-Path Relaxation

This section presents new contributions on the use of the *ng*-path relaxation. Although in this chapter they are used for solving MLP, we remark that they can be used in several other problems.

3.3.1 Multiple Partial Label Dominance

We will now introduce a stronger dominance rule called *multiple partial label dominance* (MPLD). Suppose that condition (IV) of the classical dominance rule discussed in Section 3.2 is the only one not satisfied by $L(P)$ and $L(P')$. Therefore, label $L(P)$ is not dominated by label $L(P')$ because there exists a vertex i such that $P+i$ is *ng*-feasible, whereas $P'+i$ is not. While completely disregarding label $L(P)$ because of label $L(P')$ is not correct, some extensions for the former may be unnecessary. For example, in Figure 3.2, one can see that $L(P')$ does not dominate $L(P)$ only because $P+2$ is *ng*-feasible, whereas $P'+2$ is not. However, observe that $\Pi(P'+3) = \{4,3\} \subseteq \Pi(P+3) = \{4,3\}$ since $3 \notin M_1 \cup M_2$. Thus, label $L(P'+3)$ dominates $L(P+3)$, which makes the extension of $L(P)$ along the arc $(1,3)$ unnecessary as long as $L(P')$ is extended along this arc. In this case, we say that label $L(P)$ is *partially dominated* by label $L(P')$.

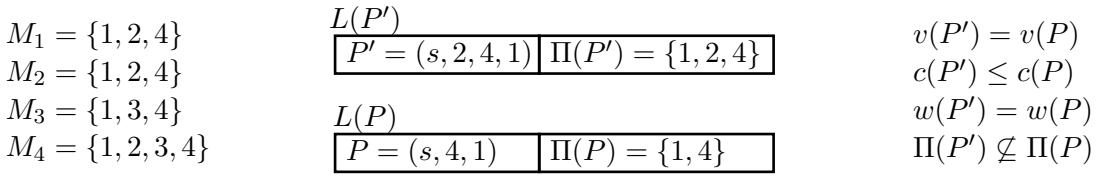


Figure 3.2: Introducing MPLD. Label $L(P')$ does not dominate label $L(P)$, but $L(P'+3)$ dominates $L(P+3)$.

The general result is the following.

Proposition 3.1. *Let $L(P)$ and $L(P')$ be two labels such that*

- $v(P') = v(P) = i$

- $c(P') \leq c(P)$
- $w(P') = w(P)$
- $\Pi(P') \not\subseteq \Pi(P)$

Then any extension to a vertex $j \in V' \setminus \bigcup_{v \in \Pi'} M_v$ *can be safely disregarded for* $L(P)$*, where* $\Pi' = \Pi(P') \setminus \Pi(P)$*.*

Proof. Let $j \in V'$ be a vertex such that $(i, j) \in A'$ and $j \notin \bigcup_{v \in \Pi'} M_v$. Since we have assumed that $j \in M_j$, we have that $j \notin \Pi'$. Hence, if $P + j$ is *ng*-feasible, so is $P' + j$ and clearly $v(P' + j) = v(P + j)$, $c(P' + j) \leq c(P + j)$ and $w(P' + j) = w(P + j)$. Furthermore, as $j \notin \bigcup_{v \in \Pi'} M_v$, any vertex $v \in \Pi'$ is forgotten by $P' + j$ and $P + j$, and thus $\Pi(P' + j) \subseteq \Pi(P + j)$. Therefore, label $L(P' + j)$ dominates label $L(P + j)$. □

In view of Proposition 3.1, a label $L(P)$ should also store a set $\xi(P)$ representing the extensions that can be avoided because of partial dominance, which we call *dominated extensions*. Let $\mathcal{L}(P)$ be the set of all labels $L(P')$ that together with label $L(P)$ satisfy conditions (I), (II) and (III). By applying the partial dominance from the multiple labels in $\mathcal{L}(P)$, the resulting set of dominated extensions for label $L(P)$ is:

$$\xi(P) = \bigcup_{L(P') \in \mathcal{L}(P)} \left\{ j \in V' : j \notin \bigcup_{\substack{v \in \Pi(P') \\ v \notin \Pi(P)}} M_v \right\} \tag{3.8}$$

A small example of MPLD is presented in Figure 3.3. In this example, we have six nodes (besides source and sink nodes) and labels $L(P_1)$, $L(P_2)$, $L(P_3)$ and $L(P_4)$ represent all partial paths ending at vertex 1 with a given resource consumption. As can be seen in the figure, we have: $\mathcal{L}(P_1) = \emptyset, \mathcal{L}(P_2) = \{L(P_1)\}, \mathcal{L}(P_3) = \{L(P_1), L(P_2)\}$ and $\mathcal{L}(P_4) = \{L(P_1), L(P_2), L(P_3)\}$. Label $L(P_1)$ does not dominate label $L(P_2)$ only because of condition (IV), but any extension of $L(P_2)$ to a vertex $j \notin M_2$ is unnecessary since $\Pi(P_1) \setminus \Pi(P_2) = \{2\}$, which implies $\Pi(P_1 + j) \subseteq \Pi(P_2 + j)$ if $j \notin M_2$. Thus, $\xi(P_2) = V \setminus M_2$. Set $\xi(P_3)$ is defined as $\xi(P_3) = (V \setminus (M_2 \cup M_5)) \cup (V \setminus M_5)$, where $(V \setminus (M_2 \cup M_5)$ is because of label $L(P_1)$ and $(V \setminus M_5)$ is because of label $L(P_2)$. Finally, $\xi(P_4) = (V \setminus M_5) \cup (V \setminus M_3)$, where $(V \setminus M_5)$ is because of label $L(P_1)$ or label $L(P_2)$ and $(V \setminus M_3)$ is because of label $L(P_3)$. All extensions for label $L(P_4)$ are either forbidden because of *ng*-sets or dominated because of labels $L(P_1)$, $L(P_2)$ and $L(P_3)$, therefore $L(P_4)$ can be removed. This illustrates a situation where MPLD results in complete dominance.

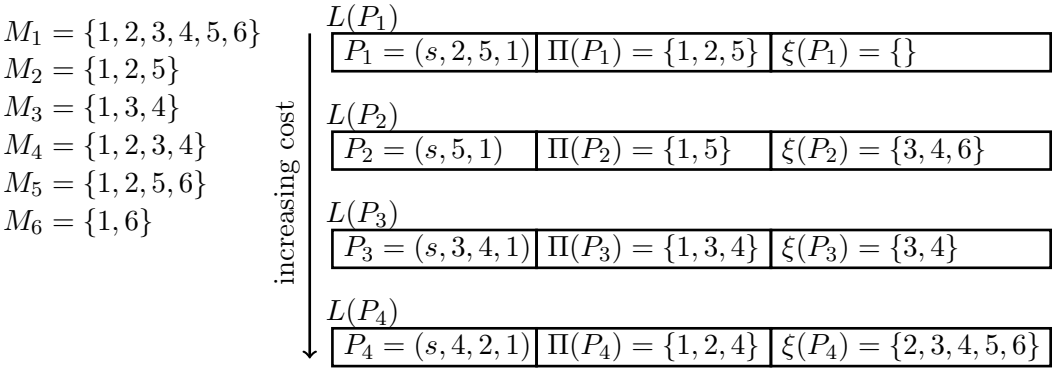


Figure 3.3: An example of MPLD. Left: *ng*-memories. Right: labels representing paths ending at vertex 1, all of them with the same resource consumption.

The intuition behind MPLD is to avoid extensions by anticipating that a label $L(P + j)$ would be dominated by a label $L(P' + j)$. However, a good implementation of MPLD is required, otherwise the additional cost of checking for partial dominance will not pay off, as will be discussed in Section 3.4.2.

3.3.2 Arc-Based *ng*-Path Relaxation

The main idea exploited by the *ng*-path relaxation is that, in many problems, cycles are often confined to small "neighborhoods" of the graph: once a new visit to a node i is performed by a partial path P, any node j that is not a neighbor of i is forgotten. The set of nodes remembered by P is computed as a function of *ng*-memories defined over the set of vertices V'. In this section, we show a generalized definition of the *ng*-path relaxation where *ng*-memories are defined in terms of arcs instead of nodes. The inspiration for this new definition comes from the arc-based limited memory technique developed in (Pecin et al., 2017c) in order to reduce the impact of the non-robust Rank-1 Chvátal-Gomory cuts on the labeling algorithm.

For each arc $(i, j) \in A'$, we define an *ng*-set $\overrightarrow{N}_{ij} \subseteq V'$, which is the set of vertices remembered by the arc. The arc-based *ng*-memory of a vertex $j \in V'$ is defined as the set of arcs that remember j, i.e., $\overrightarrow{M}_j = \{(i, j) \in A' : j \in \overrightarrow{N}_{ij}\}$. Let $P = (a_0, a_1, \ldots, a_p)$ be a partial path in G' composed of arcs $a_0 = (s = i_0, i_1), a_1 = (i_1, i_2), \ldots, a_p = (i_p, i_{p+1})$. Similar to Equations (3.6) and (3.7), we now have the following equivalent definitions of the set of forbidden extensions for path P:

$$\overrightarrow{\Pi}(P) = \left\{ i_m \in \{i_0, \ldots, i_p\} : i_m \in \bigcap_{q=m}^{p} \overrightarrow{N}_{a_q} \right\} \cup \{i_{p+1}\} \tag{3.9}$$

$$\overrightarrow{\Pi}(P) = \left\{ i_m \in \{i_0, \ldots, i_p\} : a_q \in \overrightarrow{M}_{i_m}, q = m, \ldots, p \right\} \cup \{i_{p+1}\} \tag{3.10}$$

Set $\overrightarrow{\Pi}(P)$ equals set $\Pi(P)$ if one defines $\overrightarrow{M}_v = \{(i,j) \in A' : i \in M_v \wedge j \in M_v\}$ for every vertex $v \in V'$. This generalized definition is particularly useful in the context of dynamically defined *ng*-memories. In this setting, one wants to augment current *ng*-memories in order to forbid a given cycle $H = (a_0 = (i_0, i_1), a_1 = (i_1, i_2), \ldots, a_p = (i_p, i_{p+1} = i_0))$. For doing so, vertices $i_1, \ldots, i_p$ should be added to the vertex-based *ng*-memory M_{i_0}. However, this forbids not only H, but any cycle $H' = (i_0, \ldots, i_0)$ passing through a subset of $\{i_0, i_1, \ldots, i_p\}$, which may represent a considerable impact on the labeling algorithm. On the other hand, the impact of adding $a_0, \ldots, a_{p-1}$ to $\overrightarrow{M}_{i_0}$ is much less considerable since only cycles $H' = (i_0, \ldots, i_q, i_0)$, with $q = 1, \ldots, p$, are forbidden — notice that it is not necessary to add (i_q, i_0) to $\overrightarrow{M}_{i_0}$ to forbid a cycle $H' = (i_0, \ldots, i_q, i_0)$ because $i_0 \in \overrightarrow{\Pi}((i_0, i_1, \ldots, i_q))$ if $a_0, \ldots, a_{q-1} \in \overrightarrow{M}_{i_0}$. We will show in our computational experiments that this reduced impact is crucial for solving hard MLP instances.

Hereafter, the term *ng*-memory refers to arc-based *ng*-memory and we will explicitly indicate if we refer to vertex-based *ng*-memory.

3.3.3 Fully Dynamic *ng*-Path Relaxation

Let us consider (arc-based) *ng*-memories $\overrightarrow{\mathcal{M}}$ and define $\Omega(\overrightarrow{\mathcal{M}})$ as the set of all feasible *ng*-routes w.r.t. $\overrightarrow{\mathcal{M}}$. Notice that $\Omega \subseteq \Omega(\overrightarrow{\mathcal{M}})$. As we have discussed before, the following relaxation of formulation (3.1)-(3.3), hereafter denoted $LP(\overrightarrow{\mathcal{M}})$, is the basis of several state-of-the-art column generation based algorithms for vehicle routing problems.

$$LB(\overrightarrow{\mathcal{M}}) = \min \sum_{R \in \Omega(\overrightarrow{\mathcal{M}})} c_R \lambda_R \tag{3.11}$$

$$\text{s.t.} \quad \sum_{R \in \Omega(\overrightarrow{\mathcal{M}})} a_R^i \lambda_R = 1, \qquad \forall i \in \mathcal{C}, \tag{3.12}$$

$$\lambda_R \geq 0, \qquad \forall R \in \Omega(\overrightarrow{\mathcal{M}}). \tag{3.13}$$

The quality of the bound $LB(\overrightarrow{\mathcal{M}})$ depends on the *ng*-memories $\overrightarrow{\mathcal{M}}$. Ideally, one should define $\overrightarrow{\mathcal{M}}$ so as to guarantee that $LB(\overrightarrow{\mathcal{M}})$ corresponds to the bound attained if $\Omega(\overrightarrow{\mathcal{M}})$ is replaced by Ω in the formulation, i.e., if only elementary routes are generated in the pricing subproblem. This may require large *ng*-memories, and thus advanced techniques are needed for solving the pricing subproblem in reasonable computational times.

For example, Martinelli et al., (2014) described an algorithm based on the decremental state-space relaxation (DSSR) technique suggested by Righini and Salani, (2008) where *ng*-memories are iteratively augmented while the best column generated in the pricing subproblem is not *ng*-feasible w.r.t. some large *ng*-memories.

Good bounds can also be obtained if *ng*-memories are very well chosen, but not necessarily large. In this regard, Roberti and Mingozzi, (2014) introduced the *dynamic ng-path relaxation*, which is roughly a sequence of non-decreasing lower bounds $LB(\mathcal{M}_1), LB(\mathcal{M}_2), \ldots, LB(\mathcal{M}_k)$ associated with dynamically defined vertex-based *ng*-memories $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$. Initial *ng*-memories $\mathcal{M}_1$ correspond to the Δ^1 nearest customers, and memories $\mathcal{M}_{k'+1}$ are computed by extending memories $\mathcal{M}_{k'}$, $k' \in \{1, \ldots, k-1\}$, in order to forbid the smallest cycle of the column with the largest primal value in a near-optimal solution of the linear relaxation of $LP(\mathcal{M}_{k'})$. A limit of Δ^2 is imposed for the size of each *ng*-memory, and the method stops when no cycle can be forbidden. The interested reader should consult (Roberti and Mingozzi, 2014) for further details.

In this section, we propose improvements to the dynamic *ng*-path relaxation, the main difference being the possibility to also reduce the *ng*-memories. This reduction is carried out if the pricing subproblems in the previous iteration of the method have been considered expensive. However, the role of this reduction is not only to make the pricing subproblem easier, but also to allow one a better choice of *ng*-memories. Finally, instead of forbidding cycles of a single column, we employ a potentially more aggressive algorithm for augmenting the *ng*-memories. The *fully dynamic ng-path relaxation* is outlined in Algorithm 4, and its main ingredients are described next.

Algorithm 4 Fully Dynamic *ng*-Path Relaxation

1: **procedure** fullyDynNgPath($\overrightarrow{\mathcal{M}}$, Δ^{max}, α_I, β_I, α_R, β_R)
2: $k \leftarrow 1$, $\overrightarrow{\mathcal{M}}_1 \leftarrow \overrightarrow{\mathcal{M}}$
3: **repeat**
4: Compute $(\bar{\lambda}_k, \bar{\pi}_k)$, an optimal primal-dual solution pair of $LP(\overrightarrow{\mathcal{M}}_k)$
5: **if** *ng*-memories reduction condition is satisfied **then**
6: $\overrightarrow{\mathcal{M}}_k \leftarrow$ reduceNGMemories($\overrightarrow{\mathcal{M}}_k$, $\bar{\pi}_k$, α_R, β_R)
7: Let $\mathcal{R}$ be the set of *ng*-routes associated with the primal solution $\bar{\lambda}_k$
8: $\overrightarrow{\mathcal{M}}_{k+1} \leftarrow$ augmentNgMemories($\mathcal{R}$, $\overrightarrow{\mathcal{M}}_k$, $\emptyset$, Δ^{max}, α_I, β_I, `aggressive`)
9: **until** stop condition is met

Algorithm 5 presents the procedure adopted to augment *ng*-memories. Up to two phases are executed for each *ng*-route $R \in \mathcal{R}$: first, the procedure attempts to forbid all cycles $H \in \mathcal{H}(R)$ with at most α vertices, where $\mathcal{H}(R)$ denotes the set of all cycles of

R (Phase 1); if no cycle could be forbidden in Phase 1, then all cycles $H \in \mathcal{H}(R)$ are considered, regardless of their size, and R is skipped after one cycle could be forbidden (Phase 2). We prioritize small cycles because they require less augmentations to be forbidden and many of them are likely to appear repeatedly in low-cost *ng*-routes. The *ng*-routes are sorted in a non-increasing order of primal value or non-decreasing order of reduced cost, depending on where the procedure is called from (Algorithm 4 or Algorithm 6, respectively). The procedure stops if β *ng*-routes have been explicitly forbidden. Notice that parameters α and β control the aggressiveness of the *ng*-memories augmentation. The other parameters of this procedure are:

- A set $\mathcal{A}$ of forbidden augmentations. This set is used just in the proposed *ng*-memories reduction algorithm that will be explained later in this section (observe that $\mathcal{A} = \emptyset$ when the procedure is called from Algorithm 4).
- A limit Δ^{max} to $\gamma(j)$, the number of *ng*-memories arcs $\delta^-(j)$ belong to. As usual, $\delta^-(j)$ denotes the set of arcs entering vertex j. This is needed because the number of non-dominated labels representing paths ending at a vertex j may be exponential in $\gamma(j)$. Thus, even though arc-based *ng*-memories are used, the complexity of the labeling algorithm is still somewhat vertex-dependent.
- The mode of *ng*-memories augmentation, which is either `moderate` or `aggressive`. Suppose one wants to augment *ng*-memories so that cycle $H = (a_0 = (i_0 = v, i_1), a_1 = (i_1, i_2), \ldots, a_p = (i_p, i_{p+1} = v))$ is forbidden. In `moderate` mode, this is attained by adding arcs $a_0, \ldots, a_{p-1}$ to $\overrightarrow{M}_v$. Notice that cycles $H = (v, i_1, \ldots, i_q, v)$, with $q = 1, \ldots, p$, are now also forbidden. On the other hand, in `aggressive` mode, all arcs between nodes in $\{v, i_1, \ldots, i_p\}$ are added to $\overrightarrow{M}_v$, thus forbidding any cycle starting and ending at v and passing through a subset of $\{i_1, \ldots, i_p\}$. This latter mode is equivalent to the vertex-based augmentations implemented by Roberti and Mingozzi, (2014).

Quickly, some augmentations performed in previous iterations may become unnecessary to guarantee the bound of the current iteration k. Therefore, we propose Algorithm 6 to reduce *ng*-memories $\overrightarrow{\mathcal{M}}_k$. This algorithm is based on the same DSSR technique suggested by Righini and Salani, (2008), and also implemented by Martinelli et al., (2014). A problem-dependent condition (pricing time, number of non-dominated labels, etc.) is used to trigger such reductions. Initially empty *ng*-memories are iteratively augmented in order to forbid the columns with the best reduced costs w.r.t. $\bar{\pi}_k$. Of course, those

Algorithm 5 *ng*-Memories Augmentation Algorithm

1: **procedure** augmentNgMemories($\mathcal{R}$, $\overrightarrow{\mathcal{M}}$, $\mathcal{A}$, Δ^{max}, α, β, `mode`)
2: **input** $\mathcal{R}$: set of target *ng*-routes, $\overrightarrow{\mathcal{M}}$: current *ng*-memories, $\mathcal{A}$: set of forbidden augmentations, Δ^{max}: maximum value allowed for $\gamma(\cdot)$, α: maximum cycle size in Phase 1, β: maximum number of *ng*-routes explicitly forbidden, `mode`: mode of augmentation.
3: **output:** new *ng*-memories
4:
5: $\mathcal{F}_H \leftarrow \emptyset$, $\mathcal{F}_R \leftarrow \emptyset$
6: **for each** $R \in \mathcal{R}$ **do**
7: phase $\leftarrow 1$
8: **for each** $H = (v, \ldots, v) \in \mathcal{H}(R)$ in non-decreasing order of size **do**
9: **if** $|H| > \alpha$ **and** phase $= 1$ **then**
10: **if** $\mathcal{H}(R) \cap \mathcal{F}_H = \emptyset$ **then**
11: phase $\leftarrow 2$
12: **else**
13: **break**
14: **if** cycleCanBeForbidden(H, $\overrightarrow{\mathcal{M}}$, Δ^{max}, $\mathcal{A}$) **then**
15: $\overrightarrow{\mathcal{M}} \leftarrow$ forbidCycle(H, $\overrightarrow{\mathcal{M}}$, `mode`)
16: $\mathcal{F}_H \leftarrow \mathcal{F}_H \cup \{H\}$, $\mathcal{F}_R \leftarrow \mathcal{F}_R \cup \{R\}$
17: **if** phase $= 2$ **then**
18: **break**
19: Stop if $|\mathcal{F}_R| \geq \beta$
20: **return** $\overrightarrow{\mathcal{M}}$

improvements are confined to $\overrightarrow{\mathcal{M}}_k$ and new iterations are performed as long as the best column generated is not *ng*-feasible w.r.t. $\overrightarrow{\mathcal{M}}_k$. The algorithm ends up with a set of reduced *ng*-memories $\overrightarrow{\mathcal{M}}$ such that $LB(\overrightarrow{\mathcal{M}}) = LB(\overrightarrow{\mathcal{M}}_k)$. Even though the final *ng*-memories are not necessarily minimal, in practice we have observed that Algorithm 6 often reduces significantly the size of the *ng*-memories.

Remark that we have adopted the same algorithm for augmenting *ng*-memories (i) in the fully dynamic *ng*-path relaxation, and (ii) in the *ng*-memories reduction algorithm, but with different parameters. In case (i), more aggressive augmentations are needed for the sake of convergence, whereas in case (ii) moderate augmentations are performed in order to get smaller final *ng*-memories. This is mainly controlled by parameter `mode` of procedure forbidCycle($\cdot$).

Algorithm 6 *ng*-Memories Reduction Algorithm

procedure reduceNGMemories($\overrightarrow{\mathcal{M}}$, π^*, α, β)
  input: $\overrightarrow{\mathcal{M}}$: current *ng*-memories, π^*: dual solution of $LP(\overrightarrow{\mathcal{M}})$, and parameters to augmentNgMemories($\cdot$)
  output: new *ng*-memories

  $\overrightarrow{\mathcal{M}}_{ori} \leftarrow \overrightarrow{\mathcal{M}}$, $\overrightarrow{\mathcal{M}} \leftarrow \emptyset$, *ng*-feasible $\leftarrow$ false
  while not *ng*-feasible **do**
    $\mathcal{R} \leftarrow$ LabelingAlgorithm($\overrightarrow{\mathcal{M}}$, π^*)
    if best route in $\mathcal{R}$ is *ng*-feasible w.r.t. $\overrightarrow{\mathcal{M}}_{ori}$ **then**
      ng-feasible $\leftarrow$ true
    else
      Define $\mathcal{A}$ as the set of augmentations that are not confined to $\overrightarrow{\mathcal{M}}_{ori}$
      $\overrightarrow{\mathcal{M}} \leftarrow$ augmentNgMemories($\mathcal{R}$, $\overrightarrow{\mathcal{M}}$, $\mathcal{A}$, ∞, α, β, `moderate`)
  return $\overrightarrow{\mathcal{M}}$

Algorithm 7 Test for Cycle Elimination

procedure cycleCanBeForbidden(H, $\overrightarrow{\mathcal{M}}$, $\mathcal{A}$, Δ^{max})
  input: $H = (a_0 = (v = i_0, i_1), a_1 = (i_1, i_2), \ldots, a_p = (i_p, i_{p+1} = v))$: target cycle, $\overrightarrow{\mathcal{M}}$: current *ng*-memories, $\mathcal{A}$: set of forbidden augmentations, Δ^{max}: maximum value allowed for $\gamma(\cdot)$.
  output: a flag indicating whether H can be forbidden or not

  for $q = 0$ **to** $p - 1$ **do**
    if $a_q \notin \overrightarrow{M}_v$ **then**
      $\gamma(i_{q+1}) \leftarrow |\{j \in V \setminus \{v\} : \delta^-(i_{q+1}) \cap \overrightarrow{M}_j \neq \emptyset\}|$
      if $\gamma(i_{q+1}) \geq \Delta^{max}$ **or** $(\overrightarrow{M}_v, a_q) \in A$ **then**
        return false
  return true

3.4 Branch-and-Price Algorithm

In this section, we describe the proposed branch-and-price algorithm (BP) for MLP. The solution of a node in our algorithm, outlined in Algorithm 9, is an iterative approach based on the fully dynamic *ng*-path relaxation described in Section 3.3.3. At each iteration k, we first solve $LP(\overrightarrow{\mathcal{M}}_k)$ by means of a two-stage column generation. The pricing network $G' = (V', A')$ has the following definition:

- $V' = \{s, 1, \ldots, n, t\}$, where nodes in $\{1, \ldots, n\}$ represent customers and source and sink nodes are associated with the depot. Hence, $V' = V \setminus \{0\} \cup \{s, t\}$.

Algorithm 8 Cycle Elimination

```
1: procedure forbidCycle(H, $\overrightarrow{\mathcal{M}}$, mode)
2:   input: H = (a_0 = (v = i_0, i_1), a_1 = (i_1, i_2), ..., a_p = (i_p, i_{p+1} = v)): target cycle,
     $\overrightarrow{\mathcal{M}}$: current ng-memories, mode: mode of augmentation.
3:
4:   for q = 0 to p − 1 do
5:     if mode = moderate then
6:       $\overrightarrow{M}_v \leftarrow \overrightarrow{M}_v \cup \{(i_q, i_{q+1})\}$
7:     else
8:       for t = q + 1 to p do
9:         $\overrightarrow{M}_v \leftarrow \overrightarrow{M}_v \cup \{(i_q, i_t), (i_t, i_q)\}$
```

- $A' = \{(s,i) : i \in \{1,\dots,n\}\} \cup \{(i,t) : i \in \{1,\dots,n\}\} \cup \{(i,j) : i,j \in \{1,\dots,n\}, i \neq j\}$
- The single resource indicates the number of arcs in a partial path, and hence $w_{ij} = 1$ for any arc $(i,j) \in A'$. Moreover, $(l_i, u_i) = (1, n)$ for a vertex $i \in \{1,\dots,n\}$; $(l_s, u_s) = (0,0)$ and $(l_t, u_t) = (n+1, n+1)$. Therefore, any *ng*-path generated visits exactly n customers, although it may contain cycles.
- Finally, the CEF $c_{ij} : \mathbb{Z} \to \mathbb{R}$ for an arc $(i,j) \in A'$ is the function $c_{ij}(w) = (n-w+1)t_{ij} - (\mu_i + \mu_j)/2$, where μ_i, $i \in \{1,\dots,n\}$, is the value of the dual variable associated with customer i, and $\mu_s = \mu_t = 0$.

Of course, branching constraints may also be present in $LP(\overrightarrow{\mathcal{M}}_k)$. In Stage 1 of column generation, dominance tests take into account only conditions (I), (II) and (III), thus at most a single *ng*-path is kept for a given vertex i and resource consumption w, which is the one with minimum reduced cost. This is a heuristic pricing intended to quickly generate good *ng*-paths. When Stage 1 fails to find an *ng*-path with negative reduced cost, we switch to Stage 2, where the exact pricing is solved. In both phases, the dual stabilization technique of Pessoa et al., (2017) is applied for the sake of convergence. Stage 1 (2) is solved by a mono-directional (bidirectional) labeling algorithm that returns at most 50 (300) *ng*-paths. The reader is referred to Section 3.4.1 for further details on the labeling algorithms.

If the node is the root, the first memories $\overrightarrow{\mathcal{M}}_1$ are equivalent to *ng*-sets of size 8 defined according to the classical distance-based rule — time-based for the case of MLP. Otherwise, they correspond to the final memories of the parent node, which are inherited by the child. In any node, a hybrid strategy for augmenting *ng*-memories may be used. Initially, we set `mode` ← `aggressive`. The method switches to `moderate` mode

Algorithm 9 Solution of a node

1: **procedure** solveNode($\overrightarrow{\mathcal{M}}$, B, α_I, β_I, α_R, β_R, Δ^{max})
2: $k \leftarrow 1$, $\overrightarrow{\mathcal{M}}_1 \leftarrow \overrightarrow{\mathcal{M}}$
3: `mode` $\leftarrow$ `aggressive`
4: **repeat**
5: Compute $(\bar{\lambda}_k, \bar{\pi}_k)$, an optimal primal-dual solution pair of $LP(\overrightarrow{\mathcal{M}}_k)$ + branching constraints B
6: **if** *ng*-memories reduction condition is satisfied **then**
7: $\overrightarrow{\mathcal{M}}_k \leftarrow$ reduceNGMemories($\overrightarrow{\mathcal{M}}_k$, $\bar{\pi}_k$, α_R, β_R)
8: `mode` $\leftarrow$ `moderate`
9: **continue**
10: Apply reduced cost fixing
11: Try to finish the node by enumeration
12: Let $\mathcal{R}$ be the set of *ng*-routes associated with the primal solution $\bar{\lambda}_k$
13: $\overrightarrow{\mathcal{M}}_{k+1} \leftarrow$ augmentNgMemories($\mathcal{R}$, $\overrightarrow{\mathcal{M}}_k$, $\emptyset$, Δ^{max}, α_I, β_I, `mode`)
14: $k \leftarrow k + 1$
15: **until** stop condition is met

if the computational time of a single call to the labeling algorithm exceeds a threshold value t^{red}. In this case, column generation is interrupted and the *ng*-memories reduction algorithm (see Section 3.3.3) is called.

As we have already discussed, the pricing problem corresponds to finding a least-cost *ng*-path in the resource constrained network G' defined in Section 3.2. However, in practice, we work with an extended network $G^{ext} = (V^{ext}, A^{ext})$, where:

$$V^{ext} = \{(i, w) : i, w \in \{1, \ldots, n\}\} \cup \{(s, 0), (t, n+1)\}$$
$$A^{ext} = \{((s, 0), (i, 1)) : i \in V \setminus \{0\}\} \cup \{((i, n), (t, n+1)) : i \in V \setminus \{0\}\}$$
$$\cup \{((i, w), (j, w+1)) : (i, j) \in A, i \neq 0, j \neq 0\}$$

Network G^{ext} is defined in such a way that resource constraints are naturally satisfied by any *ng*-path. Once an optimal dual solution $\bar{\pi}_k$ is available, a reduced cost fixing procedure is used to remove from A^{ext} the arcs that cannot be part of a solution that improves the current upper bound. For fixing an arc $((i, w), (j, w+1))$, one has to prove that the minimum reduced cost of an *ng*-path traversing this arc is above a given threshold. To compute this cost, the minimum reduced cost of a partial path ending at (i, w) and of a partial path from $(j, w+1)$ to the sink node are computed by a forward and a backward labeling algorithm, respectively. Such a procedure is well-known in the literature and has been applied in many routing problems (see, for instance, (Irnich et al., 2010; Roberti and Mingozzi, 2014; Pecin et al., 2017a)). Dual solution $\bar{\pi}_k$ is also used in

an enumerative procedure that tries to finish the node. As implemented by Roberti and Mingozzi, (2014), the enumeration is performed by a mono-directional labeling algorithm that computes only elementary paths, using completion bounds associated with *ng*-paths to prune unpromising partial paths. However, we abort the enumeration if the number of non-dominated labels is greater than 10 million. If enumeration finishes, either a single path representing the best integer solution for the node is returned, or all labels are eliminated, meaning that this solution does not improve the current upper bound.

Besides the obvious stop conditions (node is solved or pruned), Algorithm 9 stops if:

- **Stop condition I:** Labeling algorithm time has exceeded the threshold value t^{red} twice.
- **Stop condition II:** No column could be forbidden by augmentNgMemories($\cdot$) because of Δ^{max}.
- **Stop condition III:** For 5 times, the gap of the current iteration is less than 2% smaller than the one of the previous iteration.

We branch on an edge $\{i, j\}$ defined by a strong branching mechanism in the spirit of the works of Røpke, (2012) and Pecin et al., (2017a). Hence, vertices i and j must appear consecutively (either $i \rightarrow j$ or $j \rightarrow i$) in the solution of one child and must not be consecutive in the solution of the other child. The strong branching procedure evaluates a number of candidate edges that is based on an estimation for the size of tree rooted at the current node. As pointed out by Pecin et al., (2017a), the rationale is that the greater the estimation, the more the cost of evaluating a larger number of candidates will pay. The evaluation of candidate edges comprises three steps, with an increasing level of accuracy and a decreasing number of candidates. The three steps to select the branching edge in a node N are:

- **Step 0**. We define $\min\{100, TS(N)\}$ candidate edges, where $TS(N)$ is an estimation for the size of the tree rooted at node N. The estimation takes into account the average bound improvement in the branch history, following the model of Kullmann, (2009). $TS(N) = \infty$ if N is the root node. First, we select the best $\min\{|\mathcal{P}|, \min\{100, TS(N)\}/2\}$ candidate edges from a pool $\mathcal{P}$ containing the average scores of the candidates evaluated in previous executions of Step 2 in the whole branch history. The other candidates are the edges whose values are the closest to 0.5 in the current fractional solution.

- **Step 1**. For each candidate edge $\{i, j\}$ selected in Phase 0, we perform a rough evaluation of both children by solving the master LP of node N with the corresponding branching constraint, but without any column generation. Let Ω_N be the set of *ng*-routes present in the master LP of node N. Thus, the branching constraint is either $\sum_{R \in \Omega_N} a_{ij}^R \lambda_R = 0$ or $\sum_{R \in \Omega_N} a_{ij}^R \lambda_R = 1$, where a_{ij}^R is a binary value assuming 1 if i and j appear consecutively in R. The optimal solution values of the two LPs are used to evaluate the candidate according to the product rule of (Achterberg, 2007). The best $\min\{5, TS(N)/10\}$ candidates are selected to Step 2.

- **Step 2**. This step uses the same approach as for Step 1, but heuristic column generation (Stage 1) is applied when evaluating the children. The selected edge is the one with the best score in this step, also according to the product rule. If a candidate is evaluated in this step for the first time, then it is added to the pool $\mathcal{P}$. Otherwise, its average score is updated in the pool.

We now provide more information on the implementation of the labeling algorithm and of MPLD.

3.4.1 Labeling Algorithm

Following many related works in the literature — for instance, (Martinelli et al., 2014) and (Pecin et al., 2017a) — we have adopted the concept of buckets in our implementation of the general method described in Algorithm 3. A *forward bucket* $\overrightarrow{B}(j, w)$ is data structure that stores all non-dominated labels associated with *forward* $s - j$ paths with resource consumption w. Therefore, for each vertex j we define buckets $\overrightarrow{B}(j, w)$, $\forall w \in \{l_j, \ldots, u_j\}$. To accelerate the algorithm, labels in a bucket are kept sorted by non-decreasing order of cost and dominance checks are performed only between labels of the same bucket, just as implemented by Pecin et al., (2017a). Buckets are considered by the algorithm in a non-decreasing order of resource consumption. When a bucket $\overrightarrow{B}(j, w)$ is reached, we extend all labels in $\overrightarrow{B}(j, w)$ over all arcs $(j, v) \in A$. At the end, the optimal $s - t$ paths will be stored in the buckets associated with the sink node t.

The algorithm outlined above is called mono-directional since all labels kept correspond to partial paths from the source node to some node j and arcs are traversed in their regular directions. However, in our implementation we have used a bidirectional labeling algorithm, which generates more diversified sets of $s - t$ paths and is typically faster than its mono-directional counterpart. A *backward path* corresponds to a partial

path from the sink to some node j obtained by traversing arcs in their reverse directions. Labels corresponding to such paths are stored in *backward buckets* $\overleftarrow{B}(j, w)$. In the forward (backward) labeling algorithm, one computes non-dominated labels for buckets $\overrightarrow{B}(j, w)$ ($\overleftarrow{B}(j, w)$) such that $w \leq w^*$ ($w \geq w^*$), where $w^* = \lfloor \frac{n}{2} \rfloor$. Then, a concatenation procedure is execute to build complete $s - t$ paths from the partial forward and backward paths. The reader is referred to (Righini and Salani, 2006) for further references on bidirectional labeling.

3.4.2 Multiple Partial Label Dominance

Let us consider again labels $L(P')$ and $L(P)$ such that conditions stated in Proposition 3.1 are satisfied, and define $\Pi' = \Pi(P') \setminus \Pi(P)$. As we have discussed, the extensions to vertices $j \in V \setminus \bigcup_{v \in \Pi'} M_v$ can be avoided for label $L(P)$. For example, in Figure 3.3, one can verify that any extension to vertices $j \notin M_3$ can be avoided for label $L(P_4)$ because of label $L(P_3)$, and thus set $\xi(P_4)$ is augmented as $\xi(P_4) \leftarrow \xi(P_4) \cup \{2, 5, 6\}$. Such an operation is performed several times during the course of the labeling algorithm and an efficient implementation is required, otherwise the gains incurred by MPLD will not pay off. In what follows, we will describe two approaches that we have tried to take advantage of this new dominance rule.

Explicit Representation of $\xi(\cdot)$ We first tried an explicit representation of set $\xi(P)$ inside label $L(P)$.

- **Bitmap representation**: Let $L(P)$ be a label with $v(P) = i$. Set $\xi(P)$ is represented as a bitmap implemented over a 64-bit integer. Each bit is associated with a different vertex $j \in V \setminus \{i\}$ and is set to 1 to only if $j \in \xi(P)$. Before extending label $L(P)$ over an arc $(i, j) \in A$, we first retrieve the position in the bitmap associated with vertex j (which is stored in the arc itself) and check in constant time through bitwise operations if this bit is set to 1. If so, the extension is not performed because $j \in \xi(P)$. Since the bitmap has 64 positions, for large instances it is not possible to assign a position for each $j \in V \setminus \{i\}$. In this case, MPLD will not be completely explored. More precisely, let E_i be a set of 63 vertices $j \in V \setminus \{i\}$ that minimizes $\sum_{j \in E_i} t_{ij}$, i.e, E_i contains the 63 closest vertices according to the travel times. We assign the first 63 positions of the bitmap to the vertices of E_i, while the position of any vertex $j \notin E_i$ is 64. The 64th bit of the bitmap is then set to 0 so that an extension over an arc (i, j) such that $j \notin E_i$ is never avoided because of MPLD.

- **Precomputed union of memories**: The number of different possible sets $\Pi(P)$ for a path P such that $v(P) = i$ is $2^{\gamma(i)}$. Thus, for two labels $L(P_1)$ and $L(P_2)$ such that $v(P_1) = v(P_2) = i$, the number of different possible sets $\Pi' = \Pi(P_1) \setminus \Pi(P_2)$ is also $2^{\gamma(i)}$. For reasonable values of $\gamma(i)$, it is practical to precompute sets $V \setminus \bigcup_{v \in \Pi'} M_v$ for every possible Π'. In fact, we have observed that the gains of MPLD are diminished if these sets are not precomputed. Suppose that label $L(P_1)$ partially dominates label $L(P_2)$. Thus, the test of dominance of label $L(P_2)$ by label $L(P_1)$ comprises the following steps. First, we compute set $\Pi' = \Pi(P_1) \setminus \Pi(P_2)$. Second, a bitmap b_1 representing $V \setminus \bigcup_{v \in \Pi'} M_v$ is retrieved. Actually, b_1 represents $E_i \cap (V \setminus \bigcup_{v \in \Pi'} M_v)$ since in this implementation the bitmaps keep track of at most 63 extensions. Lastly, we update b_2 as $b_2 \leftarrow b_1 \otimes b_2$, where b_2 is the bitmap representing $E_i \cap \xi(P_2)$ and $\otimes$ is the bitwise OR operator.

Computational experiments showed that the computational time speedup incurred by this approach is not significant. The main limitation is that one needs to restrict the cardinality of sets $\xi(\cdot)$ to at most 63 in order to quickly manipulate the bitmaps representing them. In this case, MPLD is not completely explored. Moreover, we have adopted a distance-based criterion to decide the 63 extensions tracked, which may be a very crude criterion in some cases. For example, an extension over a *long arc* (i.e., an arc connecting two customers that are far away from each other) will probably generate a label $L(P)$ with a large cost, but with few customers in set $\Pi(P)$. This latter attribute makes very hard to dominate label $L(P)$, increasing the number of dominance checks in the destination bucket. Finally, as we will see in Section 3.5, for many instances large values (up to 63) of $\gamma(\cdot)$ are needed, breaking this approach. A dynamic definition of tracked extensions or a more advanced implementation with larger bitmaps would probably improve results, but the following approach is simpler and mitigates all aforementioned problems.

Implicit Representation of $\xi(\cdot)$

Recall that, in our implementation of the labeling algorithm, buckets are considered in a predefined order and labels of a same bucket are extended in a non-decreasing order of cost. Moreover, we extend all labels of a bucket over an arc, then over another arc and so on. Let us consider a (forward or backward) bucket $B(i, w)$ and an arc (i, j). Suppose that a label $L(P') \in B(i, w)$ has already been extended to a label $L(P' + j)$. Now let $L(P) \in B(i, w)$ be a label that has not yet been extended over (i, j) such that $\Pi(P' + j) \subseteq \Pi(P + j)$. Since its cost is larger, $L(P + j)$ will be dominated by label $L(P' + j)$ and thus one can avoid the extension of $L(P)$ over (i, j) — this is just an

example of MPLD. In general, for each bucket $B(i, w)$ and arc (i, j), we keep a list of bitmaps representing sets $\Pi(\cdot)$ of already extended labels. Then, before extending a label $L(P) \in B(i, w)$ over (i, j), we first compute $\Pi(P + j)$ and check if it is a superset of some $\Pi(P' + j)$ contained in the list. If so, the extension is not necessary and we proceed to the next label, otherwise we complete the extension of the label and update the list. Of course, if $\Pi(P + j) = \{j\}$, then we can stop extending labels in $B(i, w)$ over (i, j). Furthermore, there is no need to keep the list after considering the extension of all labels in $B(i, w)$ over (i, j). This approach better explores the potential of MPLD and typically yields a significant speedup in the pricing time, as will be seen in the next section.

3.5 Computational Experiments

This section reports our computational experiments over 40 TSPLIB instances with up to 200 vertices. The proposed BP algorithm was implemented in C++ over the BaPCod platform of Vanderbeck et al., (2017). The LP solver adopted is IBM CPLEX Optimizer version 12.6.0. All experiments were conducted on an Intel Xeon E5-2680 v3, running at 2.5 GHz with a single thread. We will compare our results to those obtained by the algorithm of Roberti and Mingozzi, (2014) on an Intel Xeon X7350, running at 2.93 GHz. According to the CPU benchmark website `www.cpuboss.com`, the single thread performance of our CPU is about 1.5 times better than the one of Roberti and Mingozzi, (2014). Therefore, our computational times are increased by this factor in Table 3.1. Preliminary experiments suggested the following parameter values: $t^{red} = 100$ seconds, $\alpha_R = \alpha_I = 5$, $\beta_R = \beta_I = 200$ and $\Delta^{max} = 63$. The solutions found are provided in Appendix A

3.5.1 Main Results

Table 3.1 presents the main results of the proposed BP and the results of Roberti and Mingozzi, (2014) over instances with up to 152 vertices. In this case, we set a time limit of 2 days for BP. For what concerns the results of Roberti and Mingozzi, (2014), we report the following data: lb, the final lower bound; t_{lb}, the computational time to obtain such lower bound; t_{tot}, the total computational time; and gap, the percentage gap before applying enumeration. A symbol "-" in column t_{tot} indicates that enumeration failed to find the optimal solution. The results of the proposed method are divided into root node and complete BP. For the root node, we report the first and last lower bounds obtained,

lb^0 and lb^k, where k is the number of *ng*-memories augmentations performed. Further, we present the average and maximum values of $\gamma(\cdot)$ and the time spent at the root node. We also indicate if the method has switched to `moderate` mode in the root node. For the complete BP, we report the final bounds and gap, the total computational time and the number of nodes solved. All computational times in Table 3.1 are given in seconds.

We can observe in Table 3.1 that the proposed BP outperforms the method of Roberti and Mingozzi, (2014) in most instances. In particular, 6 instances were solved for the first time and the BKS for instance kroA150 was improved from 1831766 to the optimal value 1825769 (see Figure 3.4). The main advantage of our method is the possibility of dealing with larger *ng*-memories without combinatorial explosion. For example, for the 6 instances solved only by BP, the maximum values of $\gamma(\cdot)$ are greater than 30, reaching 62 in eil101. This is possible only because of the proposed generalized definition of *ng*-sets in terms of arcs. Instances pr136, pr152 and kroB150 could not be solved, but we remark that "pr" instances have a special structure with a lot of symmetry (see Figure 3.5). However, we will show in the next section that a specific parameterization of our method is capable of solving those 3 instances.

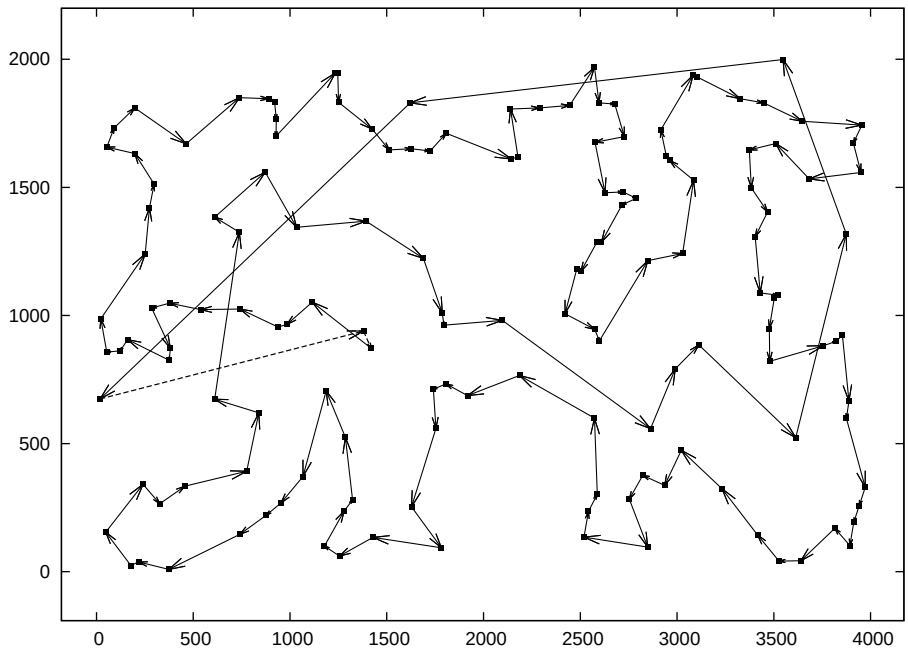


Figure 3.4: Optimal solution of kroA150. The dashed arcs are the ones entering or leaving the depot.

Table 3.2 shows the results of BP over large TSPLIB instances with up to 200 vertices. To our knowledge, this is the first time that such large instances are considered by an exact method for MLP. The upper bounds were computed by the method of Silva et al.,

Table 3.1: Results of BP over TSPLIB instances used by Roberti and Mingozzi, (2014). Computational times are normalized according to the CPU benchmark website `www.cpuboss.com`.

		Roberti and Mingozzi, (2014)				Proposed Method											
						Root Node							B&P				
Instance	z^*_{lit}	lb	t_{lb}	t_{tot}	gap	lb^0	lb^k	k	switched mode	Max. $\gamma(\cdot)$	Avg. $\gamma(\cdot)$	t_{root}	lb	ub	gap	nodes	t_{tot}
dantzig42	12528	12528	5	5	0.00	12528	12528	0	no	8	8.0	4.7	12528	12528	0.00	1	4.7
swiss42	22327	22327	3	3	0.00	22327	22327	0	no	8	8.0	2.1	22327	22327	0.00	1	2.1
att48	209320	209320	7	7	0.00	209320	209320	0	no	8	8.0	8.5	209320	209320	0.00	1	8.5
gr48	102378	102378	15	15	0.00	102378	102378	0	no	8	8.0	7.2	102378	102378	0.00	1	7.2
hk48	247926	247926	19	19	0.00	247926	247926	0	no	8	8.0	10.1	247926	247926	0.00	1	10.1
eil51	10178	10178	10	10	0.00	10178	10178	0	no	8	8.0	6.5	10178	10178	0.00	1	6.5
berlin52	143721	143721	10	10	0.00	143721	143721	0	no	8	8.0	10.1	143721	143721	0.00	1	10.1
brazil58	512361	512361	36	36	0.00	512361	512361	0	no	8	8.0	17.7	512361	512361	0.00	1	17.7
st70	20557	20557	61	61	0.00	20557	20557	0	no	8	8.0	24.0	20557	20557	0.00	1	24.0
eil76	17976	17976	91	91	0.00	17976	17976	0	no	8	8.0	26.6	17976	17976	0.00	1	26.6
pr76	3455242	3423016	169	223	0.93	3375941	3455242	2	no	11	8.6	169.1	3455242	3455242	0.00	1	169.3
gr96	2097170	2087792	365	369	0.45	2061228	2097170	2	no	11	8.7	215.3	2097170	2097170	0.00	1	215.3
rat99	57986	57541	540	-	0.77	57064	57623	17	yes	38	25.7	1467.9	57986	57986	**0.00**	3	3180.4
kroA100	983128	980428	679	688	0.27	968668	983128	3	no	15	10.3	337.2	983128	983128	0.00	1	337.2
kroB100	986008	986008	197	197	0.00	986008	986008	0	no	8	8.0	75.2	986008	986008	0.00	1	75.2
kroC100	961324	957564	409	428	0.39	939298	961324	5	no	22	14.7	815.7	961324	961324	0.00	1	815.9
kroD100	976965	972814	708	729	0.42	957467	976965	2	no	13	9.2	290.0	976965	976965	0.00	1	290.0
kroE100	971266	971266	266	266	0.00	971266	971266	0	no	8	8.0	87.7	971266	971266	0.00	1	87.7
rd100	340047	339919	317	317	0.04	337230	340047	1	no	10	8.2	213.1	340047	340047	0.00	1	213.3
eil101	27513	27235	630	-	1.01	26892	27319	17	yes	62	35.1	4943.9	27513	27513	**0.00**	3	6238.9
lin105	603910	603910	154	154	0.00	603910	603910	0	no	8	8.0	85.5	603910	603910	0.00	1	85.5
pr107	2026626	2026626	246	246	0.00	2007255	2026626	2	no	12	9.1	470.9	2026626	2026626	0.00	1	470.9
gr120	363454	360460	4685	-	0.82	353054	359970	18	yes	41	25.2	10532.0	363454	363454	**0.00**	11	79351.6
pr124	3154346	3154346	14080	14080	0.00	3001043	3154346	21	yes	31	20.2	58283.6	3154346	3154346	0.00	1	58283.6
bier127	4545005	4545005	723	723	0.00	4513768	4545005	1	no	12	8.3	307.7	4545005	4545005	0.00	1	307.7
ch130	349874	348015	2385	-	0.53	341809	348013	16	yes	43	25.7	3767.8	349874	349874	**0.00**	3	7810.0
pr136	6199268	6160253	17190	-	0.63	5908131	6155720	23	yes	31	22.7	15845.4	6175226	6199268	0.39	7	259200.0
gr137	4061498	4025358	7844	-	0.89	3952022	4017150	15	yes	33	19.8	12140.2	4061498	4061498	**0.00**	15	65614.2
pr144	3846137	3841847	8043	8061	0.11	3671878	3846137	4	no	13	9.9	6101.5	3846137	3846137	0.00	1	6101.6
ch150	444424	444118	2585	2588	0.07	438450	444424	6	no	22	12.6	1742.6	444424	444424	0.00	1	1743.0
kroA150	1831766	1818024	5187	-	0.75	1794351	1813870	13	yes	33	17.4	7250.5	1825769	**1825769**	**0.00**	15	93566.8
kroB150	1793204	1775914	20120	-	0.96	1720498	1752709	17	yes	42	24.7	12265.3	1766343	1793204	1.52	19	259200.0
pr152	5064566	4931382	31218	-	2.63	4803249	4984428	15	yes	18	13.2	127472.1	4988337	5064566	1.53	3	259200.0

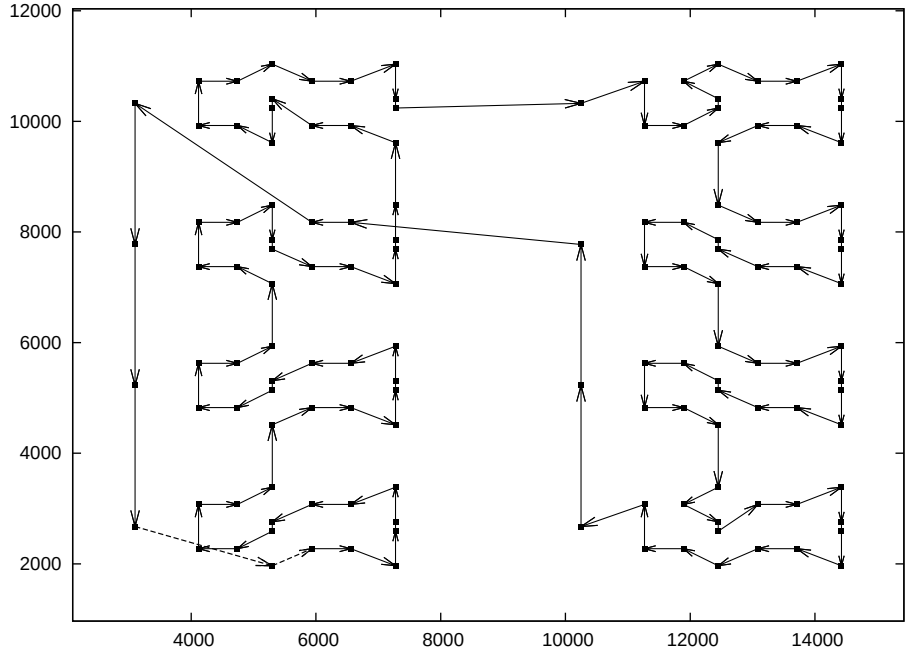


Figure 3.5: Optimal solution of pr136. The dashed arcs are the ones entering or leaving the depot.

Table 3.2: Results of BP over large TSPLIB instances

		Root Node							B&P				
Instance	z^*_{lit}	lb^0	lb^k	k	switched mode	Max. $\gamma(\cdot)$	Avg. $\gamma(\cdot)$	t_{root}	lb	ub	gap	nodes	t_{tot}
u159	2972030	2892904	2960764	25	yes	43	26.65	19881.7	2972030	2972030	**0.00**	3	21437.5
si175	1808532	1801860	1808195	20	yes	38	22.04	22874.3	1808532	1808532	**0.00**	3	24004.1
brg180	174750	164672	174750	8	no	18	13.42	3353.6	174750	174750	**0.00**	1	3353.7
rat195	218675	216725	217830	13	yes	50	25.50	3529.6	218632	**218632**	**0.00**	11	55814.8
d198	1186049	1128131	1144047	12	yes	23	10.97	30457.7	1147364	1186050	3.37	7	172800.0
kroA200	2672437	2600209	2631523	15	yes	45	24.39	20536.9	2637610	2672438	1.32	17	172800.0
kroB200	2669515	2580374	2636041	15	yes	30	15.08	13809.2	2643616	2669516	0.98	13	172800.0

(2012). Instances u159, si175, brg180 and rat195 were solved in reasonable computational times and small BP trees, but BP finished with considerable gaps for the other instances. For rat195, BP found an optimal solution with cost 218632, an improvement of 43 units over the heuristic solution found by the method of Silva et al., (2012) (see Figure 3.6). In spite of the new contributions presented in this chapter, MLP instances with about 200 vertices still seem to be very challenging.

3.5.2 Longer runs with `moderate` mode

Here we show that our method can solve some more hard instances by using a different parameterization. The idea of this parameterization is to augment *ng*-memories very slowly and over a large number of iterations. More precisely, we adopt `moderate` mode in

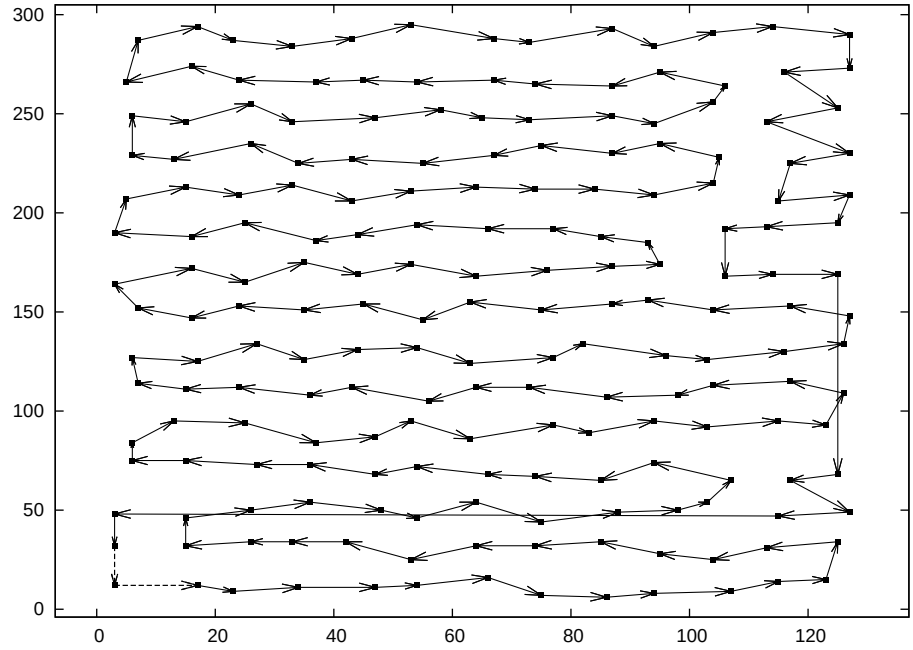


Figure 3.6: Optimal solution of rat195. The dashed arcs are the ones entering or leaving the depot.

Table 3.3: Results of BP over hard instances with a different parameterization

		Root Node						B&P				
Instance	z^*_{lit}	lb^0	lb^k	k	Max. $\gamma(\cdot)$	Avg. $\gamma(\cdot)$	t_{root}	lb	ub	gap	nodes	t_{tot}
pr136	6199268	5908131	6182837	125	52	37.43	45071.2	6199268	6199268	**0.00**	15	187356.3
kroB150	1793204	1720498	1760182	103	63	44.98	13371.8	1786546	**1786546**	**0.00**	23	293479.1
pr152	5064566	4803249	4980497	110	26	17.22	38839.8	5064566	5064566	**0.00**	3	281061.9
d198	1186049	1128131	1145300	101	31	17.68	27433.4	1152702	1186050	2.89	17	432000.0
kroA200	2672437	2600209	2639043	100	63	48.40	32154.6	2651933	2672438	0.77	18	432000.0
kroB200	2669515	2580374	2646679	115	63	39.91	61444.4	2653034	2669516	0.62	15	432000.0

all iterations of Algorithm 9, set a time limit of 5 days for BP and change stop conditions I and III as below.

- **Stop condition I:** Labeling algorithm time has exceeded the threshold value $\mathbf{t^{red} = 150}$.
- **Stop condition III:** For **100** times, the gap of the current iteration is less than 2% smaller than the one of the previous iteration.

Table 3.3 shows the results obtained. For the first time, instances pr136, pr152 and kroB150 were solved to optimality. Therefore, we solved all instances that were not solved by Roberti and Mingozzi, (2014). Furthermore, the BKS for kroB150 was improved from 1793204 to the optimal value 1786546, an improvement of 0.03% (see Figure 3.7). Still,

we should point out that such a parameterization is useful only for hard instances. In general, BP has a worse performance if a weaker tailing off condition is used.

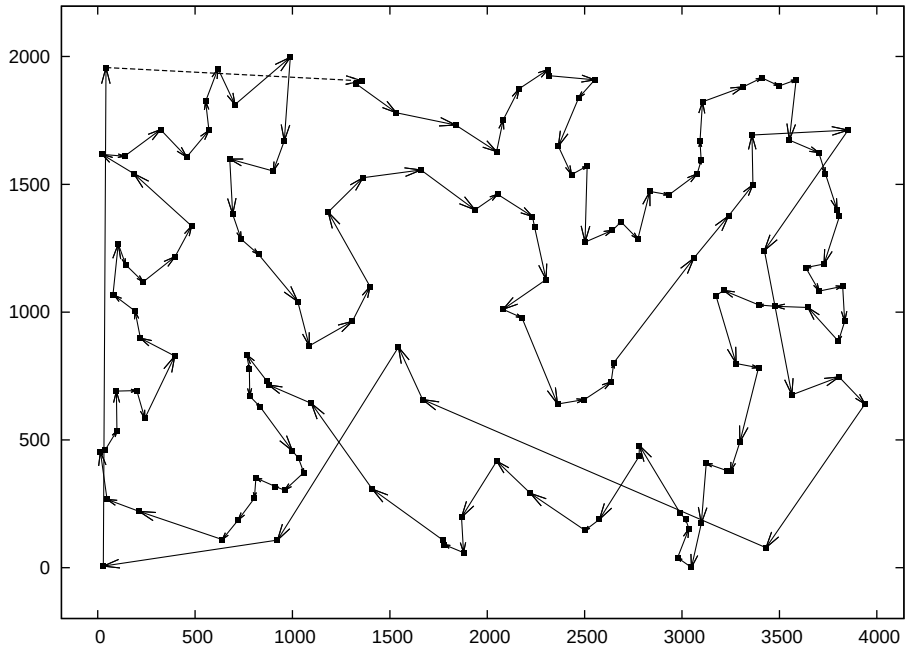


Figure 3.7: Optimal solution of kroB150. The dashed arcs are the ones entering or leaving the depot.

3.5.3 Multiple Partial Label Dominance

Table 3.4 shows the performance of the labeling algorithm with and without MPLD. The average results presented in that table concern the exact calls to the labeling algorithm in the first descent of column generation with *ng*-sets of a given fixed size in $\{8, 12, 16\}$. A representative subset of the instances was used. With *ng*-sets of size 16, instances pr124, pr144 and pr152 exceeded a time limit of 300 seconds for a single call of the labeling algorithm, thus no results are reported in those cases. It can be seen in Table 3.4 that the new dominance rule has a significant impact on the labeling algorithm. The average number of extensions decreased by an order of magnitude for any tested instance. This allowed the algorithm to achieve remarkable average speedups of 4.37, 6.01 and 7.28 in computational time for *ng*-sets of sizes 8, 12 and 16, respectively. Of course, with good lower and upper bounds and some rounds of reduced cost fixing, the pricing networks gets sparser and those speedups are smaller, but still considerable. Surprisingly, for most instances, the percentage of non-dominated labels that are never extended — column **MPLD (%)** — is significantly large. This means that not rarely a label is completely dominated by a set of two or more labels, but not by a single label as required by the

classical dominance rule.

Table 3.4: Performance of the labeling algorithm with/without multiple partial label dominance

		Avg. time (s)			**Avg. number of extensions**			**MPLD**
Instance	$\|N_i\|$	without	with	factor	without ($\times 10^6$)	with ($\times 10^6$)	factor	(%)
gr120	8	1.16	0.33	3.51	17.90	1.97	9.08	24.54
pr124	8	1.83	0.48	3.82	30.38	2.60	11.68	7.76
bier127	8	1.78	0.42	4.23	28.37	2.56	11.07	11.56
ch130	8	1.78	0.42	4.25	27.06	2.58	10.47	16.83
pr136	8	2.35	0.55	4.26	36.67	3.08	11.89	19.58
gr137	8	1.63	0.46	3.53	28.54	2.99	9.55	17.73
pr144	8	8.51	1.28	6.64	108.24	5.38	20.11	2.01
ch150	8	2.26	0.61	3.71	36.25	3.75	9.66	24.69
kroA150	8	2.74	0.69	3.98	42.51	3.90	10.89	18.29
kroB150	8	2.96	0.73	4.07	44.23	3.97	11.14	17.64
pr152	8	7.77	1.27	6.11	108.04	5.87	18.41	5.08
Average for ng 8				**4.37**			**12.18**	**15.06**
gr120	12	4.74	0.82	5.76	53.77	2.98	18.04	19.40
pr124	12	38.77	15.66	2.48	203.96	9.73	20.96	4.03
bier127	12	8.40	1.46	5.77	85.20	4.24	20.09	11.58
ch130	12	7.60	1.14	6.68	81.84	3.98	20.58	16.33
pr136	12	11.40	1.80	6.32	120.52	5.44	22.17	13.50
gr137	12	7.87	1.41	5.59	95.40	4.93	19.33	13.96
pr144	12	174.41	33.38	5.23	555.34	19.80	28.05	1.62
ch150	12	8.55	1.27	6.72	102.46	5.09	20.11	22.03
kroA150	12	12.12	1.62	7.48	129.82	5.82	22.32	16.56
kroB150	12	14.06	2.04	6.90	140.31	6.37	22.02	15.14
pr152	12	212.33	29.70	7.15	731.47	23.54	31.08	2.28
Average for ng 12				**6.01**			**22.25**	**12.40**
gr120	16	28.68	5.21	5.50	158.58	6.84	23.18	12.30
pr124	16	-	-	-	-	-	-	-
bier127	16	57.66	8.99	6.42	256.19	9.41	27.22	9.46
ch130	16	35.69	4.35	8.21	210.17	7.62	27.58	13.70
pr136	16	65.27	8.96	7.29	334.53	13.12	25.51	10.41
gr137	16	48.09	8.84	5.44	286.45	11.56	24.78	10.08
pr144	16	-	-	-	-	-	-	-
ch150	16	40.89	5.12	7.98	279.85	9.47	29.55	16.99
kroA150	16	66.00	6.69	9.86	360.41	11.21	32.16	13.62
kroB150	16	89.06	11.86	7.51	409.30	13.98	29.28	12.40
pr152	16	-	-	-	-	-	-	-
Average for ng 16				**7.28**			**27.41**	**12.37**

3.5.4 Modes of *ng*-memories augmentation

The general conclusions suggested by our computational experience with the modes of *ng*-memories augmentation can be summarized as follows: (i) aggressive augmentations achieve a given bound in fewer iterations, however the pricing time may increase quickly; on the other hand, (ii) moderate augmentations need many more iterations to achieve the same bound, however the pricing time remains much more controlled. Figure 3.8 presents results on instance kroB150 that support those conclusions and also to show that the hybrid strategy may decrease the overall computational time. The charts in the left part of the figure plot the lower bound (x axis) vs average pricing time (y axis) at

each iteration of each augmentation mode. The charts in the right plot, for the same runs, lower bound vs total elapsed time. All the runs reported in the figure disabled the enumeration procedure.

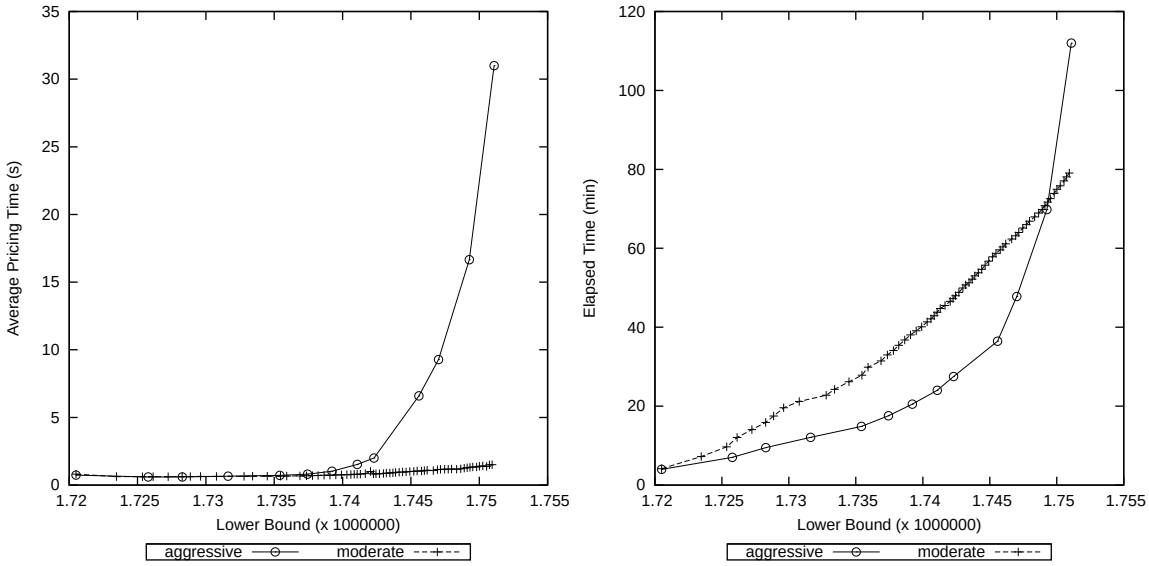
We can clearly observe (i) and (ii) in Figure 3.8 (a). For example, the run with aggressive augmentations produced the bound around 1.745×10^6 after 10 iterations, in a total elapsed time of 36 minutes. On the other hand, the run with moderate augmentations needed 43 iterations to produce the same bound, in a total elapsed time of 58 minutes. However, the average pricing times in the latter run were very controlled (around 1 second along all the 43 iterations), while in former run the average pricing time began to increase a lot after the 7th iteration. Eventually, the very large pricing times made the run with aggressive augmentations slower than the one with moderate augmentations (112 minutes against 79 minutes for reaching a bound around 1.751×10^6).

The idea of the hybrid strategy is to take advantage of the aggressive augmentations in the first iterations (to converge faster to a given bound), and switch to `moderate` mode when the pricing time becomes too large. For example, as shown in Figure 3.8 (b), the run with the hybrid strategy (using $t_{red} = 10$) switched to `moderate` mode in the 11th iteration. The ng-memories reduction algorithm made the pricing time to decrease from 6.6 to 1 second. After that, the hybrid achieves the bound of 1.751×10^6 in its 31st iteration, with an elapsed time of 55 minutes.

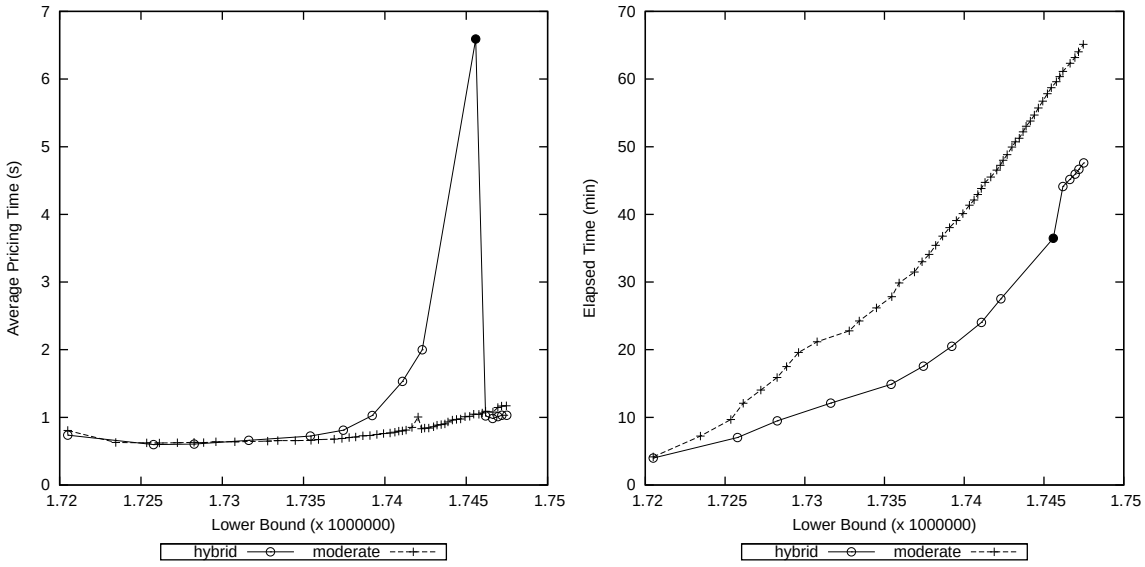
Our computational experience shows that the hybrid strategy is also beneficial for the robustness of the proposed method. Even though some instances are better solved if only moderate augmentations are performed (e.g. eil101 and gr120), instance pr144 cannot be solved (due to slow convergence) in reasonable time without aggressive augmentations.

3.6 Conclusions

This chapter dealt with the Minimum Latency Problem (MLP), a variant of the Traveling Salesman Problem (TSP) where the objective is to minimize the sum of waiting times of customers. A branch-and-price (BP) algorithm over a set partitioning formulation was introduced, where columns are *ng*-paths. As implemented by Roberti and Mingozzi, (2014), our algorithm is based on dynamically defined *ng*-memories. The proposed BP benefits from well-known elements of efficient exact method for routing problems, such as dual stabilization, reduced cost fixing, route enumeration and strong branching.



(a) Aggressive versus moderate augmentations on instance kroB150.



(b) Hybrid versus moderate augmentations on instance kroB150. The filled circle indicates the last iteration of the hybrid strategy in which aggressive augmentations were performed ($t_{red} = 10$).

Figure 3.8: Modes of augmentation: impact on pricing and elapsed times.

Although those elements are very important for the good performance of BP, the main sources of efficiency of our method are the new features for the *ng*-path relaxation. For example, the new dominance rules typically speeds up the labeling algorithm by factors between 4 and 8. Furthermore, the proposed generalized definition of *ng*-sets in terms of arcs opened the way for less harmful augmentations of *ng*-memories (moderate augmentations), in contrast to the vertex-based augmentations introduced by Roberti and Mingozzi, (2014) (aggressive augmentations). Nevertheless, our experiments showed that a good strategy for augmenting *ng*-memories is to start with aggressive augmentations, and then switch to moderate ones if the pricing time is too large. The proposed *ng*-memories reduction algorithm is beneficial in this transition, reverting previous augmentations that are not needed to attain the current bound. We should mention that we did try to use this reduction of *ng*-memory more frequently, but it is time-consuming for large instances and affects the convergence of Algorithm 9. Still, we believe that the combination of augmentations and reductions of *ng*-memories deserves further investigation.

All the 9 instances not solved by Roberti and Mingozzi, (2014) were solved. Also, BP could solve the larger instances u159, si175, br180 and rat195, but could not solve d198, kroA200 and kroB200. In general, MLP instances with about 150 vertices (or more) are still very challenging. As for future works, we intend to investigate the impact of the new features for the *ng*-path relaxation on other routing problems.

Chapter 4

Conclusions

This thesis presented theoretical and practical contributions on the use of column generation over set partitioning formulations. The contributions are generic and can be potentially applied for solving several combinatorial problems. In particular, they are closely related to state-of-the-art exact algorithms for several vehicle routing problems.

On the theoretical side, we studied the set packing and set partitioning polytopes associated with a binary n-row matrix having all $2^n - 1$ non-empty columns, denoted as $CSPP_{\leq}(n)$ and $CSPP_{=}(n)$, respectively. The motivation for investigating these polytopes comes from the application of column generation for solving set partitioning problems. We showed the precise relation between these polytopes: with very few exceptions, every facet-inducing inequality for $CSPP_{\leq}(n)$ is also facet-inducing for $CSPP_{=}(n)$, and vice-versa. Thus, the study of $CSPP_{=}(n)$ reduces to the study of $CSPP_{\leq}(n)$. For example, this property is useful for proving that an inequality is facet-inducing for $CSPP_{=}(n)$ by showing that it is facet-inducing for $CSPP_{\leq}(n)$, which is often easier. We also studied rank 1 facets of these polytopes. In detail, we characterized the set of multipliers that induce rank 1 clique facets and described infinite families of multipliers that induce facets for arbitrarily large dimensions. Regarding these theoretical contributions, promising research avenues include:

- Investigating whether a modification of Theorem 2.1 holds for more general cases – a promising case is when the matrix A contains all singleton columns.
- Investigating the practical behavior of the new rank 1 facets found in this work.

On the practical side, this thesis presented a new branch-and-price algorithm for the Minimum Latency Problem, a variant of the classical Traveling Salesman Problem.

The proposed algorithm relies on well-known elements of efficient exact methods for routing problems, such as dual stabilization, reduced cost fixing, route enumeration and strong branching. However, the main sources of efficiency of our method are the new features for the *ng*-path relaxation, which is the state-of-the-art route relaxation employed by several exact algorithms for routing problems. The proposed method was capable of solving all the instances solved by Roberti and Mingozzi, (2014), besides all the 9 instances that they could not solve. Also, larger instances with up to 200 (never considered before by exact methods) could also be solved. As for future works, we intend to investigate the impact of the new features for the *ng*-path relaxation on other routing problems.

References

Abeledo, H. et al. "The time dependent traveling salesman problem: polyhedra and algorithm". In: *Mathematical Programming Computation* 5.1 (2013), pp. 27–55 (cit. on pp. 35, 36, 40).

Achterberg, T. "Constraint integer programming". PhD thesis. Technische Universitat Berlin, 2007 (cit. on p. 53).

Afrati, Foto et al. "The complexity of the travelling repairman problem". In: *RAIRO-Theor. Inf. Appl.* 20.1 (1986), pp. 79–87 (cit. on pp. 35, 36).

Araóz, J. "Polyhedral Neopolarities". PhD thesis. University of Waterloo, 1974 (cit. on p. 17).

Archer, A. and A. Blasiak. "Improved approximation algorithms for the minimum latency problem via prize-collecting strolls". In: *Proceedings of the 21th Annual ACM-SIAM Symposium on Discrete Algorithms*. Austin, Texas, 2010, pp. 429–447 (cit. on p. 36).

Archer, A. and D. P. Williamson. "Faster approximation algorithms for the minimum latency problem". In: *Proceedings of the 40th Annual ACM-SIAM Symposium on Discrete algorithms*. Baltimore, Maryland, 2003, pp. 88–96 (cit. on p. 35).

Archetti, C., N. Bianchessi, and M. G. Speranza. "A column generation approach for the split delivery vehicle routing problem". In: *Networks* 58.4 (2011), pp. 241–254 (cit. on p. 13).

Balas, E. and E. Zemel. "Graph substitution and set packing polytopes". In: *Networks* 7.3 (1977), pp. 267–284 (cit. on p. 16).

Balas, E. "Some Valid Inequalities for the Set Partitioning Problem". In: *Annals of Discrete Mathematics* 1 (1977), pp. 13 –47 (cit. on pp. 17, 26).

Balas, E. and M. W. Padberg. "Set Partitioning: A Survey". In: *SIAM Review* 18.4 (1976), pp. 710–760 (cit. on p. 17).

Balas, E. and E. Zemel. "Critical Cutsets of Graphs and Canonical Facets of Set-Packing Polytopes". In: *Mathematics of Operations Research* 2.1 (1977), pp. 15–19 (cit. on p. 16).

Baldacci, R., A. Mingozzi, and R. Roberti. "New Route Relaxation and Pricing Strategies for the Vehicle Routing Problem". In: *Operations Research* 59.5 (2011), pp. 1269–1283 (cit. on pp. 36, 37, 40).

Barahona, F. and A. R. Mahjoub. "Compositions of Graphs and Polyhedra II: Stable Sets". In: *SIAM Journal on Discrete Mathematics* 7.3 (1994), pp. 359–371 (cit. on p. 16).

— "Compositions of Graphs and Polyhedra III: Graphs with No W_4 Minor". In: *SIAM Journal on Discrete Mathematics* 7.3 (1994), pp. 372–389 (cit. on p. 16).

Bianco, L., A. Mingozzi, and S. Ricciardelli. "The traveling salesman problem with cumulative costs". In: *Networks* 23.2 (1993), pp. 81–91 (cit. on pp. 35, 36).

Bigras, L.-P., M. Gamache, and G. Savard. "The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times". In: *Discrete Optimization* 5.4 (2008), pp. 685–699 (cit. on p. 36).

Blum, A. et al. "The minimum latency problem". In: *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*. Montreal, Quebec, Canada, 1994, pp. 163–171 (cit. on p. 36).

Cánovas, L., M. Landete, and A. Marín. "New facets for the set packing polytope". In: *Operations Research Letters* 27.4 (2000), pp. 153 –161 (cit. on p. 16).

Caprara, A. and M. Fischetti. "{0, 1/2}-Chvátal-Gomory cuts". In: *Mathematical Programming* 74.3 (1996), pp. 221–235 (cit. on p. 19).

Chaudhuri, K. et al. "Paths, trees, and minimum latency tours". In: *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2003*. Cambridge, MA, USA, 2003, pp. 36–45 (cit. on p. 36).

Chen, D.-S., R. G. Batson, and Y. Dang. *Applied Integer Programming*. John Wiley & Sons, Inc., 2009, pp. 334–358 (cit. on p. 13).

Cheng, E. and W. H. Cunningham. "Wheel inequalities for stable set polytopes". In: *Mathematical Programming* 77.2 (1997), pp. 389–421 (cit. on p. 16).

Cheng, E. and S. de Vries. "On the Facet-Inducing Antiweb-Wheel Inequalities for Stable Set Polytopes". In: *SIAM Journal on Discrete Mathematics* 15.4 (2002), pp. 470–487 (cit. on p. 16).

Christofides, N., A. Mingozzi, and P. Toth. "Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations". In: *Mathematical Programming* 20.1 (1981), pp. 255–282 (cit. on p. 40).

Chvátal, V. "On certain polytopes associated with graphs". In: *Journal of Combinatorial Theory, Series B* 18.2 (1975), pp. 138 –154 (cit. on p. 16).

Chvátal, V. "Edmonds polytopes and a hierarchy of combinatorial problems". In: *Discrete Mathematics* 4.4 (1973), pp. 305 –337 (cit. on p. 19).

Corrêa, R. C. et al. "General cut-generating procedures for the stable set polytope". In: *Discrete Applied Mathematics* (2017), pp. – (cit. on p. 17).

Dash, S., R. Fukasawa, and O. Günlük. "On a generalization of the master cyclic group polyhedron". In: *Mathematical Programming* 125.1 (2010), pp. 1–30 (cit. on p. 17).

Desaulniers, G. "Branch-and-Price-and-Cut for the Split-Delivery Vehicle Routing Problem with Time Windows". In: *Operations Research* 58.1 (2010), pp. 179–192 (cit. on p. 13).

Dror, M. "Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW". In: *Operations Research* 42.5 (1994), pp. 977–978 (cit. on p. 39).

Dunkel, J. and A. S. Schulz. "The Gomory-Chvátal Closure of a Nonrational Polytope Is a Rational Polytope". In: *Mathematics of Operations Research* 38.1 (2013), pp. 63–91 (cit. on p. 19).

Eisenbrand, F. "NOTE – On the Membership Problem for the Elementary Closure of a Polyhedron". In: *Combinatorica* 19.2 (1999), pp. 297–300 (cit. on p. 19).

Fischetti, M., G. Laporte, and S. Martello. "The delivery man problem and cumulative matroids". In: *Operations Research* 41.6 (1993), pp. 1055–1064 (cit. on p. 36).

Fox, K. R., B. Gavish, and S. C. Graves. "Technical Note-An n-Constraint Formulation of the (Time-Dependent) Traveling Salesman Problem". In: *Operations Research* 28.4 (1980), pp. 1018–1021 (cit. on p. 36).

García, A., P. Jodrá, and J. Tejel. "A note on the traveling repairman problem". In: *Networks* 40.1 (2002), pp. 27–31 (cit. on p. 36).

Giles, R. and L. Trotter. "On stable set polyhedra for $K_{1,3}$-free graphs". In: *Journal of Combinatorial Theory, Series B* 31.3 (1981), pp. 313 –326 (cit. on p. 16).

Godinho, M. T., L. Gouveia, and P. Pesneau. "Natural and extended formulations for the Time-Dependent Traveling Salesman Problem". In: *Discrete Applied Mathematics* 164, Part 1 (2014). Combinatorial Optimization, pp. 138 –153 (cit. on p. 36).

Gomory, R. E. "An algorithm for integer solutions to linear programs". In: *Recent Advances in Mathematical Programming*. Ed. by R. Graves and P. Wolfe. McGraw-Hil, New York, 1963, pp. 269–302 (cit. on p. 19).

Irnich, S. "Resource extension functions: properties, inversion, and generalization to segments". In: *OR Spectrum* 30.1 (2008), pp. 113–148 (cit. on p. 39).

Irnich, S. and G. Desaulniers. "Shortest path problems with resource constraints". In: *Column generation* (2005), pp. 33–65 (cit. on p. 39).

Irnich, S. and D. Villeneuve. "The Shortest-Path Problem with Resource Constraints and k-Cycle Elimination for k $\geq$ 3". In: *INFORMS Journal on Computing* 18.3 (2006), pp. 391–406 (cit. on p. 40).

Irnich, S. et al. "Path-Reduced Costs for Eliminating Arcs in Routing and Scheduling". In: *INFORMS Journal on Computing* 22.2 (2010), pp. 297–313 (cit. on p. 51).

Jepsen, M. et al. "Subset-Row Inequalities Applied to the Vehicle-Routing Problem with Time Windows". In: *Operations Research* 56.2 (2008), pp. 497–511 (cit. on p. 18).

Kullmann, O. "Fundaments of branching heuristics". In: *Handbook of Satisfiability*. Ed. by A. Biere et al. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 205–244 (cit. on p. 52).

Letchford, A. N., S. Pokutta, and A. S. Schulz. "On the membership problem for the {0, 1/2}-closure". In: *Operations Research Letters* 39.5 (2011), pp. 301 –304 (cit. on p. 19).

Lucena, A. "Time-dependent traveling salesman problem - The deliveryman case". In: *Networks* 20.6 (1990), pp. 753–763 (cit. on p. 36).

Lysgaard, J. and S. Wøhlk. "A branch-and-cut-and-price algorithm for the cumulative capacitated vehicle routing problem". In: *European Journal of Operational Research* 236.3 (2014), pp. 800 –810 (cit. on p. 35).

Mahjoub, A. R. "On the stable set polytope of a series-parallel graph". In: *Mathematical Programming* 40.1 (1988), pp. 53–57 (cit. on p. 16).

Martinelli, R., D. Pecin, and M. Poggi. "Efficient elementary and restricted non-elementary route pricing". In: *European Journal of Operational Research* 239.1 (2014), pp. 102 –111 (cit. on pp. 40, 46, 47, 53).

Méndez-Díaz, I., P. Zabala, and A. Lucena. "A new formulation for the traveling deliveryman problem". In: *Discrete Applied Mathematics* 156.17 (2008), pp. 3223–3237 (cit. on pp. 35, 36).

Mladenović, N., D. Urošević, and S. Hanafi. "Variable neighborhood search for the travelling deliveryman problem". In: *4OR* 11.1 (2013), pp. 57–73 (cit. on p. 37).

Nemhauser, G. L. and L. E. Trotter. "Properties of vertex packing and independence system polyhedra". In: *Mathematical Programming* 6.1 (1974), pp. 48–61 (cit. on p. 16).

Ngueveu, S., C. Prins, and R. Wolfler Calvo. "An effective memetic algorithm for the cumulative capacitated vehicle routing problem". In: *Computers & Operations Research* 37.11 (2010), pp. 1877–1885 (cit. on p. 37).

Nucamendi-Guillén, S. et al. "A mixed integer formulation and an efficient metaheuristic procedure for the k-Travelling Repairmen Problem". In: *Journal of the Operational Research Society* 67.8 (2016), pp. 1121–1134 (cit. on p. 35).

Padberg, M. W. "On the Complexity of Set Packing Polyhedra". In: *Annals of Discrete Mathematics* 1 (1977), pp. 421 –434 (cit. on p. 16).

— "On the facial structure of set packing polyhedra". In: *Mathematical Programming* 5.1 (1973), pp. 199–215 (cit. on p. 16).

Pecin, D. et al. "Improved branch-cut-and-price for capacitated vehicle routing". In: *Mathematical Programming Computation* 9.1 (2017), pp. 61–100 (cit. on pp. 13, 18, 20, 51–53).

Pecin, D. et al. "Limited memory Rank-1 Cuts for Vehicle Routing Problems". In: *Operations Research Letters* 45.3 (2017), pp. 206 –209 (cit. on pp. 17–19, 26).

Pecin, D. et al. "New Enhancements for the Exact Solution of the Vehicle Routing Problem with Time Windows". In: *INFORMS Journal on Computing* 29.3 (2017), pp. 489–502 (cit. on pp. 13, 18, 20, 26, 44).

Pessoa, A. et al. "Automation and combination of linear-programming based stabilization techniques in column generation". In: *INFORMS Journal on Computing* Forthcoming (2017) (cit. on p. 50).

Pessoa, A. et al. "Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems". In: *Mathematical Programming Computation* 2.3 (2010), pp. 259–290 (cit. on p. 13).

Picard, J.-C. and M. Queyranne. "The Time-Dependent Traveling Salesman Problem and Its Application to the Tardiness Problem in One-Machine Scheduling". In: *Operations Research* 26.1 (1978), pp. 86–110 (cit. on p. 36).

Poggi de Aragão, M. and E. Uchoa. "Integer program reformulation for robust branch-and-cut-and-price". In: *Annals of Mathematical Programming in Rio*. Ed. by L. Wolsey. Búzios, Brazil, 2003, pp. 56–61 (cit. on p. 19).

Rebennack, S. et al. "A Branch and Cut solver for the maximum stable set problem". In: *Journal of Combinatorial Optimization* 21.4 (2011), pp. 434–457 (cit. on p. 16).

Righini, G. and M. Salani. "New Dynamic Programming Algorithms for the Resource Constrained Elementary Shortest Path Problem". In: *Networks* 51.3 (2008), pp. 155–170 (cit. on pp. 46, 47).

— "Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints". In: *Discrete Optimization* 3.3 (2006), pp. 255 –273 (cit. on p. 54).

Rivera, J. C., H. M. Afsar, and C. Prins. "Mathematical formulations and exact algorithm for the multitrip cumulative capacitated single-vehicle routing problem". In: *European Journal of Operational Research* 249.1 (2016), pp. 93 –104 (cit. on p. 35).

Roberti, R. and A. Mingozzi. "Dynamic ng-Path Relaxation for the Delivery Man Problem". In: *Transportation Science* 48.3 (2014), pp. 413–424 (cit. on pp. 13, 35–38, 46, 47, 51, 52, 56–58, 60, 63, 65, 67).

Røpke, S. "Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems". In: *Presentation in Column Generation 2012* (2012) (cit. on p. 52).

Rossi, F and S Smriglio. "A branch-and-cut algorithm for the maximum cardinality stable set problem". In: *Operations Research Letters* 28.2 (2001), pp. 63 –74 (cit. on p. 16).

S. Xavier, Álinson and M. Campêlo. "A New Facet Generating Procedure for the Stable Set Polytope". In: *Electronic Notes in Discrete Mathematics* 37 (2011), pp. 183 –188 (cit. on p. 17).

Sahni, S. and T. Gonzalez. "P-complete approximation problems". In: *Journal of The ACM* 23.3 (3 1976), pp. 555–565 (cit. on p. 35).

Salehipour, A. et al. "Efficient GRASP+VND and GRASP+VNS metaheuristics for the traveling repairman problem". In: *4OR* 9.2 (2011), pp. 189–209 (cit. on p. 37).

Sherali, H. D. and Y. Lee. "Tighter representations for set partitioning problems". In: *Discrete Applied Mathematics* 68.1 (1996), pp. 153 –167 (cit. on p. 17).

Silva, M. M. et al. "A simple and effective metaheuristic for the Minimum Latency Problem". In: *European Journal of Operational Research* 221.3 (2012), pp. 513 – 520 (cit. on pp. 35, 37, 57, 59).

Sitters, R. "The minimum latency problem is NP-hard for weighted trees". In: *Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization, IPCO 2002*. Cambridge, MA, USA, 2002, pp. 230–239 (cit. on pp. 35, 36).

Sze, J. F., S. Salhi, and N. Wassan. "The cumulative capacitated vehicle routing problem with min-sum and min-max objectives: An effective hybridisation of adaptive variable neighbourhood search and large neighbourhood search". In: *Transportation Research Part B: Methodological* 101 (2017), pp. 162 –184 (cit. on p. 35).

Toth, P. et al. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2014 (cit. on pp. 13, 14).

Trotter, L. "A class of facet producing graphs for vertex packing polyhedra". In: *Discrete Mathematics* 12.4 (1975), pp. 373 –388 (cit. on p. 16).

Tsitsiklis, J. N. "Special cases of traveling salesman and repairman problems with time windows". In: *Networks* 22.3 (1992), pp. 263–282 (cit. on p. 35).

Van Eijl, C. A. *A polyhedral approach to the delivery man problem*. Tech. rep. COSOR 95-19. Eindhoven University of Technology, 1995 (cit. on p. 36).

Vanderbeck, F., R. Sadykov, and I. Tahiri. *BaPCod — a generic Branch-And-Price Code*. Tech. rep. 2017 (cit. on p. 56).

Wolsey, L. A. "Further facet generating procedures for vertex packing polytopes". In: *Mathematical Programming* 11.1 (1976), pp. 158–163 (cit. on p. 16).

APPENDIX A – MLP Solutions

Optimal solution of dantzig42

depot → 1 → 41 → 40 → 39 → 38 → 37 → 36 → 34 → 33 → 32 → 31 → 30 → 29 → 28 → 27 → 25 → 26 → 23 → 24 → 7 → 6 → 5 → 4 → 3 → 2 → 8 → 9 → 11 → 10 → 22 → 21 → 20 → 19 → 18 → 17 → 15 → 14 → 13 → 12 → 16 → 35 → depot

Optimal solution of swiss42

depot → 1 → 6 → 4 → 3 → 27 → 2 → 28 → 30 → 29 → 8 → 9 → 23 → 41 → 10 → 25 → 11 → 12 → 18 → 26 → 5 → 13 → 19 → 14 → 16 → 15 → 37 → 7 → 17 → 31 → 36 → 35 → 20 → 33 → 34 → 32 → 38 → 22 → 39 → 21 → 40 → 24 → depot

Optimal solution of att48

depot → 8 → 37 → 30 → 43 → 17 → 6 → 27 → 5 → 36 → 18 → 26 → 16 → 42 → 29 → 35 → 45 → 32 → 19 → 46 → 20 → 12 → 24 → 13 → 22 → 10 → 11 → 14 → 39 → 21 → 15 → 2 → 33 → 28 → 4 → 47 → 38 → 31 → 23 → 9 → 44 → 34 → 3 → 25 → 41 → 1 → 40 → 7 → depot

Optimal solution of gr48

depot → 12 → 15 → 10 → 47 → 28 → 6 → 27 → 45 → 17 → 33 → 22 → 24 → 2 → 18 → 3 → 29 → 37 → 19 → 34 → 1 → 44 → 42 → 46 → 36 → 23 → 9 → 11 → 30 → 4 → 32 → 7 → 21 → 20 → 31 → 26 → 16 → 8 → 13 → 35 → 5 → 25 → 14 → 39 → 38 → 41 → 40 → 43 → depot

Optimal solution of hk48

depot → 1 → 14 → 47 → 25 → 28 → 31 → 24 → 27 → 12 → 21 → 11 → 17 → 13 → 37 → 6 → 46 → 7 → 5 → 33 → 35 → 39 → 9 → 8 → 22 → 2 → 4 → 40 → 23 → 34 → 16 → 30 → 19 → 10 → 15 → 41 → 3 → 45 → 44 → 38 → 43 → 26 → 36 → 18 → 42 → 20 → 32 → 29 → depot

Optimal solution of eil51

depot → 21 → 1 → 15 → 49 → 8 → 48 → 4 → 37 → 10 → 31 → 26 → 50 → 45 → 11 → 46 → 17 → 3 → 16 → 36 → 43 → 14 → 44 → 32 → 9 → 29 → 33 → 20 → 28 → 19

→ 34 → 35 → 2 → 27 → 30 → 25 → 7 → 47 → 5 → 22 → 6 → 42 → 23 → 13 → 24 → 12 → 40 → 18 → 39 → 41 → 38 → depot

Optimal solution of berlin52

depot → 21 → 31 → 48 → 35 → 34 → 33 → 38 → 39 → 36 → 37 → 47 → 23 → 4 → 14 → 5 → 3 → 24 → 45 → 43 → 15 → 49 → 19 → 22 → 29 → 1 → 6 → 41 → 20 → 30 → 17 → 2 → 44 → 18 → 40 → 7 → 8 → 9 → 42 → 32 → 50 → 11 → 27 → 26 → 25 → 46 → 12 → 13 → 51 → 10 → 28 → 16 → depot

Optimal solution of brazil58

depot → 17 → 43 → 57 → 23 → 56 → 11 → 26 → 42 → 48 → 46 → 50 → 51 → 9 → 34 → 40 → 1 → 53 → 54 → 47 → 2 → 28 → 35 → 16 → 25 → 18 → 5 → 27 → 13 → 32 → 44 → 55 → 45 → 33 → 14 → 36 → 20 → 38 → 10 → 21 → 7 → 4 → 22 → 29 → 12 → 39 → 24 → 8 → 31 → 19 → 52 → 49 → 3 → 15 → 37 → 41 → 30 → 6 → depot

Optimal solution of st70

depot → 35 → 28 → 12 → 69 → 30 → 68 → 58 → 62 → 65 → 21 → 37 → 22 → 15 → 46 → 36 → 57 → 49 → 9 → 51 → 59 → 11 → 33 → 20 → 16 → 42 → 40 → 5 → 41 → 17 → 3 → 1 → 6 → 31 → 2 → 7 → 27 → 13 → 19 → 29 → 43 → 67 → 8 → 26 → 45 → 39 → 60 → 44 → 24 → 38 → 61 → 53 → 32 → 66 → 47 → 10 → 55 → 64 → 50 → 4 → 52 → 56 → 14 → 23 → 18 → 25 → 48 → 54 → 34 → 63 → depot

Optimal solution of eil76

depot → 72 → 61 → 27 → 73 → 1 → 29 → 47 → 28 → 44 → 26 → 51 → 33 → 45 → 7 → 34 → 6 → 66 → 25 → 75 → 74 → 3 → 67 → 5 → 50 → 16 → 11 → 39 → 2 → 43 → 31 → 8 → 38 → 71 → 57 → 9 → 37 → 64 → 65 → 10 → 52 → 13 → 18 → 53 → 12 → 56 → 14 → 4 → 36 → 19 → 69 → 59 → 70 → 68 → 35 → 46 → 20 → 60 → 21 → 63 → 41 → 42 → 40 → 55 → 22 → 62 → 32 → 15 → 48 → 23 → 17 → 49 → 24 → 54 → 30 → 58 → depot

Optimal solution of pr76

depot → 1 → 2 → 3 → 4 → 19 → 18 → 30 → 29 → 28 → 25 → 26 → 27 → 42 → 41 → 53 → 52 → 51 → 48 → 49 → 50 → 65 → 64 → 55 → 54 → 57 → 56 → 62 → 63 → 61 → 60 → 58 → 59 → 40 → 39 → 33 → 32 → 31 → 34 → 38 → 37 → 35 → 36 → 17 → 16 → 15 → 14 → 13 → 12 → 11 → 10 → 9 → 8 → 5 → 6 → 7 → 74 → 75 → 22 → 21 → 20 → 24 → 23 → 45 → 44 → 43 → 47 → 46 → 68 → 67 → 66 → 69 → 70 → 71 → 72 → 73 → depot

Optimal solution of gr96

depot → 29 → 30 → 31 → 35 → 36 → 37 → 38 → 39 → 40 → 41 → 42 → 48 → 49 → 52 → 51 → 50 → 54 → 55 → 56 → 58 → 70 → 71 → 72 → 74 → 73 → 83 → 84 → 85 → 86 → 89 → 88 → 87 → 77 → 76 → 75 → 67 → 66 → 65 → 63 → 62 → 61 → 60 → 59 → 24 → 23 → 22 → 25 → 21 → 20 → 18 → 17 → 19 → 16 → 15 → 14 → 13 → 12 → 11 → 9 → 8 → 7 → 6 → 5 → 4 → 3 → 2 → 1 → 28 → 34 → 33 → 32 → 43 → 44 → 45 → 46 → 47 → 53 → 57 → 68 → 69 → 81 → 80 → 82 → 90 → 91 → 92 → 94 → 93 → 95 → 64 → 27 → 26 → 10 → 78 → 79 → depot

Optimal solution of rat99

depot → 1 → 9 → 10 → 11 → 2 → 3 → 13 → 4 → 5 → 6 → 7 → 8 → 17 → 16 → 15 → 14 → 22 → 23 → 24 → 25 → 34 → 35 → 43 → 33 → 42 → 41 → 40 → 31 → 30 → 29 → 28 → 36 → 37 → 47 → 38 → 39 → 49 → 50 → 51 → 52 → 53 → 62 → 61 → 60 → 59 → 48 → 58 → 57 → 56 → 55 → 64 → 65 → 66 → 67 → 68 → 69 → 70 → 71 → 80 → 79 → 78 → 77 → 76 → 75 → 83 → 84 → 85 → 86 → 87 → 88 → 89 → 98 → 97 → 96 → 95 → 94 → 93 → 92 → 91 → 90 → 82 → 81 → 72 → 74 → 73 → 63 → 54 → 46 → 45 → 27 → 18 → 19 → 12 → 21 → 20 → 32 → 44 → 26 → depot

Optimal solution of kroA100

depot → 62 → 5 → 48 → 89 → 78 → 52 → 87 → 15 → 21 → 93 → 17 → 23 → 37 → 35 → 83 → 9 → 71 → 20 → 73 → 58 → 16 → 14 → 10 → 31 → 44 → 90 → 97 → 22 → 76 → 59 → 61 → 34 → 85 → 26 → 11 → 19 → 56 → 8 → 6 → 54 → 82 → 33 → 28 → 45 → 42 → 2 → 13 → 70 → 40 → 99 → 47 → 29 → 38 → 95 → 77 → 51 → 4 → 36 → 32 → 75 → 12 → 94 → 81 → 49 → 43 → 1 → 53 → 39 → 63 → 68 → 72 → 67 → 84 → 80 → 24 → 86 → 50 → 60 → 57 → 66 → 27 → 92 → 91 → 7 → 41 → 88 → 30 → 79 → 55 → 96 → 74 → 18 → 3 → 64 → 25 → 65 → 69 → 98 → 46 → depot

Optimal solution of kroB100

depot → 94 → 89 → 20 → 9 → 67 → 85 → 48 → 37 → 19 → 79 → 29 → 74 → 68 → 25 → 99 → 55 → 61 → 4 → 66 → 39 → 38 → 69 → 52 → 72 → 84 → 92 → 10 → 2 → 27 → 90 → 96 → 98 → 7 → 28 → 75 → 58 → 31 → 97 → 11 → 70 → 45 → 24 → 8 → 26 → 60 → 34 → 93 → 56 → 6 → 83 → 57 → 51 → 53 → 87 → 22 → 21 → 54 → 76 → 23 → 17 → 44 → 35 → 95 → 18 → 91 → 40 → 16 → 77 → 12 → 62 → 30 → 47 → 81 → 32 → 14 → 5 → 3 → 82 → 63 → 13 → 41 → 1 → 15 → 49 → 42 → 88 → 86 → 65 → 73 → 59 → 33 → 71 → 36 → 64 → 78 → 80 → 46 → 43 → 50 → depot

Optimal solution of kroC100

depot → 52 → 84 → 14 → 12 → 78 → 63 → 19 → 41 → 54 → 66 → 30 → 46 → 5 → 53 → 74 → 21 → 7 → 16 → 24 → 89 → 33 → 57 → 97 → 87 → 27 → 38 → 37 → 70

→ 71 → 82 → 61 → 49 → 94 → 93 → 90 → 75 → 69 → 22 → 34 → 1 → 67 → 29 → 88 → 40 → 58 → 72 → 2 → 68 → 13 → 98 → 18 → 91 → 9 → 35 → 56 → 73 → 99 → 32 → 44 → 80 → 96 → 95 → 86 → 51 → 10 → 83 → 47 → 65 → 43 → 62 → 50 → 15 → 36 → 8 → 77 → 81 → 6 → 25 → 60 → 31 → 23 → 45 → 28 → 17 → 48 → 92 → 3 → 59 → 11 → 39 → 26 → 64 → 79 → 76 → 20 → 85 → 4 → 42 → 55 → depot

Optimal solution of kroD100

depot → 49 → 33 → 80 → 37 → 65 → 7 → 51 → 45 → 75 → 79 → 77 → 30 → 98 → 44 → 32 → 67 → 13 → 64 → 85 → 95 → 15 → 62 → 53 → 90 → 76 → 42 → 63 → 84 → 58 → 2 → 82 → 39 → 5 → 17 → 24 → 19 → 9 → 8 → 54 → 25 → 86 → 91 → 47 → 1 → 99 → 29 → 34 → 50 → 18 → 36 → 61 → 4 → 52 → 59 → 43 → 92 → 3 → 70 → 38 → 72 → 14 → 83 → 40 → 23 → 26 → 87 → 78 → 12 → 93 → 31 → 96 → 97 → 48 → 69 → 88 → 89 → 16 → 10 → 22 → 41 → 21 → 57 → 28 → 35 → 66 → 74 → 6 → 73 → 60 → 71 → 56 → 81 → 46 → 11 → 27 → 55 → 94 → 20 → 68 → depot

Optimal solution of kroE100

depot → 73 → 20 → 6 → 75 → 11 → 31 → 57 → 62 → 87 → 34 → 39 → 24 → 53 → 44 → 98 → 16 → 83 → 48 → 61 → 55 → 19 → 14 → 22 → 63 → 94 → 35 → 58 → 97 → 50 → 68 → 46 → 2 → 45 → 25 → 78 → 15 → 72 → 60 → 70 → 80 → 88 → 13 → 51 → 43 → 71 → 67 → 37 → 32 → 21 → 38 → 42 → 85 → 18 → 93 → 17 → 23 → 28 → 99 → 77 → 52 → 33 → 36 → 76 → 84 → 95 → 10 → 1 → 3 → 81 → 66 → 26 → 86 → 40 → 74 → 91 → 69 → 5 → 54 → 90 → 92 → 29 → 89 → 47 → 41 → 4 → 7 → 64 → 96 → 9 → 79 → 56 → 30 → 65 → 12 → 8 → 59 → 49 → 27 → 82 → depot

Optimal solution of rd100

depot → 59 → 7 → 68 → 14 → 62 → 85 → 96 → 66 → 12 → 48 → 20 → 74 → 81 → 84 → 13 → 11 → 3 → 31 → 8 → 25 → 73 → 19 → 77 → 32 → 9 → 26 → 91 → 55 → 45 → 49 → 72 → 16 → 71 → 69 → 37 → 53 → 18 → 36 → 27 → 92 → 76 → 94 → 58 → 75 → 57 → 87 → 83 → 30 → 65 → 97 → 93 → 5 → 78 → 52 → 98 → 46 → 56 → 35 → 10 → 60 → 4 → 80 → 79 → 64 → 89 → 50 → 82 → 47 → 29 → 21 → 40 → 33 → 6 → 41 → 23 → 24 → 42 → 39 → 54 → 95 → 38 → 99 → 43 → 28 → 34 → 22 → 1 → 88 → 15 → 44 → 51 → 63 → 90 → 67 → 70 → 17 → 61 → 86 → 2 → depot

Optimal solution of eil101

depot → 68 → 26 → 100 → 52 → 57 → 12 → 93 → 5 → 95 → 98 → 58 → 92 → 84 → 60 → 15 → 90 → 99 → 97 → 36 → 91 → 94 → 96 → 86 → 1 → 56 → 40 → 21 → 74 → 73 → 71 → 72 → 20 → 39 → 25 → 11 → 79 → 67 → 2 → 76 → 75 → 49 → 32 → 80 → 8 → 50 → 29 → 69 → 30 → 87 → 6 → 51 → 88 → 17 → 82 → 59 → 4 → 83 →

16 → 44 → 7 → 81 → 47 → 46 → 18 → 10 → 61 → 9 → 62 → 89 → 31 → 19 → 65 → 70 → 34 → 33 → 77 → 78 → 28 → 23 → 53 → 54 → 24 → 3 → 55 → 38 → 66 → 22 → 14 → 42 → 41 → 13 → 43 → 37 → 85 → 45 → 35 → 48 → 63 → 64 → 27 → depot

Optimal solution of lin105

depot → 1 → 5 → 6 → 9 → 10 → 14 → 102 → 20 → 21 → 28 → 31 → 30 → 29 → 27 → 22 → 19 → 26 → 23 → 18 → 15 → 16 → 17 → 24 → 25 → 35 → 36 → 41 → 40 → 42 → 45 → 51 → 52 → 57 → 56 → 53 → 50 → 46 → 43 → 44 → 39 → 48 → 47 → 49 → 54 → 55 → 58 → 104 → 61 → 62 → 69 → 68 → 73 → 74 → 80 → 72 → 75 → 79 → 85 → 78 → 76 → 71 → 66 → 67 → 70 → 77 → 81 → 82 → 83 → 84 → 90 → 91 → 95 → 96 → 100 → 101 → 92 → 88 → 89 → 97 → 98 → 99 → 94 → 93 → 87 → 86 → 65 → 64 → 60 → 59 → 38 → 37 → 34 → 33 → 13 → 12 → 3 → 4 → 8 → 7 → 2 → 11 → 32 → 103 → 63 → depot

Optimal solution of pr107

depot → 3 → 2 → 5 → 1 → 4 → 6 → 8 → 9 → 7 → 10 → 11 → 12 → 15 → 14 → 17 → 13 → 16 → 19 → 22 → 23 → 20 → 18 → 21 → 24 → 27 → 26 → 29 → 25 → 28 → 31 → 34 → 35 → 32 → 30 → 33 → 36 → 39 → 38 → 41 → 37 → 40 → 43 → 46 → 47 → 42 → 44 → 45 → 48 → 51 → 50 → 53 → 52 → 49 → 54 → 105 → 106 → 104 → 103 → 55 → 100 → 98 → 101 → 102 → 99 → 97 → 94 → 96 → 95 → 56 → 93 → 57 → 90 → 88 → 91 → 92 → 89 → 87 → 84 → 86 → 85 → 58 → 83 → 59 → 80 → 78 → 81 → 82 → 79 → 77 → 74 → 76 → 75 → 60 → 73 → 61 → 72 → 69 → 70 → 71 → 68 → 67 → 64 → 66 → 65 → 63 → 62 → depot

Optimal solution of gr120

depot → 75 → 28 → 29 → 31 → 91 → 27 → 119 → 60 → 15 → 59 → 7 → 69 → 115 → 33 → 3 → 25 → 70 → 46 → 54 → 88 → 47 → 101 → 100 → 109 → 111 → 35 → 103 → 98 → 61 → 36 → 66 → 56 → 72 → 113 → 79 → 26 → 4 → 62 → 76 → 63 → 108 → 20 → 92 → 1 → 114 → 10 → 22 → 8 → 81 → 2 → 118 → 102 → 37 → 6 → 55 → 40 → 41 → 97 → 16 → 117 → 48 → 49 → 45 → 19 → 106 → 112 → 68 → 64 → 42 → 67 → 78 → 57 → 99 → 32 → 51 → 107 → 24 → 18 → 17 → 84 → 116 → 30 → 65 → 21 → 85 → 93 → 80 → 58 → 14 → 77 → 44 → 13 → 74 → 43 → 86 → 73 → 104 → 71 → 39 → 23 → 110 → 95 → 53 → 89 → 5 → 83 → 34 → 9 → 105 → 82 → 38 → 94 → 11 → 96 → 87 → 52 → 50 → 12 → 90 → depot

Optimal solution of pr124

depot → 7 → 1 → 2 → 3 → 5 → 31 → 30 → 29 → 33 → 32 → 61 → 65 → 64 → 62 → 63 → 67 → 68 → 69 → 70 → 71 → 72 → 73 → 74 → 75 → 76 → 77 → 78 → 79 → 91

→ 92 → 93 → 94 → 95 → 96 → 97 → 98 → 99 → 123 → 122 → 121 → 120 → 119 → 118 → 117 → 116 → 115 → 114 → 83 → 81 → 80 → 82 → 66 → 59 → 60 → 58 → 35 → 34 → 28 → 27 → 26 → 6 → 25 → 36 → 37 → 23 → 24 → 21 → 22 → 38 → 39 → 20 → 19 → 40 → 41 → 42 → 43 → 44 → 18 → 17 → 16 → 15 → 8 → 9 → 10 → 11 → 12 → 13 → 14 → 50 → 49 → 48 → 47 → 46 → 45 → 51 → 52 → 87 → 88 → 103 → 104 → 105 → 106 → 107 → 108 → 109 → 110 → 111 → 112 → 113 → 102 → 101 → 100 → 90 → 89 → 84 → 85 → 86 → 53 → 54 → 55 → 56 → 57 → 4 → depot

Optimal solution of bier127

depot → 6 → 104 → 113 → 5 → 105 → 14 → 107 → 19 → 16 → 20 → 21 → 3 → 18 → 71 → 7 → 22 → 23 → 8 → 10 → 2 → 89 → 115 → 59 → 58 → 61 → 60 → 90 → 57 → 63 → 99 → 9 → 119 → 12 → 114 → 49 → 120 → 55 → 4 → 51 → 123 → 46 → 52 → 48 → 117 → 47 → 44 → 102 → 43 → 53 → 56 → 50 → 1 → 15 → 13 → 11 → 30 → 26 → 29 → 40 → 35 → 36 → 34 → 39 → 42 → 33 → 41 → 38 → 37 → 25 → 24 → 32 → 121 → 27 → 28 → 31 → 79 → 78 → 76 → 17 → 73 → 72 → 66 → 67 → 70 → 69 → 68 → 74 → 75 → 77 → 116 → 83 → 80 → 125 → 82 → 81 → 62 → 118 → 95 → 108 → 87 → 86 → 85 → 84 → 109 → 103 → 124 → 88 → 91 → 98 → 64 → 112 → 65 → 54 → 45 → 93 → 111 → 110 → 106 → 126 → 92 → 94 → 122 → 96 → 97 → 100 → 101 → depot

Optimal solution of ch130

depot → 40 → 38 → 70 → 129 → 49 → 1 → 117 → 79 → 45 → 19 → 92 → 36 → 21 → 46 → 39 → 22 → 32 → 20 → 17 → 7 → 107 → 113 → 2 → 82 → 116 → 111 → 61 → 104 → 127 → 15 → 44 → 75 → 108 → 60 → 128 → 123 → 25 → 96 → 69 → 106 → 126 → 103 → 42 → 33 → 16 → 30 → 26 → 18 → 99 → 14 → 28 → 23 → 115 → 94 → 78 → 86 → 11 → 80 → 102 → 76 → 93 → 88 → 109 → 97 → 67 → 47 → 24 → 112 → 31 → 35 → 83 → 118 → 110 → 122 → 100 → 81 → 56 → 8 → 55 → 64 → 51 → 74 → 73 → 98 → 72 → 91 → 37 → 105 → 57 → 48 → 52 → 119 → 59 → 50 → 41 → 43 → 54 → 121 → 13 → 9 → 95 → 66 → 12 → 77 → 120 → 125 → 29 → 58 → 89 → 124 → 84 → 65 → 27 → 114 → 4 → 10 → 63 → 68 → 85 → 87 → 6 → 62 → 53 → 34 → 3 → 71 → 90 → 5 → 101 → depot

Optimal solution of pr136

depot → 6 → 7 → 1 → 15 → 19 → 31 → 26 → 27 → 18 → 16 → 5 → 4 → 29 → 28 → 30 → 37 → 40 → 41 → 36 → 49 → 53 → 65 → 60 → 61 → 52 → 50 → 39 → 38 → 63 → 62 → 64 → 71 → 73 → 72 → 97 → 96 → 98 → 85 → 83 → 74 → 75 → 70 → 82 → 86 → 99 → 104 → 109 → 108 → 119 → 117 → 105 → 107 → 106 → 131 → 130 → 135 → 129 → 128 → 134 → 120 → 116 → 121 → 127 → 110 → 111 → 115 → 122 → 126 →

133 → 125 → 124 → 132 → 123 → 114 → 102 → 113 → 112 → 103 → 100 → 91 → 90 → 101 → 89 → 80 → 68 → 79 → 78 → 81 → 88 → 92 → 93 → 76 → 77 → 69 → 66 → 57 → 56 → 67 → 55 → 46 → 34 → 45 → 44 → 47 → 54 → 58 → 59 → 42 → 43 → 35 → 32 → 24 → 20 → 13 → 23 → 22 → 33 → 21 → 12 → 3 → 11 → 10 → 2 → 9 → 8 → 25 → 14 → 48 → 87 → 94 → 95 → 118 → 84 → 51 → 17 → depot

Optimal solution of gr137

depot → 1 → 2 → 3 → 4 → 17 → 18 → 6 → 5 → 8 → 7 → 9 → 28 → 29 → 35 → 36 → 40 → 11 → 12 → 13 → 14 → 44 → 45 → 46 → 47 → 41 → 42 → 37 → 31 → 30 → 25 → 23 → 19 → 20 → 21 → 22 → 24 → 26 → 27 → 51 → 52 → 53 → 54 → 55 → 56 → 60 → 57 → 58 → 59 → 61 → 62 → 63 → 64 → 65 → 66 → 67 → 68 → 69 → 70 → 88 → 89 → 90 → 92 → 93 → 94 → 96 → 97 → 98 → 99 → 100 → 101 → 102 → 103 → 115 → 113 → 112 → 111 → 110 → 117 → 118 → 119 → 120 → 121 → 122 → 123 → 126 → 127 → 128 → 129 → 130 → 131 → 132 → 133 → 134 → 82 → 83 → 84 → 80 → 79 → 78 → 77 → 76 → 75 → 73 → 50 → 49 → 48 → 43 → 38 → 32 → 33 → 34 → 39 → 71 → 72 → 74 → 87 → 86 → 81 → 85 → 135 → 136 → 125 → 124 → 116 → 109 → 108 → 107 → 106 → 105 → 104 → 114 → 95 → 91 → 15 → 16 → 10 → depot

Optimal solution of pr144

depot → 2 → 3 → 4 → 5 → 1 → 27 → 28 → 29 → 26 → 25 → 34 → 35 → 24 → 23 → 21 → 22 → 36 → 37 → 38 → 39 → 7 → 6 → 8 → 9 → 10 → 11 → 18 → 19 → 20 → 17 → 16 → 40 → 41 → 15 → 14 → 42 → 43 → 44 → 70 → 71 → 69 → 68 → 67 → 66 → 65 → 64 → 63 → 62 → 61 → 60 → 45 → 46 → 47 → 58 → 59 → 57 → 56 → 55 → 54 → 53 → 52 → 51 → 50 → 49 → 48 → 78 → 79 → 80 → 81 → 82 → 83 → 84 → 85 → 109 → 108 → 107 → 106 → 105 → 104 → 103 → 86 → 87 → 102 → 101 → 100 → 99 → 98 → 95 → 94 → 92 → 93 → 96 → 97 → 89 → 88 → 77 → 76 → 75 → 74 → 73 → 72 → 90 → 91 → 115 → 116 → 117 → 118 → 119 → 120 → 121 → 122 → 123 → 124 → 125 → 126 → 127 → 128 → 129 → 133 → 132 → 131 → 130 → 114 → 134 → 135 → 136 → 137 → 138 → 139 → 140 → 141 → 142 → 143 → 110 → 111 → 112 → 113 → 30 → 31 → 32 → 33 → 13 → 12 → depot

Optimal solution of ch150

depot → 97 → 102 → 33 → 86 → 75 → 72 → 47 → 62 → 29 → 83 → 6 → 7 → 88 → 95 → 34 → 92 → 125 → 32 → 104 → 110 → 15 → 132 → 14 → 77 → 58 → 78 → 120 → 87 → 93 → 9 → 112 → 2 → 61 → 148 → 124 → 21 → 103 → 3 → 149 → 114 → 43 → 70 → 44 → 127 → 67 → 118 → 90 → 105 → 12 → 73 → 122 → 30 → 26 → 128 → 143 → 146 → 48 → 71 → 79 → 13 → 121 → 76 → 130 → 31 → 22 → 37 → 66 → 42 →

108 → 50 → 19 → 24 → 109 → 80 → 28 → 85 → 134 → 69 → 107 → 101 → 113 → 98 → 18 → 1 → 36 → 5 → 27 → 8 → 41 → 119 → 46 → 138 → 39 → 52 → 11 → 23 → 117 → 126 → 68 → 35 → 60 → 10 → 147 → 129 → 16 → 65 → 59 → 38 → 56 → 40 → 100 → 115 → 133 → 137 → 53 → 91 → 45 → 89 → 55 → 82 → 140 → 57 → 54 → 49 → 136 → 131 → 64 → 84 → 141 → 17 → 74 → 25 → 145 → 96 → 142 → 99 → 4 → 106 → 94 → 81 → 123 → 51 → 20 → 63 → 111 → 135 → 144 → 116 → 139 → depot

Optimal solution of kroA150

depot → 129 → 62 → 5 → 48 → 89 → 105 → 78 → 136 → 133 → 52 → 87 → 15 → 21 → 93 → 17 → 23 → 37 → 103 → 110 → 101 → 98 → 35 → 126 → 58 → 140 → 73 → 20 → 16 → 14 → 10 → 31 → 108 → 90 → 97 → 22 → 109 → 76 → 59 → 61 → 149 → 34 → 85 → 26 → 11 → 19 → 56 → 6 → 116 → 8 → 144 → 86 → 124 → 50 → 60 → 24 → 80 → 139 → 134 → 33 → 82 → 54 → 148 → 119 → 114 → 122 → 42 → 135 → 40 → 70 → 99 → 13 → 2 → 45 → 28 → 131 → 111 → 106 → 29 → 120 → 100 → 38 → 95 → 77 → 51 → 4 → 36 → 102 → 145 → 32 → 75 → 12 → 94 → 125 → 81 → 115 → 49 → 43 → 113 → 143 → 1 → 53 → 39 → 63 → 68 → 107 → 66 → 104 → 141 → 147 → 132 → 137 → 88 → 30 → 79 → 121 → 41 → 7 → 91 → 138 → 55 → 142 → 118 → 117 → 123 → 25 → 128 → 65 → 64 → 3 → 96 → 74 → 18 → 9 → 83 → 71 → 112 → 46 → 130 → 92 → 27 → 57 → 72 → 67 → 84 → 146 → 47 → 127 → 44 → 69 → depot

Optimal solution of kroB150

depot → 52 → 84 → 14 → 12 → 78 → 109 → 19 → 63 → 41 → 54 → 66 → 30 → 46 → 103 → 148 → 5 → 53 → 133 → 74 → 21 → 111 → 7 → 105 → 16 → 24 → 89 → 33 → 144 → 108 → 97 → 147 → 87 → 27 → 38 → 37 → 100 → 55 → 118 → 70 → 123 → 71 → 82 → 61 → 49 → 132 → 94 → 93 → 122 → 90 → 75 → 130 → 138 → 120 → 110 → 69 → 22 → 101 → 20 → 121 → 88 → 40 → 58 → 117 → 72 → 2 → 68 → 13 → 140 → 98 → 104 → 18 → 91 → 9 → 35 → 56 → 73 → 99 → 32 → 44 → 80 → 96 → 143 → 95 → 149 → 51 → 10 → 83 → 127 → 102 → 47 → 65 → 43 → 62 → 50 → 126 → 145 → 134 → 15 → 119 → 36 → 8 → 81 → 77 → 25 → 60 → 131 → 31 → 116 → 23 → 136 → 124 → 45 → 141 → 28 → 17 → 48 → 92 → 142 → 3 → 115 → 11 → 39 → 26 → 106 → 129 → 64 → 146 → 79 → 114 → 76 → 29 → 67 → 34 → 1 → 137 → 112 → 107 → 113 → 57 → 125 → 85 → 4 → 42 → 128 → 139 → 59 → 135 → 86 → 6 → depot

Optimal solution of pr152

depot → 34 → 35 → 33 → 15 → 36 → 32 → 31 → 14 → 2 → 30 → 29 → 28 → 13 → 37 → 27 → 26 → 12 → 4 → 25 → 24 → 23 → 11 → 38 → 22 → 21 → 10 → 6 → 19 → 20 → 18 → 9 → 7 → 8 → 16 → 17 → 39 → 40 → 59 → 60 → 61 → 62 → 58 → 57 →

41 → 63 → 82 → 85 → 106 → 105 → 104 → 86 → 103 → 110 → 111 → 102 → 101 → 100 → 87 → 56 → 64 → 65 → 66 → 55 → 54 → 43 → 67 → 80 → 88 → 99 → 112 → 113 → 79 → 89 → 98 → 97 → 96 → 78 → 68 → 53 → 44 → 69 → 70 → 71 → 52 → 51 → 45 → 46 → 50 → 72 → 74 → 73 → 49 → 48 → 47 → 75 → 76 → 92 → 93 → 94 → 91 → 77 → 90 → 95 → 114 → 115 → 123 → 124 → 125 → 151 → 149 → 150 → 126 → 127 → 122 → 121 → 128 → 129 → 148 → 147 → 146 → 130 → 131 → 120 → 119 → 132 → 133 → 145 → 144 → 143 → 134 → 135 → 118 → 117 → 136 → 137 → 142 → 141 → 140 → 138 → 139 → 116 → 109 → 108 → 107 → 84 → 83 → 81 → 42 → 5 → 3 → 1 → depot

Optimal solution of u159

depot → 158 → 2 → 1 → 3 → 4 → 155 → 156 → 154 → 153 → 152 → 151 → 6 → 150 → 149 → 148 → 145 → 146 → 147 → 7 → 8 → 9 → 11 → 10 → 143 → 144 → 142 → 12 → 13 → 14 → 15 → 16 → 141 → 140 → 138 → 139 → 137 → 36 → 35 → 37 → 38 → 39 → 40 → 41 → 42 → 43 → 44 → 45 → 46 → 47 → 48 → 49 → 50 → 51 → 52 → 53 → 62 → 61 → 63 → 64 → 65 → 66 → 67 → 68 → 69 → 70 → 71 → 77 → 78 → 79 → 80 → 81 → 82 → 83 → 84 → 85 → 96 → 97 → 98 → 99 → 100 → 101 → 102 → 103 → 104 → 105 → 106 → 94 → 95 → 86 → 87 → 93 → 88 → 89 → 90 → 91 → 92 → 107 → 108 → 109 → 110 → 111 → 112 → 113 → 114 → 115 → 116 → 117 → 118 → 119 → 120 → 121 → 122 → 129 → 123 → 128 → 127 → 130 → 131 → 133 → 132 → 134 → 135 → 136 → 34 → 33 → 32 → 31 → 30 → 18 → 19 → 20 → 17 → 21 → 22 → 23 → 24 → 26 → 25 → 27 → 28 → 29 → 54 → 55 → 56 → 57 → 58 → 59 → 60 → 72 → 73 → 74 → 75 → 76 → 126 → 124 → 125 → 157 → 5 → depot

Optimal solution of si175

depot → 1 → 9 → 8 → 7 → 6 → 35 → 34 → 36 → 38 → 39 → 40 → 81 → 41 → 42 → 43 → 82 → 44 → 45 → 83 → 47 → 46 → 48 → 50 → 49 → 51 → 53 → 54 → 55 → 57 → 59 → 58 → 60 → 61 → 63 → 66 → 65 → 67 → 68 → 69 → 70 → 71 → 72 → 73 → 74 → 75 → 76 → 108 → 94 → 107 → 93 → 106 → 92 → 91 → 105 → 90 → 104 → 89 → 103 → 88 → 102 → 147 → 146 → 112 → 113 → 158 → 168 → 161 → 157 → 114 → 156 → 115 → 155 → 160 → 166 → 165 → 116 → 164 → 154 → 153 → 152 → 151 → 163 → 162 → 174 → 117 → 119 → 133 → 121 → 122 → 123 → 124 → 125 → 126 → 127 → 128 → 129 → 130 → 131 → 132 → 120 → 139 → 145 → 144 → 143 → 142 → 134 → 140 → 141 → 138 → 137 → 136 → 135 → 173 → 172 → 171 → 111 → 167 → 170 → 169 → 159 → 79 → 37 → 80 → 97 → 98 → 84 → 100 → 99 → 85 → 101 → 86 → 56 → 21 → 62 → 64 → 23 → 24 → 25 → 26 → 27 → 28 → 29 → 30 → 118 → 31 → 3 → 22 → 20 → 19 → 18 → 17 → 16 → 15 → 14 → 13 → 12 → 11 → 10 → 5 → 4 →

33 → 32 → 78 → 77 → 95 → 96 → 52 → 87 → 109 → 150 → 149 → 148 → 110 → 2 → depot

Optimal solution of brg180

depot → 11 → 10 → 9 → 8 → 7 → 6 → 5 → 4 → 3 → 2 → 1 → 165 → 166 → 167 → 156 → 157 → 158 → 159 → 160 → 161 → 162 → 163 → 164 → 129 → 130 → 131 → 120 → 121 → 122 → 123 → 124 → 125 → 126 → 127 → 128 → 83 → 72 → 73 → 74 → 75 → 76 → 77 → 78 → 79 → 80 → 81 → 82 → 143 → 132 → 133 → 134 → 135 → 136 → 137 → 138 → 139 → 140 → 141 → 142 → 95 → 84 → 85 → 86 → 87 → 88 → 89 → 90 → 91 → 92 → 93 → 94 → 119 → 108 → 109 → 110 → 111 → 112 → 113 → 114 → 115 → 116 → 117 → 118 → 107 → 96 → 97 → 98 → 99 → 100 → 101 → 102 → 103 → 104 → 105 → 106 → 45 → 46 → 47 → 36 → 37 → 38 → 39 → 40 → 41 → 42 → 43 → 44 → 71 → 60 → 61 → 62 → 63 → 64 → 65 → 66 → 67 → 68 → 69 → 70 → 57 → 58 → 59 → 48 → 49 → 50 → 51 → 52 → 53 → 54 → 55 → 56 → 155 → 144 → 145 → 146 → 147 → 148 → 149 → 150 → 151 → 152 → 153 → 154 → 21 → 22 → 23 → 12 → 13 → 14 → 15 → 16 → 17 → 18 → 19 → 20 → 179 → 168 → 169 → 170 → 171 → 172 → 173 → 174 → 175 → 176 → 177 → 178 → 33 → 34 → 35 → 24 → 25 → 26 → 27 → 28 → 29 → 30 → 31 → 32 → depot

Optimal solution of rat195

depot → 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11 → 12 → 25 → 24 → 23 → 22 → 21 → 20 → 19 → 18 → 17 → 16 → 15 → 14 → 27 → 28 → 29 → 30 → 31 → 32 → 33 → 34 → 35 → 36 → 49 → 48 → 47 → 46 → 45 → 44 → 43 → 42 → 41 → 40 → 39 → 52 → 53 → 54 → 55 → 56 → 57 → 58 → 59 → 60 → 61 → 62 → 63 → 64 → 77 → 76 → 75 → 74 → 73 → 72 → 71 → 70 → 69 → 68 → 67 → 66 → 65 → 78 → 79 → 80 → 81 → 82 → 83 → 84 → 85 → 86 → 87 → 88 → 89 → 90 → 103 → 102 → 101 → 100 → 99 → 98 → 97 → 96 → 95 → 94 → 93 → 92 → 91 → 104 → 105 → 106 → 107 → 108 → 109 → 110 → 111 → 112 → 113 → 126 → 125 → 124 → 123 → 122 → 121 → 120 → 119 → 118 → 117 → 130 → 131 → 132 → 133 → 134 → 135 → 136 → 137 → 138 → 139 → 140 → 153 → 152 → 151 → 150 → 149 → 148 → 147 → 146 → 145 → 144 → 143 → 156 → 157 → 158 → 159 → 160 → 161 → 162 → 163 → 164 → 165 → 166 → 179 → 178 → 177 → 176 → 175 → 174 → 173 → 172 → 171 → 170 → 169 → 182 → 183 → 184 → 185 → 186 → 187 → 188 → 189 → 190 → 191 → 192 → 193 → 194 → 181 → 180 → 168 → 167 → 155 → 154 → 141 → 142 → 129 → 128 → 127 → 114 → 115 → 116 → 51 → 50 → 38 → 37 → 26 → 13 → depot

Best known solution of d198

depot → 1 → 6 → 5 → 2 → 3 → 4 → 7 → 8 → 9 → 10 → 11 → 12 → 85 → 99 → 84 → 70 → 68 → 57 → 56 → 41 → 42 → 55 → 43 → 54 → 59 → 58 → 67 → 66 → 69 → 72 → 71 → 83 → 86 → 98 → 97 → 96 → 87 → 82 → 73 → 65 → 60 → 53 → 44 → 37 → 30 → 31 → 36 → 35 → 32 → 33 → 27 → 29 → 28 → 22 → 21 → 23 → 20 → 24 → 25 → 26 → 34 → 38 → 46 → 50 → 49 → 47 → 48 → 62 → 78 → 91 → 90 → 79 → 77 → 63 → 76 → 75 → 80 → 64 → 51 → 45 → 52 → 61 → 74 → 81 → 88 → 89 → 95 → 94 → 93 → 92 → 100 → 101 → 102 → 103 → 104 → 113 → 112 → 111 → 105 → 106 → 110 → 109 → 108 → 107 → 119 → 118 → 117 → 116 → 115 → 114 → 121 → 120 → 122 → 123 → 168 → 124 → 125 → 130 → 129 → 132 → 131 → 133 → 139 → 140 → 142 → 141 → 146 → 145 → 148 → 147 → 152 → 151 → 161 → 160 → 159 → 158 → 157 → 156 → 155 → 154 → 153 → 138 → 137 → 134 → 135 → 128 → 126 → 169 → 127 → 136 → 143 → 144 → 149 → 150 → 162 → 163 → 164 → 165 → 171 → 170 → 181 → 182 → 180 → 176 → 175 → 172 → 173 → 174 → 177 → 179 → 183 → 184 → 178 → 193 → 194 → 197 → 196 → 195 → 185 → 192 → 191 → 186 → 187 → 190 → 189 → 188 → 167 → 166 → 19 → 18 → 17 → 16 → 15 → 14 → 13 → 39 → 40 → depot

Best known solution of kroA200

depot → 52 → 114 → 116 → 110 → 131 → 84 → 144 → 190 → 26 → 197 → 122 → 14 → 12 → 78 → 159 → 161 → 63 → 19 → 54 → 41 → 134 → 185 → 126 → 111 → 119 → 46 → 30 → 156 → 106 → 108 → 5 → 53 → 74 → 154 → 182 → 21 → 133 → 7 → 16 → 24 → 142 → 89 → 33 → 57 → 140 → 170 → 199 → 97 → 113 → 87 → 147 → 27 → 38 → 37 → 55 → 151 → 177 → 195 → 70 → 129 → 71 → 82 → 61 → 184 → 167 → 172 → 22 → 168 → 67 → 34 → 1 → 180 → 124 → 160 → 79 → 76 → 157 → 192 → 127 → 59 → 100 → 3 → 162 → 92 → 105 → 148 → 18 → 91 → 9 → 174 → 35 → 56 → 73 → 99 → 155 → 32 → 44 → 196 → 80 → 96 → 103 → 164 → 95 → 165 → 51 → 10 → 83 → 169 → 121 → 115 → 187 → 43 → 62 → 193 → 50 → 15 → 117 → 123 → 137 → 8 → 77 → 81 → 198 → 25 → 60 → 135 → 31 → 23 → 158 → 173 → 120 → 45 → 171 → 48 → 17 → 109 → 28 → 183 → 36 → 178 → 152 → 65 → 118 → 98 → 13 → 191 → 107 → 68 → 141 → 179 → 130 → 188 → 72 → 2 → 58 → 40 → 88 → 153 → 20 → 139 → 163 → 101 → 75 → 69 → 143 → 149 → 90 → 94 → 93 → 181 → 194 → 112 → 175 → 132 → 136 → 42 → 104 → 4 → 85 → 138 → 49 → 128 → 102 → 145 → 66 → 176 → 64 → 186 → 150 → 29 → 166 → 146 → 39 → 11 → 189 → 47 → 125 → 86 → 6 → depot

Best known solution of kroB200

depot → 94 → 97 → 31 → 166 → 174 → 58 → 135 → 28 → 7 → 98 → 75 → 96 → 90 → 128 → 27 → 157 → 10 → 92 → 121 → 84 → 72 → 52 → 169 → 148 → 69 → 197 → 38 → 39 → 116 → 189 → 110 → 67 → 48 → 85 → 137 → 165 → 107 → 151 → 37 →

19 → 79 → 180 → 133 → 29 → 74 → 100 → 149 → 68 → 25 → 99 → 55 → 112 → 178 → 78 → 187 → 126 → 123 → 64 → 140 → 183 → 36 → 71 → 168 → 6 → 83 → 57 → 51 → 115 → 162 → 53 → 153 → 190 → 176 → 142 → 170 → 87 → 22 → 21 → 192 → 143 → 159 → 152 → 23 → 17 → 44 → 35 → 118 → 95 → 136 → 18 → 150 → 91 → 199 → 101 → 147 → 40 → 191 → 16 → 77 → 154 → 12 → 186 → 125 → 62 → 30 → 47 → 194 → 155 → 120 → 81 → 63 → 13 → 124 → 119 → 41 → 108 → 1 → 15 → 104 → 76 → 102 → 182 → 139 → 105 → 158 → 49 → 184 → 42 → 195 → 185 → 59 → 73 → 164 → 86 → 113 → 167 → 132 → 65 → 144 → 198 → 177 → 130 → 156 → 171 → 34 → 60 → 26 → 175 → 179 → 93 → 56 → 33 → 8 → 24 → 45 → 145 → 160 → 70 → 106 → 11 → 173 → 89 → 20 → 122 → 141 → 9 → 188 → 66 → 196 → 4 → 131 → 193 → 61 → 80 → 46 → 172 → 138 → 103 → 54 → 163 → 88 → 117 → 181 → 82 → 3 → 5 → 32 → 146 → 14 → 111 → 127 → 50 → 109 → 43 → 129 → 134 → 161 → 114 → 2 → depot