Cadernos do LOGIS

A Generic Exact Solver for Vehicle Routing and Related Problems

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, François Vanderbeck

Volume 2019, Number 2

June, 2019

LOGIS
Núcleo de Logística Integrada e Sistemas

Universidade Federal Fluminense

A Generic Exact Solver for Vehicle Routing and Related Problems

Artur Pessoa[a,*], Ruslan Sadykov[b], Eduardo Uchoa[a], Franois Vanderbeck[c]

[a] *a Universidade Federal Fluminense, Rua Passo da Patria, 156/309-D, Niteroi – RJ, 24210-240, Brazil.*
[b] *INRIA Bordeaux - Sud-Ouest, 200 avenue de la Vieille Tour, 33405 Talence, France.*
[c] *Atoptima, 351 cours de la Liberation, 33400 Talence, France.*

Abstract

Major advances were recently obtained in the exact solution of Vehicle Routing Problems (VRPs). Sophisticated Branch-Cut-and-Price (BCP) algorithms for some of the most classical VRP variants now solve many instances with up to a few hundreds of customers. However, adapting and reimplementing those successful algorithms for other variants can be a very demanding task. This work proposes a BCP solver for a generic model that encompasses a wide class of VRPs. It incorporates the key elements found in the best existing VRP algorithms: ng-path relaxation, rank-1 cuts with limited memory, path enumeration, and rounded capacity cuts; all generalized through the new concept of "packing set". This concept is also used to derive a branch rule based on accumulated resource consumption and to generalize the Ryan and Foster branch rule. Extensive experiments on several variants show that the generic solver has an excellent overall performance, in many problems being better than the best specific algorithms. Even some non-VRPs, like bin packing, vector packing and generalized assignment, can be modeled and effectively solved.

Keywords: Integer Programming, Column Generation, Routing

1. Introduction

Since its introduction by Dantzig and Ramser (29), the Vehicle Routing Problem (VRP) has been one of the most widely studied in combinatorial optimization. Google Scholar indicates that 691 works containing the exact string "vehicle routing" in the title were published only in 2018. VRP relevance stems from its direct use in the real systems that distribute goods and provide services, vital to the modern economies. Reflecting the large variety of conditions in those systems, the VRP literature is spread into dozens, perhaps hundreds, of variants. For example, there are variants that consider capacities, time windows, heterogeneous fleets, multiple depots, pickups and deliveries, optional customer visits, arc routing, etc.

*Corresponding author
**A short version of this work was already published in (66).
Email addresses: artur@producao.uff.br (Artur Pessoa), ruslan.sadykov@inria.fr (Ruslan Sadykov), uchoa@producao.uff.br (Eduardo Uchoa), fv@atoptima.com (Franois Vanderbeck)

In recent years, big advances in the exact solution of VRPs had been accomplished. A milestone was certainly the Branch-Cut-and-Price (BCP) algorithm of (59; 61), that could solve Capacitated VRP (CVRP) instances with up to 360 customers, a large improvement upon the previous record of 150 customers. That algorithm exploits many elements introduced by several authors, combining and enhancing them. In particular, the new concept of *limited memory cut* proved to be pivotal. Improvements of the same magnitude were later obtained for a number of classical variants like VRP with Time Windows (VRPTW) (60), Heterogeneous Fleet VRP (HFVRP) and Multi Depot VRP (MDVRP) (64), and Capacitated Arc Routing (CARP) (63). For all those variants, instances with about 200 customers are now likely to be solved, perhaps in hours or even days. However, there is something even more interesting: many instances with about 100 customers, that a few years ago would take hours, are solved in less than 1 minute. This means that many more real world instances can now be tackled by exact algorithms in reasonable times.

Unhappily, designing and coding each one of those complex and sophisticated BCPs has been a highly demanding task, measured on several work-months of a skilled team. In effect, this prevents the use of those algorithms in real world problems, that actually, seldom correspond exactly to one of the most classical variants. This work presents a framework that can handle most VRP variants found in the literature and can be used to model and solve many other new variants. In order to obtain state-of-the-art BCP performance, some key elements found in the best specific VRP algorithms had to be generalized. The new concept of *packing set* was instrumental for that.

The quest for general exact VRP algorithms can be traced back to Balinski and Quandt (11), where a set partitioning formulation valid for many variants was proposed. That formulation had only turned practical in the 1980's and 1990's, when the Branch-and-Price (BP) method was developed. At that time, it was recognized that the pricing subproblems could often be modeled as Resource Constrained Shortest Path (RCSP) problems and solved by labeling algorithms, leading to quite generic methods (for example, Desaulniers et al.(32)). However, those BP algorithms only worked well on problems with "tightly constrained" routes, like VRPTW with narrow time windows. Many variants, including CVRP, were much better handled by Branch-and-Cut (BC) algorithms using problem-specific cuts (for example, Lysgaard et al. (56)). In the late 2000's decade, after works like (39; 8; 48; 33; 74; 7; 17), it became clear that the combination of cut and column generation performs better than pure BP or pure BC on almost all problems. Until today, BCP remains the dominant VRP approach. A first attempt of a generic BCP was presented in Baldacci and Mingozzi (9), where 7 variants, all of them particular cases of the HFVRP, could be solved. Recently, (76) proposed a BCP for several particular cases of the HFVRP with time windows. The framework now proposed is far more generic than that.

2. The Basic Model

In this section, we provide a formal definition of a generic model that can be solved by a branch-cut-and-price algorithm, where all pricing subproblems are modeled as RCSP problems.

2.1. Graphs for RCSP Generation

Define directed graphs $G^k = (V^k, A^k)$, $k \in K$. Let $V = \cup_{k \in K} V^k$ and $A = \cup_{k \in K} A^k$. The graphs are not necessarily simple and may even have loops. Vertices and arcs in all graphs are distinct. Each graph has special source and sink vertices: v^k_{source} and v^k_{sink}. The source and sink may be distinct vertices, but may also be the same vertex. Define set R^k of resources. For each $r \in R^k$ and $a \in A^k$, $q_{a,r} \in \mathbb{R}$ is the consumption of resource r in arc a. Resources without any negative consumption are called *monotone*, otherwise they are *non-monotone*. Set R^k is divided into *main resources* R^k_M and *secondary resources* R^k_N. Main resources should be monotone. Moreover, there should not exist a cycle in G^k with zero consumption of all main resources. Therefore, unless G^k is acyclic, it is mandatory the existence of at least one main resource. As will be discussed in Section 5, the concept of main resource is directly related to key implementation issues. Secondary resources may be monotone or non-monotone. Finally, resources are also classified as *disposable* or *non-disposable*. By default, resources are assumed to be disposable. The existence of non-disposable resources should be explicitly indicated. There are finite accumulated resource consumption intervals $[l_{a,r}, u_{a,r}]$, $a \in A^k$. Since in most applications these intervals are more naturally defined on vertices, we may define intervals $[l_{v,r}, u_{v,r}]$, $v \in V^k$, meaning that $[l_{a,r}, u_{a,r}] = [l_{v,r}, u_{v,r}]$ for every arc $a \in \delta^-(v)$ (i.e., entering v). A resource constrained path $p = (v^k_{\text{source}} = v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n = v^k_{\text{sink}})$ over a graph G^k should have $n \geq 1$ arcs, $v_j \neq v^k_{\text{source}}$ and $v_j \neq v^k_{\text{sink}}$, $1 \leq j \leq n-1$, and is feasible if:

- for every $r \in R^k$ that is disposable, the accumulated resource consumption $S_{j,r}$ at visit j, $0 \leq j \leq n$, where $S_{0,r} = 0$ and $S_{j,r} = \max\{l_{a_j,r}, S_{j-1,r} + q_{a_j,r}\}$, does not exceed $u_{a_j,r}$;
- for every $r \in R^k$ that is non-disposable, the accumulated resource consumption $S_{j,r}$ at visit j, $0 \leq j \leq n$, where $S_{0,r} = 0$ and $S_{j,r} = S_{j-1,r} + q_{a_j,r}$, lies in the interval $[l_{a_j,r}, u_{a_j,r}]$.

Note that some feasible paths may not be elementary, some vertices or arcs being visited more than once. For each $k \in K$, let P^k denote the set of all feasible resource constrained paths in G^k. Each set P^k is finite, either because G^k is acyclic or because the main resources limit the number of times that each vertex or arc can be visited. Define $P = \cup_{k \in K} P^k$. As vertices and arcs in different graphs are distinct, paths in different graphs are also distinct.

2.2. Formulation

For all $a \in A$ and $p \in P$, let h^p_a indicate how many times arc a appears in path p. The problem should be formulated as follows.

$$\text{Min} \quad \sum_{j=1}^{n_1} c_j x_j + \sum_{s=1}^{n_2} f_s y_s \tag{1a}$$

$$\text{S.t.} \quad \sum_{j=1}^{n_1} \alpha_{ij} x_j + \sum_{s=1}^{n_2} \beta_{is} y_s \geq d_i, \qquad i = 1, \ldots, m, \tag{1b}$$

$$x_j = \sum_{k \in K} \sum_{p \in P^k} \left(\sum_{a \in M(x_j)} h_a^p \right) \lambda_p, \qquad j = 1 \ldots, n_1, \tag{1c}$$

$$L^k \leq \sum_{p \in P^k} \lambda_p \leq U^k, \qquad k \in K, \tag{1d}$$

$$\lambda_p \in \mathbb{Z}_+, \qquad p \in P, \tag{1e}$$

$$x_j \in \mathbb{Z},\ y_s \in \mathbb{Z}, \qquad j = 1, \ldots, \bar{n}_1,\ s = 1, \ldots, \bar{n}_2, \tag{1f}$$

where x_j, $1 \leq j \leq n_1$, y_s, $1 \leq s \leq n_2$, and λ_p, $p \in P$, are variables. The first $\bar{n}_1$ x variables and the first $\bar{n}_2$ y are defined to be integer; all λ variables are non-negative integer. Equations (1a) and (1b) define a general objective function and m general constraints over those variables, respectively. Constraints (1b) may even contain exponentially large families of cuts, provided that suitable procedures are given for their separation. However, by simplicity, we continue the presentation as if all the m constraints are explicitly defined. For each variable x_j, $1 \leq j \leq n_1$, $M(x_j) \subseteq A$ defines its *mapping* into a non-empty subset of the arcs. We remark that mappings do not need to be disjoint, the same arc can mapped to more than one variable x_j. Define $M^{-1}(a)$ as $\{j \,|\, a \in M(x_j)\}$. As not all arcs need to belong to some mapping, some M^{-1} sets may be empty. The relation between variables x and λ is given by (1c). For each $k \in K$, L^k and U^k are given lower and upper bounds on the number of paths from G^k in a solution. Eliminating the x variables and relaxing the integrality constraints, the following LP is obtained:

$$\text{Min} \quad \sum_{k \in K} \sum_{p \in P^k} \left(\sum_{j=1}^{n_1} c_j \sum_{a \in M(x_j)} h_a^p \right) \lambda_p + \sum_{s=1}^{n_2} f_s y_s \tag{2a}$$

$$\text{S.t.} \quad \sum_{k \in K} \sum_{p \in P^k} \left(\sum_{j=1}^{n_1} \alpha_{ij} \sum_{a \in M(x_j)} h_a^p \right) \lambda_p + \sum_{s=1}^{n_2} \beta_{is} y_s \geq d_i, \qquad i = 1, \ldots, m, \tag{2b}$$

$$L^k \leq \sum_{p \in P^k} \lambda_p \leq U^k, \qquad k \in K, \tag{2c}$$

$$\lambda_p \geq 0, \qquad p \in P. \tag{2d}$$

Master LP (2) is solved by column generation. Let π_i, $1 \leq i \leq m$, denote the dual variables of Constraints (2b), ν_+^k and ν_-^k, $k \in K$, are the dual variables of Constraints (2c). The reduced cost of an arc $a \in A$ is defined as:

$$\bar{c}_a = \sum_{j \in M^{-1}(a)} c_j - \sum_{i=1}^{m} \sum_{j \in M^{-1}(a)} \alpha_{ij} \pi_i.$$

The reduced cost of a path $p = (v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n) \in P^k$ is:

$$\bar{c}(p) = \sum_{j=1}^{n} \bar{c}_{a_j} - \nu_+^k - \nu_-^k.$$

So, the pricing subproblems correspond to finding, for each $k \in K$, a path $p \in P^k$ with minimum reduced cost.

3. Generalizing State-of-the-Art Elements: Packing Sets

Formulation (1) can be used to model most VRP variants and also some other non-VRP applications. It can be solved by a standard BP algorithm (or a standard robust BCP algorithm (69), if (1b) contains separated constraints), where the RCSP subproblems are handled by a labeling dynamic programming algorithm. However, its performance on the more classic VRP variants would be very poor when compared to the best existing specific algorithms. One of the main contributions of this work is a generalization of the key additional concepts found in those state-of-the-art algorithms, leading to the construction of a powerful and still quite generic BCP algorithm.

In order to do that, we introduce a new concept. Let $\mathcal{B} \subset 2^A$ be a collection of mutually disjoint subsets of A such that the constraints:

$$\sum_{p \in P} \left(\sum_{a \in B} h_a^p \right) \lambda_p \leq 1, \quad B \in \mathcal{B}, \tag{3}$$

are satisfied by at least one optimal solution (x^*, y^*, λ^*) of Formulation (1). In those conditions, we say that each element of $\mathcal{B}$ is a *packing set*. Note that a packing set can contain arcs from different graphs and not all arcs in A need to belong to some packing set. The definition of a proper collection $\mathcal{B}$ is application specific and part of the modeling task. It does not follow automatically from the analysis of Formulation (1).

In many applications the packing sets are more naturally defined on vertices, so we also provide that modeling alternative. Let coefficient h_v^p indicate how many times vertex v appears in a path p. Let $\mathcal{B}^{\mathcal{V}} \subset 2^V$ be a collection of mutually disjoint subsets of V such that the constraints:

$$\sum_{p \in P} \left(\sum_{v \in B} h_v^p \right) \lambda_p \leq 1, \quad B \in \mathcal{B}^{\mathcal{V}}, \tag{4}$$

are satisfied by at least one optimal solution (x^*, y^*, λ^*) of Formulation (1). In those conditions, we say that the elements of $\mathcal{B}^{\mathcal{V}}$ are *packing sets on vertices*. Actually, in Section 5.1.1 we show that in some symmetric problems there is a computational advantage in defining packing sets on vertices.

The following concepts — *ng*-paths, Limited Memory Rank-1 Cuts, path enumeration, accumulated consumption branching, and rounded capacity cuts — were originally proposed and used on the most classical VRP variants, often CVRP and VRPTW. In our proposed generalization, those problems will correspond to simple models where the packing sets in $\mathcal{B}^{\mathcal{V}}$ are the singletons formed by each customer vertex.

3.1. ng-paths

When modeling classical VRPs, one of the weaknesses of linear relaxation (2) is often the existence of non-elementary paths in P that can not be part of any integer solution. In those cases, one would like to eliminate all those paths from the definition of P. However, this would make the pricing subproblems much harder, to the point of becoming intractable in many cases. A good compromise between formulation strength and pricing difficulty can be obtained by the so-called *ng*-paths, introduced in Baldacci et al. (10).

In our more general context, we say that a path is $\mathcal{B}$-*elementary* if it does not use more than one arc in the same packing set of $\mathcal{B}$. Let P^k_{elem} be the subset of the paths in P^k that are $\mathcal{B}$-elementary, $P_{elem} = \cup_{k \in K} P^k_{elem}$.

Ideally, we would like to price only $\mathcal{B}$-elementary paths. Instead, we settle for generalized $\mathcal{B}$-*ng*-paths defined as follows. For each arc $a \in A$, let $NG(a) \subseteq \mathcal{B}$ denote the *ng*-set of a. A $\mathcal{B}$-*ng*-path may use two arcs belonging to the same packing set B, but only if the subpath between those two arcs passes by an arc a such that $B \notin NG(a)$. The *ng*-sets may be determined a priori; but also dynamically, like in (72) and (21).

If the packing sets are being defined on vertices, there is the similar concept of $\mathcal{B}^{\mathcal{V}}$-elementary path: a path that does not use more than one vertex in the same packing set of $\mathcal{B}^{\mathcal{V}}$. We also denote by P^k_{elem} the subset of the paths in P^k that are $\mathcal{B}^{\mathcal{V}}$-elementary. In this context, for each vertex $v \in V$, let $NG(v) \subseteq \mathcal{B}^{\mathcal{V}}$ be the *ng*-set of v. A $\mathcal{B}^{\mathcal{V}}$-*ng*-path may use two vertices belonging to the same packing set B, but only if the subpath between those two vertices passes by a vertex v such that $B \notin NG(v)$.

When $\mathcal{B}$ or $\mathcal{B}^{\mathcal{V}}$ are clear from the context, we may still refer to $\mathcal{B}$-*ng*-paths or $\mathcal{B}^{\mathcal{V}}$-*ng*-paths simply as *ng*-paths.

3.2. Limited Memory Rank-1 Cuts

The Rank-1 Cuts (R1Cs) (67; 62; 20) are a generalization of the Subset Row Cuts proposed by Jepsen et al. (48). Here, they are further generalized as follows. Consider a collection of packing sets $\mathcal{B}$. A Chvátal-Gomory rounding of Constraints (4), using a non-negative multiplier ρ_B for each $B \in \mathcal{B}$, yields:

$$\sum_{p \in P} \left\lfloor \sum_{B \in \mathcal{B}} \rho_B \sum_{a \in B} h^p_a \right\rfloor \lambda_p \leq \left\lfloor \sum_{B \in \mathcal{B}} \rho_B \right\rfloor . \tag{5}$$

Those R1Cs are potentially very strong, but each added cut makes the pricing subproblems significantly harder.

The limited memory technique (59) is essential for mitigating that negative impact. In this technique, a R1C, characterized by its vector of multipliers ρ, is associated to a *memory arc-set* $A(\rho) \subseteq A$. The limited-memory R1C (lm-R1C) is defined as:

$$\sum_{p \in P} \alpha(\rho, A(\rho), p)\lambda_p \leq \left\lfloor \sum_{B \in \mathcal{B}} \rho_B \right\rfloor, \tag{6}$$

where the coefficient $\alpha(\rho, A(\rho), p)$ is computed as in the pseudo-code that describes Function α.

Function $\alpha(\rho, A, p = (v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n))$

```
1 α ← 0, S ← 0;
2 for j = 1 to n do
3     if a_j ∉ A(ρ) then
4         S ← 0;
5     if a_j ∈ B ∈ B then
6         S ← S + ρ_B;
7         if S ≥ 1 then
8             S ← S − 1, α ← α + 1;
9 return α;
```

If $A(\rho) = A$, constraints (5) and (6) are identical. Otherwise, variables λ_p corresponding to paths p passing by arcs $a \notin A(\rho)$ may have their coefficients decreased. However, if the memory sets are adjusted in such a way that variables λ_p with positive values in the current linear relaxation have the same coefficients that they would have in (5), the resulting lm-R1C is as effective as the original R1C. Yet, if the final $A(\rho)$ is a small subset of A, as usually happens, the impact in the pricing is much reduced.

If the model defines its packing sets in vertices, the R1Cs are defined in a similar way. There is a non-negative multiplier ρ_B for each $B \in \mathcal{B}^{\mathcal{V}}$ and the cut is:

$$\sum_{p \in P} \left\lfloor \sum_{B \in \mathcal{B}^{\mathcal{V}}} \rho_B \sum_{v \in B^V} h_v^p \right\rfloor \lambda_p \leq \left\lfloor \sum_{B \in \mathcal{B}^{\mathcal{V}}} \rho_B \right\rfloor. \tag{7}$$

Given a memory arc-set $A(\rho) \subseteq A$ corresponding to the vector ρ, the lm-R1C is defined as:

$$\sum_{p \in P} \alpha(\rho, A(\rho), p)\lambda_p \leq \left\lfloor \sum_{B \in \mathcal{B}^{\mathcal{V}}} \rho_B \right\rfloor, \tag{8}$$

where Function Alpha is the same, except that the condition in line 5 is replaced by $(v_j \in B \in \mathcal{B}^{\mathcal{V}})$.

Regardless of if the packing sets are being defined on arcs or on vertices, it is possible to use lm-R1Cs where the memories are defined by vertex-sets. In this case, a *memory vertex-set* $V(\rho) \subseteq V$

should be assigned to the lm-R1C corresponding to vector ρ. Function Alpha should receive $V(\rho)$ instead of $A(\rho)$ as parameter and the condition in line 3 should be changed to $(v_j \notin V(\rho))$.

As discussed in (64) and (76), memory vertex-sets perform better for most instances of some classical VRPs. This happens because R1C memory adjustment converges in less iterations in that case. In the other hand, memory arc-sets may be better for some harder instances; because they allow for a finer memory adjustment, leading to less impact in the pricing.

3.3. Path Enumeration

The path enumeration technique was proposed by Baldacci et. al. (8), and later improved by Contardo and Martinelli (27). It consists in trying to enumerate into a pool all paths in a certain set P^k that can possibly be part of an improving solution. After a successful enumeration, the corresponding pricing subproblem k can be solved by inspection, saving time. Moreover, standard fixing by reduced costs can be used to remove paths from the pools. If the enumeration has already succeeded for all $k \in K$ and once the total number of paths in the tables is reduced to a reasonably small number (say, less than 10,000), the formulation restricted to those paths can be given and directly solved by a general MIP solver.

In our context, we try to enumerate all paths $p \in P^k_{elem}$ such that $\bar{c}(p) < UB - LB$, where UB is the best known integer solution cost, and LB the value of the current linear relaxation. Moreover, if two paths p and p' in P^k map to variables λ_p and λ'_p with identical coefficients in the essential constraints in (2b), the one with a larger cost is *dominated* and can be dropped. The *essential constraints* are those that are required to make the formulation valid, constraints in (2b) added only to strengthen the linear relaxation are not essential.

However, the enumeration procedure would be highly inefficient if the dominance could only be checked for pairs of complete paths. Instead, it is necessary to perform dominance over the partial paths ($\mathcal{B}$-elementary paths starting at the source vertex) that are being constructed along the procedure. Our procedure uses the following dominance rule: if p and p' are partial paths ending at the same vertex and having already visited exactly the same packing sets in $\mathcal{B}$ (regardless of the visitation order), the one with larger cost (breaking ties arbitrarily) is considered dominated and dropped. No complete path in P^k_{elem} that is the completion of a dominated partial path will be produced. The following condition is sufficient to assure that such enumeration procedure is valid (i.e., no improving solution is ever missed):

Sufficient Condition for Enumeration. *Every two feasible partial $\mathcal{B}$-elementary paths starting in v^k_{source} that end in the same vertex and map to different coefficients in some essential constraint in* (2b) *should have visited different subsets of $\mathcal{B}$.*

We remark that the above condition can not be checked automatically. In fact, in general it is not even possible to automatically determine what are the essential constraints in (2b). It is up to the modeler to prove that the provided model satisfies the sufficient condition, so enumeration can be used. Happily, in many models (including all the examples in Section 4) it is easy to prove

that the condition is satisfied. However, if it is not satisfied, it is up to the modeler to prove that the enumeration is valid for his model directly from the dominance rule. Otherwise, the enumeration should be turned off.

3.4. Branching

Branching over individual x and y variables (or over constraints defined over those variables) is simple and do not change the structure of the pricing subproblems. In many models this kind of branching is sufficient for correctness. However, there are models where Constraints (1e) need to be explicitly enforced. However, branching over individual λ variables should be avoided due to a big negative impact in the pricing and also due to highly unbalanced branch trees (82). The model offers two ways of branching over sets of λ variables:

- Choose distinct sets B and B' in $\mathcal{B}$. Let $P(B, B') \subseteq P$ be the subset of the paths that contain arcs in both B and B'. The branch is over the value of $\sum_{p \in P(B,B')} \lambda_p$, either 0 or 1. This is a generalization of the *Ryan and Foster* branch rule (75). It is still to be avoided if possible, because it makes the pricing harder. However, using that scheme leads to more balanced search trees.

- Choose $B \in \mathcal{B}$, $r \in R_M^k$ and a certain threshold value t^*: in the left child make $u_{a,r} = t^*$, for all $a \in B \cap G^{k(a)}$; in the right child make $l_{a,r} = t^*$. This *branching over the accumulated consumption of a resource* generalizes the strategy proposed by Gélinas et al. (41). The branching is not likely to be complete, in the sense that some fractional λ solutions can not be eliminated by it. However, it does not increase the pricing difficulty and it may work well in practice, postponing (and even avoiding) the use of a branching that makes pricing harder.

3.5. Rounded Capacity Cut Separators

The Rounded Capacity Cuts (RCCs), first proposed for CVRP (51), are still useful on modern BCP algorithms for that problem and also for a number of other VRP variants. Moreover, a very good heuristic separation routine is available for it in CVRPSEP library (55). So, we decided to introduce the concept of *RCC separator* as a feature of our model.

The RCC separator can only be used if the packing sets are defined on vertices. For a vertex $v \in V$, define $B(v)$ as the packing set of $\mathcal{B}^{\mathcal{V}}$ that contains v, $B(v) = \emptyset$ if v is not in any packing set. An RCC separator is defined by setting a capacity Q and a demand function $d : \mathcal{B}^{\mathcal{V}} \cup \emptyset \to \mathbb{R}_+$ such that $d(\emptyset) = 0$ and is valid if there exists an optimal solution (x^*, y^*, λ^*) of Formulation (1) such that:

1. $\sum_{j=0}^{n} d(B(v_j)) \leq Q$, for all $p = (v_0, a_1, v_1, \ldots, a_{n-1}, v_{n-1}, a_n, v_n) \in P$ with $\lambda_p^* \geq 1$;
2. for all $B \in \mathcal{B}^{\mathcal{V}}$ such that $d(B) > 0$, the corresponding constraints in (4) should be satisfied with equality by (x^*, y^*, λ^*).

Again, it is up to the modeler to prove that the separator included in the model is valid.

Given a valid RCC separator, if $S \subseteq \mathcal{B}^{\mathcal{V}}$, $d(S)$ denotes $\sum_{B \in S} d(B)$ and h_S^p is the number of times that an arc in path $p \in P$ enters in S. We say that an arc (v_{j-1}, v_j) enters in S if $B(v_{j-1}) \notin S$ and $B(v_j) \in S$. A Rounded Capacity Cut is the following valid inequality:

$$\sum_{p \in P} h_S^p \lambda_p \geq \left\lceil \frac{d(S)}{Q} \right\rceil. \tag{9}$$

Cuts in format (9) are robust. The dual variable of the cut corresponding to an $S \subseteq \mathcal{B}^{\mathcal{V}}$ is simply subtracted from the reduced cost of all arcs entering S.

It is possible to define multiple RCC separators in the same model, each one having its demand function and capacity. This can be useful for modeling VRPs where routes are constrained by multiple dimensions. Packing sets would have zero demand in the dimensions that they do not "participate".

4. Model Examples

We selected a number of classical problems to illustrate the modeling capabilities of our solver.

4.1. Generalized Assignment Problem (GAP)

Data: Set T of tasks; set K of machines; capacity Q^k, $k \in K$; assignment cost c_t^k and machine load w_t^k, $t \in T$, $k \in K$.

Goal: Find an assignment of tasks to machines such that the total load in each machine does not exceed its capacity, with minimum total cost.

Model: RCSP generator graphs $G^k = (V^k, A^k)$ for each $k \in K$: $V^k = \{v_t^k : t = 0, \ldots, |T|\}$, $A^k = \{a_{t+}^k = (v_{t-1}^k, v_t^k), a_{t-}^k = (v_{t-1}^k, v_t^k) : t = 1, \ldots, |T|\}$, $v_{\text{source}}^k = v_0^k$, $v_{\text{sink}}^k = v_{|T|}^k$ (see Fig. 1); $R^k = R_M^k = \{r^k\}$; $q_{a_{t+}^k, r^k} = w_t^k, q_{a_{t-}^k, r^k} = 0, t \in T$; $[l_{v_t^k, r^k}, u_{v_t^k, r^k}] = [0, Q^k]$, $t \in T \cup \{0\}$. Integer variables x_t^k, $t \in T$, $k \in K$. The formulation is:

$$\text{Min} \quad \sum_{t \in T} \sum_{k \in K} c_t^k x_t^k \tag{10a}$$

$$\text{S.t.} \quad \sum_{k \in K} x_t^k = 1, \qquad t \in T; \tag{10b}$$

$L^k = 0$, $U^k = 1$, $k \in K$; $M(x_t^k) = \{a_{t+}^k\}, t \in T, k \in K$. $\mathcal{B} = \cup_{t \in T} \{\{a_{t+}^k : k \in K\}\}$. Branching is over the x variables. Enumeration is on.

Comments: Graphs G^k, illustrated in Figure 1, are designed to model binary knapsack constraints: each path in P^k corresponds to a possible assignment of a set of tasks to machine k. The basic formulation in this model is defined as follows. The objective function (10b) corresponds to the general objective (1a) and Constraints (10b) to the general constraints (1b). The definition of

variables x as integer yields integrality constraints corresponding to (1f). Constraints (1c) are indirectly defined by the RCSP graphs and by the mapping. Finally, Constraints (1d) are indirectly defined the graphs and by the values of L^k and U^k. A collection of packing sets is provided, so the features described in Section 3, that extend the basic formulation, can be used. In this model, the validity of the chosen $\mathcal{B}$ is a clear consequence of Constraints (10b) and of the mapping. However, in other problems, the validity of the packing sets provided by the modeler may not be obvious. All constraints in (10b) are essential. It can be checked that the enumeration sufficient condition is satisfied, so the enumeration procedure can be used.

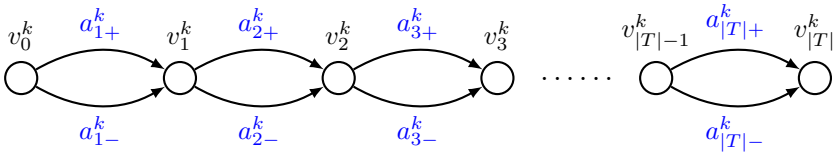

Figure 1: GAP model graph, RCSPs correspond to binary knapsack solutions.

4.2. Vector Packing (VPP) / Bin Packing (BPP)

Data: Set T of items; set D of dimensions; bin capacities Q^d, $d \in D$; item weight w_t^d, $t \in T$, $d \in D$. (Bin packing is the case where $|D| = 1$).

Goal: Find a packing using the minimum number of bins, such that, for each dimension, the total weight of the items in a bin does not exceed its capacity.

Model: A single graph $G = (V, A)$ (we omit the index k in such cases): $V = \{v_t : t = 0, \ldots, |T|\}$, $A = \{a_{t+} = (v_{t-1}, v_t), a_{t-} = (v_{t-1}, v_t) : t = 1, \ldots, |T|\}$, $v_{\text{source}} = v_0$, $v_{\text{sink}} = v_{|T|}$. $R = R^M = D$; $q_{a_{t+},d} = w_t^d, q_{a_{t-},d} = 0, t \in T, d \in D$; $[l_{v_t,d}, u_{v_t,d}] = [0, Q^d]$, $t \in T \cup \{0\}, d \in D$. Continuous variables x_t, $t \in T \cup \{0\}$. The formulation is:

$$\text{Min} \quad x_0 \tag{11a}$$

$$\text{S.t.} \quad x_t = 1, \qquad t \in T; \tag{11b}$$

$L = 0$, $U = \infty$; $M(x_0) = \{a_{1+}, a_{1-}\}$, $M(x_t) = \{a_{t+}\}, t \in T$. $\mathcal{B} = \cup_{t \in T}\{\{a_{t+}\}\}$. Branching over accumulated resource consumption and, if still needed, by Ryan and Foster rule. Enumeration is on.

Comments: Defining the x variables as integer would be useless, it would not be possible to branch over them (except, in very limited way, over x_0). Branching on λ variables is needed; Ryan and Foster rule suffices for correctness, however accumulated resource consumption branching (that may not suffice) should be performed first.

4.3. Capacitated Vehicle Routing Problem (CVRP)

Data: Undirected graph $G' = (V, E)$, $V = \{0, \ldots, n\}$, 0 is the depot, $V_+ = \{1, \ldots, n\}$ are the customers; positive cost c_e, $e \in E$; positive demand d_i, $i \in V_+$; vehicle capacity Q.

Goal: Find a minimum cost set of routes, starting and ending at the depot, visiting all customers and such that the sum of the demands of the customers in a route does not exceed vehicle capacity.

Model: A single graph $G = (V, A)$, $A = \{(i,j), (j,i) : \{i,j\} \in E\}$, $v_{\text{source}} = v_{\text{sink}} = 0$; $R = R_M = \{1\}$; $q_{a,1} = (d_i + d_j)/2$, $a = (i,j) \in A$ (define $d_0 = 0$); $l_{i,1} = 0, u_{i,1} = Q, i \in V$. Integer variables x_e, $e \in E$. The formulation is:

$$\text{Min} \quad \textstyle\sum_{e \in E} c_e x_e \tag{12a}$$

$$\text{S.t.} \quad \textstyle\sum_{e \in \delta(i)} x_e = 2, \qquad i \in V_+, \tag{12b}$$

$$x_e \leq 1, \qquad e \in E \setminus \delta(0); \tag{12c}$$

$L = \lceil \sum_{i=1}^{n} d_i/Q \rceil$, $U = n$; $M(x_e) = \{(i,j), (j,i)\}$, $e = \{i,j\} \in E$. $\mathcal{B}^{\mathcal{V}} = \cup_{i \in V_+} \{\{i\}\}$. RCC separator on $(\cup_{i \in V_+} \{(\{i\}, d_i)\}, Q)$. Branching on x variables. Enumeration is on.

Comments: Constraints (12c) are separated (by inspection) as user cuts. The packing sets are defined on vertices. In this problem, defining the resource consumption in a symmetric way ($q_{(i,j),1} = q_{(j,i),1}$) improves the efficiency of the pricing, as will be discussed in Section 5.1.1. As constraints (12c) are not essential, the enumeration condition over (12b) is satisfied. The function $d : \mathcal{B}^{\mathcal{V}} \cup \emptyset \to \mathbb{R}_+$ for the RCC separator is defined as the set of all pairs $(B, d(B))$ for which $d(B) \neq 0$.

4.4. Heterogeneous Fleet Vehicle Routing Problem (HFVRP)

Data: Undirected graph $G' = (V, E)$, $V = \{0, \ldots, n\}$, 0 is the depot, $V_+ = \{1, \ldots, n\}$ are the customers; positive demand d_i, $i \in V_+$; set of vehicle types $K = \{1, \ldots, m\}$; number of available vehicles u^k, $k \in K$; edge costs c_e^k, $e \in E$, $k \in K$ (assume that fixed costs f^k for using a vehicle of type k are included in the cost of the edges incident to the depot); vehicle type capacity Q^k, $k \in K$.

Goal: Find a minimum cost set of routes, each route associated to a vehicle type and starting and ending at the depot, visiting all customers and such that the sum of the demands of the customers in a route does not exceed its vehicle type capacity. The number of routes for a vehicle type should not exceed its availability.

Model: Graphs $G^k = (V^k, A^k)$, $V^k = \{v_0^k, \ldots, v_n^k\}$, $A^k = \{(v_i^k, v_j^k), (v_j^k, v_i^k) : \{i,j\} \in E\}$, $v_{\text{source}}^k = v_{\text{sink}}^k = v_0^k$, $k \in K$; $R^k = R_M^k = \{r^k\}$; $q_{a,1} = (d_i + d_j)/2$, $a = (v_i^k, v_j^k) \in A^k$, $k \in K$ (define $d_0 = 0$); $l_{v_i^k, r^k} = 0, u_{v_i^k, r^k} = Q^k, v_i^k \in V^k$, $k \in K$. Integer variables x_e^k, $e \in E$, $k \in K$. The

formulation is:

$$\text{Min} \quad \sum_{k\in K}\sum_{e\in E} c_e^k x_e^k \tag{13a}$$

$$\text{S.t.} \quad \sum_{k\in K}\sum_{e\in\delta(i)} x_e^k = 2, \qquad i \in V_+; \tag{13b}$$

$L^k = 0, U^k = u^k$; $M(x_e^k) = \{(v_i^k, v_j^k), (v_j^k, v_i^k)\}$, $e = \{i, j\} \in E$, $k \in K$. $\mathcal{B}^{\mathcal{V}} = \cup_{i\in V_+}\{\{v_i^k : k \in K\}\}$. RCC separator on $(\cup_{i\in V_+}\{(\{v_i^k : k \in K\}, d_i)\}, \max_{k\in K} Q^k)$. Branching first on the aggregation of x variables corresponding to number of times that a vehicle of each type is used, then on the aggregation of x variables corresponding to the assignment of a customer to a vehicle type or on the aggregation of x variables corresponding to the edges of the original graph G'. Enumeration is on.

4.5. Team Orienteering Problem (TOP)

Data: Directed graph $G = (V, A)$, $V = \{0, \ldots, n+1\}$, 0 and $n+1$ are the initial and final depots, respectively, $V_+ = \{1, \ldots, n\}$ are the customers; positive travel time t_a, $a \in A$; profit p_i, $i \in V_+$; maximum route duration T; and fleet size F.

Goal: Find a set of at most F routes, each one starting at 0, ending at $n + 1$ and not exceeding the maximum route duration, that visit each customer at most once and maximize the total profit of the visited customers.

Model: A single graph $G = (V, A)$, $v_{\text{source}} = 0, v_{\text{sink}} = n + 1$; $R = R^M = \{1\}$; $q_{a,1} = t_a$, $a = (i, j) \in A$; $l_{i,1} = 0, u_{i,1} = T, i \in V$. Binary variables x_a, $a \in A$ and y_i, $i \in V_+$. The formulation is:

$$\text{Min} \quad -\sum_{i\in V_+} p_i y_i \tag{14a}$$

$$\text{S.t.} \quad \sum_{i\in\delta^-(i)} x_a = y_i, \qquad i \in V_+; \tag{14b}$$

$L = 0$, $U = F$; $M(x_a) = \{a\}$, $a \in A$; $\mathcal{B}^{\mathcal{V}} = \cup_{i\in V_+}\{\{i\}\}$. Branching on x or on y variables. Enumeration is on.

Comments: The y variables, that indicate which customers are visited, are not mapped to any arc.

4.6. Pickup and Delivery VRPTW (PDPTW)

Data: Directed graph $G = (V, A)$, where $V = \{0\} \cup P \cup D$, $P = \{1, \ldots, n\}$ is the set of pickup vertices and $D = \{n + 1, \ldots, 2n\}$ the set of corresponding deliveries (a pickup at i correspond to a delivery at $i + n$); vehicle capacities Q; traveling cost c_a and time (including service time) t_a , $a \in A$; positive demands d_v, $v \in P$ ($d_v = -d_{v-n}$, $v \in D$); and time windows $[l_v, u_v]$, $v \in V$.

Goal: Find a minimum cost set of routes, starting and ending at the depot, performing all pickups and deliveries within the time windows (waiting is allowed) and such that, along a route, the total demand already collected but not yet delivered does not exceed vehicle capacity.

Model: A single graph $G = (V, A)$, $v_{\text{source}} = v_{\text{sink}} = 0$. $R_M = \{n+2\}$; $R_N = \{1, \ldots, n+1\}$, the first n resources are non-disposable; $q_{(v,v'),v'} = 1$, if $v' \in P$, $q_{(v,v'),v'-n} = -1$, if $v' \in D$, and $q_{(v,v'),n+1} = d_{v'}$, $(v, v') \in A$; $q_{a,n+2} = t_a$, $a \in A$; all other resource consumptions are zero; $u_{v,r} = 1$, $v \in P \cup D$, $r = 1, \ldots, n$, $u_{v,n+1} = Q$, $v \in V$, and $(l_{v,n+2}, u_{v,n+2}) = (l_v, u_v)$, $v \in V$; all other resource bounds are zero. Binary variables x_a, $a \in A$. The formulation is:

$$\text{Min} \quad \textstyle\sum_{a \in A} c_a x_a \tag{15a}$$

$$\text{S.t.} \quad \textstyle\sum_{a \in \delta^-(v)} x_a = 1, \qquad v \in P; \tag{15b}$$

$L = 0$, $U = \infty$; $M(x_a) = \{a\}$, $a \in A$. $\mathcal{B}^{\mathcal{V}} = \cup_{v \in P \cup D} \{\{v\}\}$. Branching first on the aggregation of x variables corresponding to number of routes, then on individual x variables.

Comments: This example illustrates the use of non-monotone secondary resources, some of them being defined as non-disposable. The first n resources are used to enforce that if a route performs a pickup i it can not return to the depot before doing the corresponding delivery $i + n$. If those resources were disposable, routes that visit $i + n$ without having visited first i would be possible. Resource $n + 1$ controls the capacity along the route.

4.7. Capacitated Arc Routing (CARP)

Data: Undirected graph $G' = (V', E)$, $V' = \{0, \ldots, n\}$, 0 is the depot vertex; positive cost c_e and non-negative demand d_e, $e \in E$, set of required edges $S = \{e \in E \,|\, d_e > 0\}$; vehicle capacity Q.

Goal: Find a minimum cost set of routes, closed walks starting and ending at the depot, that serve the demands in all required edges. Edges in a route can be traversed either serving or deadheading (not serving). The sum of the demands of the served edges in a route can not exceed capacity.

Model: For $i, j \in V'$, let $D(i, j) \subseteq E$ be the set of edges in a chosen cheapest path from i to j, with cost $C(i, j) = \sum_{e \in D(i,j)} c_e$. Define a dummy required edge $r_0 = (0, 0')$ and $S_0 = S \cup \{r_0\}$. Define an auxiliary undirected complete graph $G'' = (S_0, F)$. For each $r = (w_1, w_2) \in S_0$, define $o(r, w_1) = w_2$ and $o(r, w_2) = w_1$.

The model has a single RCSP graph generator $G = (V, A)$, $V = \{v_r^w : r \in S_0, w \in r\}$, $A = \{(v_{r_1}^{w_1}, v_{r_2}^{z_1}), (v_{r_1}^{w_1}, v_{r_2}^{z_2}), (v_{r_1}^{w_2}, v_{r_2}^{z_1}), (v_{r_1}^{w_2}, v_{r_2}^{z_2}) : r_1 = (w_1, w_2), r_2 = (z_1, z_2) \in S_0, r_1 \neq r_2\}$, $v_{\text{source}} = v_{r_0}^0$, $v_{\text{sink}} = v_{r_0}^{0'}$; $R = R_M = \{1\}$; for $a = (v_{r_1}^w, v_{r_2}^z) \in A$, $q_{a,1} = d_{r_2}$; $l_{v,1} = 0, u_{v,1} = Q, v \in V$. Binary variables x_a, $a \in A$. For $a = (v_{r_1}^w, v_{r_2}^z) \in A$, $c_a = C(w, o(r_2, z)) + c_{r_2}$. The formulation is:

$$\text{Min} \quad \textstyle\sum_{a \in A} c_a x_a \tag{16a}$$

$$\text{S.t.} \quad \textstyle\sum_{a \in \delta^-(\{v_r^{w_1}, v_r^{w_2}\})} x_a = 1, \qquad r = (w_1, w_2) \in S, \tag{16b}$$

plus Lifted Odd-Cutsets (14; 12); $L = 0$, $U = \infty$; $M(x_a) = \{a\}$, $a \in A$. $\mathcal{B}^{\mathcal{V}} =$

$\cup_{r=(w_1,w_2)\in S}\{\{v_r^{w_1}, v_r^{w_2}\}\}$. RCC separator on $(\cup_{r=(w_1,w_2)\in S}\{(\{v_r^{w_1}, v_r^{w_2}\}, d_r)\}, Q)$. Branching first on the aggregation of x variables corresponding to node degrees in original graph G' or on the aggregation of x variables corresponding edges of graph G''.

Comments: A case where a more complex transformation (similar to (54)) is used to fit the problem into the model. The separation of Lifted Odd-Cutsets is performed using the Gomory-Hu tree algorithm.

5. Implementation

Algorithms used in the implementation of the solver are generalizations of already published algorithms. Thus, we only give an overview of these algorithms; together with references to the original papers. In fact, the main goal of this section is to explain how modeling decisions and solver parameters impact the implemented algorithms.

5.1. Labeling Algorithm for Pricing Problems

Pricing problems are solved by a bi-directional labeling dynamic programming algorithm (71), using the bucket graph implementation proposed in (76). A label is a data structure that represents either a forward partial path started in vertex v^k_{source} or a backward partial path started in vertex v^k_{sink}. Initially, only labels representing null paths at v^k_{source} or at v^k_{sink} exist. Labels are extended along arcs in A^k, if the new extended path is feasible, then the corresponding new label is created. The key feature of a labeling algorithm is the use of dominance checks. Let p and p' be two partial paths ending at the same vertex. If it can be proved that any complete path that is a completion of p' will have a reduced cost greater than the reduced cost of the path obtained by the same completion over p, then p' is dominated by p and the corresponding label can be dropped. Labels are grouped and stored in buckets. The idea is to perform frequent dominance checks only for labels in the same bucket, in order to avoid spending an excessive time on those operations. In our algorithm, labels with the same final vertex and within the same ranges of accumulated consumption values of *main resources* are put in the same bucket. Such bucket organization has advantages over a simpler organization based on resource discretization, used for example in (61): it improves significantly the algorithm performance in the cases where resource intervals are given by very large integer numbers and even allows the use of continuous resource consumptions.

Our implementation supports *at most two main resources*, as we believe that having three or more main resources would result in too many buckets and too few labels per bucket, not being computationally advantageous. Accumulated resource consumption ranges for buckets are specified using the *step size* $\tilde{d}_r$ defined for each main resource $r \in R^k_M$. Each bucket contains labels with a main resource consumption within two consecutive multiples of the step size for this resource. Initial step sizes are defined using parameter ψ^{buck} which designates the maximum number of buckets per vertex. Let $\bar{Q}^k_r$ be the maximum spread of the resource consumption for resource $r \in R^k_M$ in graph G^k: $\bar{Q}_r = u_{v^k_{\text{sink}},r} - l_{v^k_{\text{source}},r}$. Then in the case with one main resource,

$\tilde{d}_1 = \bar{Q}_1/\psi^{\text{buck}}$. In the case two main resources $\tilde{d}_r = \bar{Q}_r/\sqrt{\psi^{\text{buck}}}$ for $r = 1, 2$. In the course of the algorithm, step sizes may be automatically reduced (i.e. number of buckets per vertex may be increased) if it determined that too many dominance checks are performed between labels in a same bucket. If this happens, the buckets are recreated. Binary parameter ψ^{reduc} determines whether automatic step size reduction is activated.

The bucket graph defines one node per bucket and directed arcs connecting pairs of buckets through which the labels can be possibly extended. Distinct bucket graphs are then defined for forward and backward labeling. The concept of bucket graph is useful because:

1. It helps to determine an efficient order of treatment for the buckets. If the bucket graph is acyclic, it is desirable to process the buckets in its topological order because no further extension from a bucket is necessary after it has been processed. If the bucket graph contains cycles, then buckets are handled in the topological order of its strongly connected components, trying to minimize such reprocessing.
2. It is used to improve the efficiency of performing dominance checks between labels in different buckets.
3. Arcs can be removed from the bucket graph by reduced cost arguments, avoiding label extensions in future calls to the pricing. This fixing procedure is more powerful than the one in (47). Binary parameter ϕ^{elimin} determines whether this more sophisticated fixing procedure is applied.

Our labelling algorithm and the bucket arc elimination procedure are bi-directional. For the first main resource, we calculate a special resource consumption value q^* equal to the average middle value of the resource consumption bounds in all vertices. In the forward (backward) labelling, we keep only labels with the first main resource consumption not larger (larger) than q^*. Then the concatenation step is performed to obtain complete paths. Completion bounds are used to speed up concatenation. If the difference between the number of forward and backward non-dominated labels is large, the next call of the pricing will use an automatically adjusted value of q^*. Parameter ϕ^{bidir} defines whether bidirectional search is used. If $\phi^{\text{bidir}} = 0$, bidirectional search is never used. If $\phi^{\text{bidir}} = 1$, bidirectional search is always used. If $\phi^{\text{bidir}} = 2$, bidirectional search is used only for the exact labeling, and not used for the heuristic labeling presented in Section 5.1.3.

5.1.1. Benefiting from symmetry

We exploit forward-backward path symmetry when solving the pricing problem for graph G^k, $k \in K$, if the following conditions are satisfied.

- Packing sets are defined on vertices.
- For each arc $a = (i, j) \in A^k$, $k \in K$, there exists arc $a' = (j, i) \in A^k$ with the same resource consumptions and the same inverse mapping: $q_{a,r} = q_{a',r}$, for all $r \in R^k$, and

$M^{-1}(a) = M^{-1}(a')$. For the sake of symmetry detection, vertices v^k_{source} and v^k_{sink} are treated as the same vertex, even if the user set them as being distinct vertices.

- Resource consumption bounds are the same for all vertices: $[l_{v,r}, u_{v,r}] = [l_{v',r}, u_{v',r}] = [0, Q_r]$, for all $r \in R$ and for all pairs $v, v' \in V^k$.

Under these conditions, value q^* is fixed to $Q_1/2$. The backward labelling is not executed, this saves a significant time. The concatenation is performed just after the forward labelling, using symmetric copies of forward labels as backward labels, as described in (76).

5.1.2. Dominance on disposable and non-disposable resources

When performing dominance checks between labels corresponding to partial paths p and p' ending at the same vertex, we need to compare their accumulated resource comsumptions. Path p can only dominate p' if, for every disposable resource $r \in R_k$, the accumulated consumption of p is not larger than the accumulated consumption of p'. Note that a strictly smaller accumulated consumption does not prevent the dominance, because it is always possible to dispose the additional units of that resource in a completion of p. This is not true for non-disposable resources. Therefore, path p can only dominate p' if, for every non-disposable resource $r \in R_k$, the accumulated consumption of p is identical to the accumulated consumption of p'. This means that models should only define resources as non-disposable if this is really necessary. Remark that monotone and non-monotone resources are not treated differently in the dominance.

A resource is binary if its accumulated consumption can only be 0 or 1. Sets of secondary binary resources of the same type (disposable or not disposable) are implemented in a special way, being represented as bitsets in the labels. This greatly decreases the time spent for dominance checks between labels.

5.1.3. Pricing heuristics

We use two labelling heuristics similar to (76). In the lighter heuristic, only one label per bucket is kept, the one with the smallest reduced costs. In the heavier heuristic, only reduced cost and resource consumption is used to check dominance between labels. Label dimensions related to *ng*-paths and rank-1 cuts are ignored.

5.2. Column and cut generation

We use three-stage column generation. In the first and second stages, the lighter and the heavier labelling heuristics are used, and at most γ^{heur} columns are generated per iteration and per $k \in K$. In the last stage, the exact labeling algorithm is used, and at most γ^{exact} columns are generated per iteration and per $k \in K$. We use automatic dual price smoothing (65) technique to improve the column generation convergence in each stage. Parameter σ^{stab} specifies the minimum column generation stage in which stabilization is active. The column clean-up procedure is also

used: each time the total number of columns exceeds 10,000, only 66% of the columns with the smallest reduced cost remain in the master problem while other columns are removed.

If enabled, the bucket arc elimination procedure for each pricing problem $k \in K$ is performed after the first column generation convergence and each time the current primal-dual gap is reduced by more than 10% since the last call to it. Immediately after this procedure, the bi-directional path enumeration labeling algorithm for graph G^k is executed. This algorithm is a generalization of the one used in (8; 61). It is aborted if the number of non-dominated labels (in this context they correspond to partial $\mathcal{B}$-elementary paths) exceeds ω^{labels} or the number of generated paths exceeds ω^{routes}. If the enumeration algorithm for graph G^k succeeds, i) the enumerated paths are stored in a pool ii) all columns corresponding to paths that are not $\mathcal{B}$-elementary are deleted from the master, iii) in future column generation iterations the pricing is performed by inspection in the pool, and iv) the fixing by reduced costs also starts to be performed in the pool and removes paths from it. If the total number of paths for all graphs drops below ω^{MIP}, all corresponding columns are added to the restricted master, and current node is solved by the MIP solver.

Initial ng-sets may be explicitly given by the user. However, it is possible (and usually much more practical) to specify a distance matrix between packing sets and a parameter η^{init}, which was always equal to 8 for the experiments reported in this paper. If packing sets are defined on arcs, $NG(a)$ will include the η^{init} closest packings sets to the packing set of a according to the distance matrix. If packing sets are defined on vertices, $NG(a)$ will include the union of the η^{init} closest packings sets to the packing sets of both extremities of a.

We have implemented dynamic extension of ng-sets (72; 21), which is useful for some problems. Given a fractional solution $\bar{\lambda}$ obtained after column generation convergence, we augment ng-sets based on "packing set cycles" in paths $p \in P^k$ such that $\bar{\lambda}_p > 0$. A packing set cycle is a partial path started and finished by arcs belonging to a same packing set B. For each considered path, we find all cycles of size at most 5 or, if there are no such cycles, a minimum size cycle. Then we add B to $NG(a)$ for every arc in these cycles. At most 100 paths p with cycles are considered, those with the largest values $\bar{\lambda}_p$. The size of ng-sets is limited by parameter $\eta^{\max}$. All columns corresponding to paths which are not feasible with respect to new ng-sets are deleted from the restricted master.

In each cut round, we generate at most 100 rounded capacity cuts (if RCC separators are defined). We call a rank-1 cut *l-row* if the corresponding vector ρ of multipliers has l positive components. In each round, we generate at most θ^{num} l-row rank-1 cuts for each $l \in \{1, 4, 5, 6, 7\}$, and at most $1.5 \cdot \theta^{\text{num}}$ 3-row rank-1 cuts (which are subset-row cuts (48)). Parameter θ^{rows} determines the maximum number of rows in rank-1 cuts, i.e. if $l > \theta^{\text{rows}}$, l-row rank-1 cuts are not separated. l-row rank-1 cuts are separated by exhaustive enumeration for $l = \{1, 3, 4\}$. For separating other rank-1 cuts, a local search heuristic is used. It is based on the distance matrix between packing sets, that should given by the user. The local search heuristic separates only cuts corresponding to multipliers ρ such that every packing set B, $\rho_B > 0$, is among 16 closest to

any other packing set B', $\rho_{B'} > 0$.

There is parameter θ^{mem} to determine how rank-1 cuts limited memory is computed. If $\theta^{\text{mem}} = 1$, the arc memory is used as described in Section 3.2. If $\theta^{\text{mem}} = 2$, the vertex memory is used, which is "projected" to arcs between vertices in the memory before executing the labeling algorithm. Vertex memory allows the algorithm to converge faster, as rank-1 cuts are stronger. However, the impact on the running time of the labeling algorithm is larger. Thus, the obtained dual bounds may be weaker due to the time thresholds τ^{soft} and τ^{hard} defined below. If $\theta^{\text{mem}} = 0$, the root node is solved two times, first time with arc memory for rank-1 cuts, and the second time with vertex memory. After solving the root, it is automatically determined based on the latest pricing time and the dual bound obtained which memory is used for the remaining of the branch-and-bound tree.

Increasing of ng-sets can also be considered as cut generation, as this procedure improves dual bound given by the master problem. In the root node, ng-set augmentation has the highest priority, robust cuts (RCCs and user cuts) have medium priority, and rank-1 cuts have the smallest priority. We separate cuts with lower priority only if the tailing-off condition is satisfied for cuts of higher priority. The tailing condition is fulfilled when the dual gap is reduced by less than $\delta^{\text{gap}}\%$ after δ^{num} round of cuts. In the other nodes of the branch-and-bound tree, priority of all cuts is the same, i.e. they all are separated in every round.

There are two thresholds τ^{soft} and τ^{hard} for the running time of the exact labeling algorithm. If the running time for at least one pricing problem exceeds τ^{soft}, after the column generation convergence, cuts are not separated and branching is performed. If the running time exceeds τ^{hard}, the labelling algorithm is interrupted and the rollback procedure (61) is executed: the cuts added in the last separation round are deleted from the master, and branching is performed.

5.3. Safe dual bounds

An optimal dual solution returned by the restricted master problem is actually an approximation of a real one, as competitive LP solvers use floating-point arithmetic. Thus the sign of the minimum reduced cost found by the a pricing problem can hardly be decided. This can lead to premature termination or to endless loops (44). For most applications, no practical difficulty occurs. However for some problems, wrong dual bound can be calculated. Vertex coloring and bin packing are examples of such problems.

Our solver has an option to compute a numerically safe dual bound similarly to the method proposed in (44). This approach can be applied in our solver if i) variables y are absent (i.e. $n_2 = 0$), and ii) all coefficients α are non-negative.

We use the property that a valid (Lagrangian) dual bound can be computed for any vector of dual values. Given as a parameter a large integer constant $\tilde{K}$, we call the pricing problem with modified dual values $\lfloor \tilde{K}\pi_i \rfloor$, $i = 1, \dots m$, $\lfloor \tilde{K}\nu_+^k \rfloor$, $\lfloor \tilde{K}\nu_-^k \rfloor$, $k \in K$. The dual values corresponding to the cuts are modified in the same way. The modified cost of variable x_j in the pricing problems

is equal to $\lceil \tilde{K} c_j \rceil$, $j = 1, \ldots, n_1$. This rounding procedure allows us to obtain a feasible dual solution at the end of column generation convergence under two conditions given in the previous paragraph. As the objective function of any modified pricing problem is integer, its solution value can be computed exactly. The (Lagrangian) dual bound is then calculated in the usual way and then multiplied by $\tilde{K}^{-1}$. When the option to use safe dual bounds is activated ($\tilde{K} > 0$), the reduced cost tolerance of the LP solver is set to the minimum possible value (10^{-9} for CPLEX).

5.4. Branching

The user has the possibility to set the priority for each branching strategy he uses for his application. The selection among branching candidates with the same priority is done using a sophisticated hierarchical evaluation strategy similar to the one proposed in (73; 61). The idea is to spend more time evaluating branching candidates in the lowest levels of the branch-and-bound tree where each selection has a greater impact on the overall time, and spend less time as the level increases, taking advantage of the history of previous evaluations. The following three evaluation phases are used:

Phase 0: Up to a half of the candidates are chosen from history using pseudo-costs (if history is not empty). The remaining candidates are chosen in a balanced way between all branching strategies of the current priority. Within the same strategy, the candidates are chosen based on the distance from its fractional value to the closest integer, the larger distance is better.

Phase 1: Evaluate the selected candidates from phase 0 by solving the current restricted master LP modified for each created child node, without generating columns. Select the variables with the maximum value of $\Delta LB_1 \times \Delta LB_2$, where ΔLB_i denotes the increase in the current lower bound obtained for the ith child node, for $i = 1, 2$ (Product Rule, (3)).

Phase 2: Evaluate the selected candidates from phase 1 by solving the relaxation associated to each created child node, including column generation with heuristic labelling for solving the pricing problem. Cut generation is not applied in this phase. The best candidate is also selected by the Product Rule.

The maximum number of candidates evaluated in phase $\rho = 1, 2$ in branch-and-bound node *bbn* is equal to $\min\{\zeta_\rho^{\text{num}}, TS(n^{\text{bb}}) \cdot \zeta_\rho^{\text{estim}}\}$, where $TS(n^{\text{bb}})$ is an estimation of the size of the subtree rooted at the parent of node *bbn* (it is equal to infinity for the root). Calculation of estimation $TS(bbn)$ follows that of (50). See also (52) for a related work.

5.5. Primal heuristics

Restricted master and diving heuristics (77) are built-in to the solver to improve the primal solution during the search. When used, these heuristics are executed at each branch-and-bound node before branching. The first heuristic uses a MIP solver to solve the current restricted master problem as a mixed-integer program with the time limit χ^{rm}. The second heuristic is the diving

heuristic with Limited Discrepancy Search (LDS). It is applied in branch-and-bound nodes of depth at most χ^{div}. The diving heuristic uses two parameters χ^{disc} and χ^{depth} which correspond to parameters *maxDiscrepancy* and *maxDepth* in (77). In each dive of the diving heuristic, we iteratively fix a column with largest value in the fractional solution to one and then we solve the resulting residual master problem with column generation. In the diving heuristic with LDS, one does several dives. Thus, a search tree is formed in which backtracking is allowed. In each node n^{div} of the diving search tree, we keep a tabu list of columns forbidden to be fixed. Tabu list of a child node is initialized with the current tabu list of the parent node (the initial tabu list at the root is empty). After a backtrack to a node n^{div}, we insert in its tabu list the column which was fixed by the previous fixing decision in n^{div}. An additional fixing decision in n^{div} is allowed (i.e. other child nodes of n^{div} can be created) if the size of its tabu list does not exceed χ^{disc} and the depth of n^{div} is not larger than χ^{depth}.

5.6. Parameterization

Based on the description above, we list in Table 1 the solver parameters available to the user. In addition the user may provide the following information to improve the solver performance.

- Designation of the first main resource which will be used for the bi-directional labelling.
- Priorities for branching strategies.
- Distance matrix between packing sets for defining initial ng-sets and for local search heuristic separation of l-row rank-1 cuts with $l \geq 5$.

Description	Notation	Default value(s)
Time thresholds for the labeling algorithm	τ^{soft}, τ^{hard}	10 sec., 20 sec.
Calculation of step sizes for buckets	ψ^{buck}, ψ^{reduc}	25, 1 (on)
Bi-directional search and bucket arc elimination	ϕ^{bidir}, ϕ^{elim}	2, 1 (on)
Max. # of generated columns per iteration	γ^{exact}, γ^{heur}	150, 30
Minimum stage for stabilization	σ^{stab}	0 (everywhere)
Max. # of labels and paths in the enumeration	ω^{labels}, ω^{routes}	10^6, 10^6
Max. total # of enumerated paths for MIP	ω^{MIP}	10^4
Initial and maximum size of ng-sets	η^{init}, η^{max}	8, 8
Limited-memory rank-1 cuts parameters	θ^{num}, θ^{rows}, θ^{mem}	100, 5, 2
Cut generation tailing-off parameters	δ^{gap}, δ^{num}	2%, 3
Numerically safe dual bound multiplier	$\tilde{K}$	-1 (off)
Strong branching parameters for phase 1	ζ_1^{num}, ζ_1^{estim}	100, 0.3
Strong branching parameters for phase 2	ζ_2^{num}, ζ_2^{estim}	3, 0.1
Restricted master heuristic	χ^{rm}	-1 (off)
Diving heuristic (with LDS)	χ^{div}, χ^{depth}, χ^{disc}	-1 (off), 0, 0

Table 1: Parameters of the solver available to the user and their default values

6. Computational Experiments

The generic BCP solver optimization algorithms were coded in C++. The interface to the solver is implemented in Julia 0.6 language using JuMP (34) and LightGraphs packages. We also used:

- BaPCod C++ library (81) which implements the BCP framework;
- the C++ code developed by Sadykov et al. (76) implementing the bucket graph based labeling algorithm, bucket arc elimination procedure, path enumeration, and the separation of limited-memory rank-1 cuts;
- CVRPSEP C++ library (55) which implements heuristic separation of rounded capacity cuts;
- IBM CPLEX Optimizer version 12.8.0 as the LP solver in column generation and as the solver for the enumerated MIPs.
- Boost C++ library version 1.55 (2)
- LEMON C++ library version 1.31 (1) for the bucket graph creation for the labeling algorithm.

The experiments were run on a 2 Deca-core Ivy-Bridge Haswell Intel Xeon E5-2680 v3 server running at 2.50 GHz. The 128 GB of available RAM was shared between 8 copies of the algorithm running in parallel on the server. Each instance is solved by one copy of the algorithm using a single thread.

In Table 2, we show computational results for 13 applications. The first column is the problem acronym, second column refers to data sets, the third indicates the number of instances. Next is the time limit per instance. The last three columns show the results obtained by our generic solver, as well as by two other algorithms, those with the best (to our knowledge) published results for the data set. For each algorithm, we give the number of instances solved within the time limit, the average time in brackets (geometric mean time if the time limit is 10 hours or more), and its reference. For instances not solved, the time limit is considered as the solution time. Best results are marked in bold. The performance of our solver depends significantly on initial primal bounds given by the user. In the experiments, we always used the same primal bounds as in the works we compare with.

For each problem below we give details concerning the models and the parameterization employed, instances considered, initial primal bounds used, as well as analysis of computational results.

Problem	Data set	#	T.L.	Gen. BCP	Best Publ.	2nd Best
CVRP	E-M (24; 25)	12	10h	12 (61s)	**12 (49s)** (61)	10 (432s) (27)
	X (80)	58	60h	**36 (147m)**	34 (209m) (80)	—
VRPTW	Solomon Hardest (79)	14	1h	**14 (5m)**	13 (17m) (60)	9 (39m) (10)
	Homberger 200 (40)	60	30h	**56 (21m)**	50 (70m) (60)	7 (-) (49)
HFVRP	BaldacciMingozzi (9)	40	1h	**40 (144s)**	39 (287s) (64)	34 (855s) (9)
MDVRP	Cordeau (28)	11	1h	**11 (6m)**	11 (7m) (64)	9 (25m) (27)
PDPTW	RopkeCordeau (74)	40	1h	**40 (5m)**	33 (17m) (42)	32 (14m) (7)
	LiLim (53)	30	1h	3 (56m)	**23 (20m)** (7)	18 (27m) (42)
TOP	Chao class 4 (23)	60	1h	**55 (8m)**	39 (15m) (16)	30 (-) (37)
CTOP	Archetti (5)	14	1h	**13 (7m)**	6 (35m) (4)	7 (34m) (5)
CPTP	Archetti open (5)	28	1h	**24 (9m)**	0 (1h) (19)	0 (1h) (4)
VRPSL	Bulhoes (19)	180	2h	**159 (16m)**	49 (90m) (19)	—
GAP	OR-Lib, type D (13)	6	2h	5 (40m)	**5 (30m)** (70)	5 (46m) (6)
	Nauss (57)	30	1h	**25 (23m)**	1 (58m) (43)	0 (1h) (57)
BPP	Falkenauer T (38)	80	10m	80 (16s)	**80 (1s)** (18)	80 (1s) (30)
	Hard28 (78)	28	10m	28 (17s)	**28 (4s)** (30)	28 (7s) (15)
	AI (31)	250	1h	**160 (25m)**	140 (28m) (83)	116 (35m) (15)
	ANI (31)	250	1h	**103 (35m)**	97 (40m) (83)	67 (45m) (30)
VPP	Classes 1,4,5,9 (22)	40	1h	**38 (8m)**	13 (50m) (45)	10 (53m) (18)
CARP	Eglese (36)	24	30h	**22 (36m)**	22 (43m) (63)	10 (237m) (12)

Table 2: Generic solver vs best specific solvers on 13 problems.

Capacitated Vehicle Routing Problem (CVRP)

The model is given in Section 4.3. The parameterization of the solver is the following (values different from defaults): $\tau^{\text{hard}} = 25$ sec., $\omega^{\text{routes}} = 5 \cdot 10^6$, $\theta^{\text{mem}} = 0$, $\zeta_1^{\text{estim}} = 0.2$, $\zeta_2^{\text{num}} = 5$, $\zeta_2^{\text{estim}} = 0.02$, $\delta^{\text{gap}} = 1.5\%$. The distance matrix between packing sets corresponds to the distance matrix between clients. The same matrix is used in all routing problems below, except CARP.

The considered E-M instances are the 12 hardest ones, those considered in (61). The considered X instances are those with less than 400 customers. The number of vehicles for E-M instances is fixed as it is usual in the literature. The number of vehicles is unbounded for X instances. The same initial upper bounds are used as in the literature (61; 80).

The results show that our solver outperforms noticeably state-of-the-art on X instances. Results for E-M instances are comparable to the state-of-the-art. Note that the initial upper bound for the hardest instance M-n200-k16 is different by 4 units from the optimum solution. This difference introduce randomness to the running time of algorithms.

According to CVRPLIB (`http://vrp.atd-lab.inf.puc-rio.br`), in November 2018 there were 55 open CVRP instances in the X set (80). We started long runs of the generic solver on the most promising ones, using a specially calibrated parameterization. We could solve 6 instances

to optimality for the first time, as indicated in Table 3. Improved best known solutions are underlined.

Instance	Prev. BKS	Root LB	Nodes	Total Time	OPT
X-n284-k15	20226	20168	940	11.0 days	**20215**
X-n322-k28	29834	29731	1197	5.6 days	**29834**
X-n344-k43	42056	41939	2791	11.6 days	**42050**
X-n393-k38	38260	38194	1331	5.8 days	**38260**
X-n469-k138	221909	221585	8964	15.2 days	**221824**
X-n548-k50	86701	86650	337	2.0 days	**86700**

Table 3: Detailed results on the open instances solved.

Vehicle Routing Problem with Time Windows (VRPTW)

We use the same model as for the CVRP except that an additional main resource is defined which represents the time. Different bounds on the consumption of the time resource prevent us from exploiting forward-backward path symmetry in the pricing.

The considered Solomon instances (all with 100 customers) are the hardest ones according to (60). As it is standard in the literature, we divide the instances in two groups. First group contains instances of type C1, RC1, and R1. For this group of instances, the first main resource for the bi-directional labelling is the capacity resource. Second group contains instances of type C2, RC2, R2. For the second group of instances, capacity resource is not used, and the feasibility related to the vehicle capacity is guaranteed by the rounded capacity cuts, separated both for fractional and integer solutions of the master problem. Therefore, those cuts becomes essential constraints and the sufficient condition for enumeration given in Section 3.3 is not verified (paths visiting the same customers in distinct orders may have different coefficients in some RCC). However, it is still possible to show that the enumeration procedure is valid. The same initial upper bounds are used as for the algorithm in (60).

The parameterization of the solver for the first group of instances is $\tau^{\text{soft}} = 5$ sec., $\tau^{\text{hard}} = 10$ sec., $\phi^{\text{bidir}} = 1$, $\eta^{\max} = 16$, $\theta^{\text{rows}} = 7$, $\theta^{\text{mem}} = 1$, $\omega^{\text{routes}} = 10^7$, $\zeta_0^{\text{num}} = 50$, $\zeta_1^{\text{estim}} = 0.1$, $\zeta_2^{\text{estim}} = 0.1$, $\delta^{\text{gap}} = 1.5\%$. The parameterization for the second group of instances is $\tau^{\text{hard}} = 30$ sec., $\phi^{\text{bidir}} = 1$, $\eta^{\max} = 16$, $\theta^{\text{mem}} = 1$, $\omega^{\text{routes}} = 5 \cdot 10^6$, $\zeta_1^{\text{num}} = 50$, $\zeta_1^{\text{estim}} = \infty$, $\zeta_2^{\text{estim}} = \infty$, $\delta^{\text{gap}} = 1.5\%$.

The results in Table 2 show that our solver outperformed significantly the algorithm proposed in (60). It is more than three times faster on average and could solve 7 more instances within the time limit. All Solomon instances with 100 customers are solved in less than 5 minutes each, except the hardest instance R208 which is solved in 37 minutes. The latter was solved in about 17 hours by the algorithm in (60).

Heterogeneous Fleet Vehicle Routing Problem (HFVRP)

The model is given in Section 4.4. The parameterization of the solver is the following: $\tau^{\text{soft}} = 6$ sec., $\tau^{\text{hard}} = 15$ sec., $\omega^{\text{routes}} = 5 \cdot 10^6$, $\theta^{\text{mem}} = 1$, $\zeta_1^{\text{estim}} = 0.2$, $\zeta_2^{\text{num}} = 5$, $\zeta_2^{\text{estim}} = 0.02$. The branching on the number of paths for each vehicle type has the highest priority. Other branching strategies have the same priority.

We use the instances with 50-100 clients proposed in (9). They include instances with limited and unlimited fleet, with and without fixed vehicle cost, and with vehicle dependent and independent routing costs. The same initial upper bounds are used as in (64). Our solver is significantly outperforms the algorithm from (9). It also has a better performance than a recent branch-cut-and-price algorithm specific for that problem (64).

Multi-Depot Vehicle Routing Problem (MDVRP)

We use a model similar to the one used on HFVRP, defining a graph G^k for every depot $k \in K$. Variables associated with edges incident to a depot k are mapped to the corresponding arcs in graph G^k. Variables associated with edges between customers are mapped to corresponding arcs in all graphs. The same parameterization and the same priorities for branching as for the HFVRP are used.

We use instances proposed in (28). Only instances without distance constraints are considered, so there is a single capacity resource. The same initial upper bounds are used as in (64). The solver obtained a performance similar to (64) and significantly better than (27).

Pickup and Delivery Problem with Time Windows (PDPTW)

The model is given in Section 4.6. The parameterization of the solver is the following: $\tau^{\text{soft}} = 5$ sec., $\tau^{\text{hard}} = 10$ sec., $\psi^{\text{buck}} = 200$, $\omega^{\text{labels}} = 5 \cdot 10^5$, $\omega^{\text{routes}} = 2.5 \cdot 10^6$, $\omega^{\text{MIP}} = 7000$, $\delta^{\text{num}} = 2$, $\zeta_1^{\text{estim}} = 30$, $\zeta_1^{\text{estim}} = 1.0$, $\zeta_2^{\text{num}} = 2$. Diving heuristic is used with parameters $\chi^{\text{div}} = 10$, $\chi^{\text{depth}} = \infty$, $\chi^{\text{disc}} = 1$.

The convention in PDPTW literature is to first minimize the number of routes, then the minimize the transportation cost. So, we add the constant 10,000 to the cost of arcs leaving the depot. The same initial upper bounds are used as in (7). Results for this problem are mixed. Worse performance for Li& Lim instances can be explained by the fact that the our solver does not incorporate some labeling algorithm acceleration techniques specific to the PDPTW. For the Ropke & Cordeau instances however, generic state-of-the-art BCP elements mitigate the effect of lacking ad-hoc enhancements. Employing diving heuristic is important here as initial upper bounds are not tight for some instances.

Team Orienteering Problem (TOP)

The model is given in Section 4.5. The parameterization of the solver is the following: $\tau^{\text{soft}} = 5$ sec., $\tau^{\text{hard}} = 10$ sec., $\psi^{\text{buck}} = 200$, $\phi^{\text{bidir}} = 1$, $\omega^{\text{labels}} = 5{\cdot}10^5$, $\omega^{\text{routes}} = 2.5{\cdot}10^6$, $\theta^{\text{mem}} = 1$, $\delta^{\text{num}} = 2$,

$\zeta_1^{\text{estim}} = 50$, $\zeta_1^{\text{estim}} = 1.0$. Both restricted master and diving heuristics are used with parameters $\chi^{\text{rm}} = 1$ sec., $\chi^{\text{div}} = 0$, $\chi^{\text{depth}} = \infty$, $\chi^{\text{disc}} = 1$.

No initial upper bound is defined, so using heuristics is important. Instances of class 4, the most difficult one according to (16), are considered for TOP. Our solver clearly outperforms the state-of-the-art on this problem. Its advantage can be explained by the fact that some important recent improvements in BCP algorithms for variants like CVRP and VRPTW were not yet adapted and used in TOP, possibly due to the complexity of their implementation.

Capacitated Team Orienteering Problem (CTOP)

The same model as for the TOP is used, except that an additional capacity resource is considered. The first main resource for the bi-directional labelling is the time resource. Pricing problem can be difficult for the instances considered. Therefore, we used a special parameterization for the number of buckets to be able to solve them in a reasonable time. Bucket arc elimination procedure and thus route enumeration are not used due to performance issues. Rank-1 cuts are also not separated due to the difficulty of the pricing problem.

The parameterization of the solver is the following: $\tau^{\text{soft}} = 5$ sec., $\tau^{\text{hard}} = 10$ sec., $\psi^{\text{buck}} = 1000$, $\phi^{\text{bidir}} = 1$, $\phi^{\text{elim}} = 0$ (off), $\sigma^{\text{stab}} = 1$, $\eta^{\max} = 30$, $\theta^{\text{rows}} = -1$ (off), $\delta^{\text{gap}} = 1.5\%$, $\zeta_1^{\text{estim}} = 50$, $\zeta_1^{\text{estim}} = 1.0$. Diving heuristic is used with parameters $\chi^{\text{div}} = 0$, $\chi^{\text{depth}} = \infty$, $\chi^{\text{disc}} = 1$.

No initial upper bound is defined. We used all instances in Set 1 from (5). We also used only open instances in Set 2 from (5) (which were not solved by Archetti et al. (4)), as Set 2 contains easier instances (with a reduced vehicle capacity and a reduced time limit). We have not solved only one instance: "p09". All other instances were solved to optimality in less than 15 minutes each, seven of them for the first time.

Capacitated Profitable Tour Problem (CPTP)

The same model as for the CTOP, except that i) the only resource is the vehicle capacity, ii) the objective is the difference between the total profit and the transportation cost. The parameterization of the solver is the following: $\tau^{\text{soft}} = 5$ sec., $\tau^{\text{hard}} = 10$ sec., $\psi^{\text{buck}} = 200$, $\phi^{\text{bidir}} = 1$, $\sigma^{\text{stab}} = 1$, $\omega^{\text{labels}} = 5 \cdot 10^5$, $\omega^{\text{routes}} = 2.5 \cdot 10^6$, $\eta^{\max} = 30$, $\theta^{\text{mem}} = 1$, $\delta^{\text{gap}} = 1.5\%$, $\zeta_1^{\text{estim}} = 50$, $\zeta_1^{\text{estim}} = 1.0$.

Only open instances from (5) are considered, which could not be solved by Bulhoes et al. (19). The same initial upper bounds were used as for the branch-and-price algorithm in (19). Note that only one of them was improved: for instance "p13-4-200" the optimum value is 304.15, whereas the best known solution is 303.18. So the heuristic suggested in (19) is of a very good quality. Among 28 open instances of the CPTP, we solved to optimality 24 within 1 hour. Two more instances "p13-4-200" and "p10-20-200" were solved within 2 hours each. The only remaining open instances are now "p15-15-200", and "p16-20-200".

VRP with Service Level constraints (VRPSL)

VRPSL is a generalisation of CVRP in which a service weight is defined for each customer. For each predefined group of customers, total service weight of visited customers should not be below a threshold. The model contains edge and y variables as for the TOP. For each group, a knapsack constraint over y variables is defined. Branching is both on y and edge variables. The parameterization of the solver is the following: $\tau^{\text{soft}} = 3$ sec., $\tau^{\text{hard}} = 6$ sec., $\psi^{\text{buck}} = 200$, $\omega^{\text{labels}} = 2 \cdot 10^5$, $\eta^{\max} = 30$, $\theta^{\text{rows}} = 4$, $\theta^{\text{mem}} = 1$, $\delta^{\text{gap}} = 1.5\%$, $\zeta_1^{\text{num}} = 50$, $\zeta_1^{\text{estim}} = 1.0$.

The instances proposed in (19) are considered. The same initial upper bounds were used as in (19). None of them was improved. Our solver outperformed largely the branch-and-price algorithm by Bulhoes et al. (19). The reason is that the latter does not use many state-of-the-art techniques for routing problems. Performance of our solver can probably be improved by separating valid inequalities for the knapsack constraints.

Generalized Assignment Problem (GAP)

The model is given in Section 4.1. We used 6 classic OR-Library instances of the most difficult type D with up to 20 machines and 200 tasks. Also we used instances by Nauss (57) with $|T| = 90, 100$ and $|K| = 25, 30$. For the classic instances, the parameterization of the solver is the following: $\psi^{\text{buck}} = 200$, $\psi^{\text{reduc}} = 0$ (off), $\phi^{\text{bidir}} = 1$, $\gamma^{\text{exact}} = 10$, $\gamma^{\text{heur}} = 10$, $\omega^{\text{MIP}} = 4000$, $\eta^{\text{init}} = 0$, $\eta^{\max} = 0$, $\theta^{\text{rows}} = 3$. For Nauss instances, the parameterization of the solver is the following: $\psi^{\text{buck}} = 100$, $\psi^{\text{reduc}} = 0$ (off), $\phi^{\text{bidir}} = 1$, $\gamma^{\text{exact}} = 10$, $\gamma^{\text{heur}} = 10$, $\omega^{\text{labels}} = 10^5$, $\omega^{\text{routes}} = 2 \cdot 10^5$, $\omega^{\text{MIP}} = 4000$, $\eta^{\text{init}} = 0$, $\eta^{\max} = 0$, $\theta^{\text{rows}} = 4$.

For OR-Library instances, we took best known solution values augmented by 1 as initial upper bounds. Performance of our solver is comparable to the state-of-the-art. Both "competitors" (70) and (6) solved the same number of instances to optimality in similar time. In Table 2, times of (70) and (6) are adjusted according to computer speeds. Note that our solver is the first algorithm in the literature which solves the pricing problem as the resource constrained shortest path problem. Although it is slower than specialised knapsack solvers, it supports rank-1 cuts and enumeration.

Initial bounds for Nauss instances were calculated by us using problem specific strong diving heuristic from (77). This heuristic is based on classic column generation for the GAP, in which the pricing problem is solved by the knapsack solver (68). The heuristic time is included in the reported time. Our solver is much more efficient than the algorithm by Nauss (57) and the MIP formulation for the GAP solved by Gurobi. Note that our solver obtains particularly good results for instances with relatively small number of tasks per machine, as in this case path enumeration procedure is very efficient. For instances with large number of tasks per machine our solver is less efficient. In particular, a more advanced stabilisation technique is required for such instances, as discussed in (65).

Bin Packing Problem (BPP)

The model is given in Section 4.2. The parameterization of the solver is the following: $\psi^{\text{buck}} = 200$, $\psi^{\text{reduc}} = 0$ (off), $\phi^{\text{bidir}} = 1$, $\gamma^{\text{exact}} = 100$, $\gamma^{\text{heur}} = 100$, $\sigma^{\text{stab}} = 2$, $\omega^{\text{routes}} = 2 \cdot 10^6$, $\omega^{\text{MIP}} = 10^5$, $\eta^{\text{init}} = 0$, $\eta^{\text{max}} = 0$, $\theta^{\text{rows}} = 4$, $\delta^{\text{num}} = 5$, $\tilde{K} = 10^{10}$, $\zeta_1^{\text{num}} = 20$, $\zeta_2^{\text{num}} = 1$. Diving heuristic with parameters $\chi^{\text{div}} = 10$, $\chi^{\text{depth}} = 0$, $\chi^{\text{disc}} = 0$ is used for all instances except set ANI.

We give smaller priority for the Ryan and Foster branching and larger priority for the branching over the accumulated resource consumption. The latter showed to be so effective that Ryan and Foster rule was never used in our experiments. For each instance, we use initial primal bound which is equal to the rounded up value of the column generation dual bound plus 1 unit (there is a long-standing conjecture that the optimal solution of a BPP instance is never larger than this). Solutions with these objective values are easily obtainable by simple heuristics.

Our solver obtains the best results for the two most difficult instance classes AI and ANI. For other less difficult instances, our algorithm is slower than the state-of-the-art. However, it can solve all instances to optimality in a relatively small time. Note that the algorithm of Clautiaux et al. (26) showed better results for the class AI. However, the authors communicated to us that they discovered an issue with their code. Therefore, their results are not included in Table 2.

The bottleneck of our algorithm for solving instances of classes AI and ANI with 600 items or more is the LP solver. For such instances, very often when the pricing problem finds columns with negative reduced cost, LP solver does not include them in the basis. This happens because the absolute value of the reduced cost is smaller than the minimum possible reduced cost tolerance of the LP solver. Thus, the final dual solution of column generation is not feasible, and the Lagrangian bound obtained by the safe procedure is weaker than it can potentially be.

Vector Packing Problem (VPP)

The model is given in Section 4.2. The parameterization of the solver is the following: $\psi^{\text{buck}} = 2000$, $\psi^{\text{reduc}} = 0$ (off), $\phi^{\text{bidir}} = 1$, $\gamma^{\text{exact}} = 100$, $\gamma^{\text{heur}} = 100$, $\sigma^{\text{stab}} = 1$, $\omega^{\text{labels}} = 10^5$, $\omega^{\text{MIP}} = 10^5$, $\eta^{\text{init}} = 0$, $\eta^{\text{max}} = 0$, $\theta^{\text{rows}} = -1$ (off), $\zeta_2^{\text{num}} = 1$. We use diving heuristic with parameters $\chi^{\text{div}} = 0$, $\chi^{\text{depth}} = 2$, $\chi^{\text{disc}} = 3$.

For the VPP, we use an additional pricing heuristic in which at most 8 labels with the best reduced cost are kept in each bucket. This heuristic helps us to solve some difficult instances. This happens because exact pricing solution time can be very different in two cases: i) when there are negative reduced cost paths to be found, and ii) when there are no such paths. Very rarely, the pricing solution time can be orders of magnitude larger in the first case, even if these two cases occur in consecutive column generation iterations. This behaviour of the labelling algorithm should be further investigated.

We do not use initial upper bounds. We consider only largest instances from the literature with 200 items and only with 2 resources. We use instances of the most difficult, according to (45), classes 1, 4, 5, and 9. Other instances are significantly easier, all of them are solved in the

literature. Our solver outperforms largely the algorithms in the literature. Only two instances were not solved to optimality. These are instances 09_200_05 and 09_200_06. Other instances are solved at the root node. 32 from 40 instances are solved in less than 5 minutes.

Capacitated Arc Routing Problem (CARP)

The model is defined in Section 4.7. The parameterization of the solver is the following: $\tau^{\text{soft}} = 6$ sec., $\tau^{\text{hard}} = 15$ sec., $\psi^{\text{buck}} = 50$, $\gamma^{\text{exact}} = 50$, $\gamma^{\text{heur}} = 300$, $\sigma^{\text{stab}} = 1$, $\omega^{\text{labels}} = 5 \cdot 10^5$, $\theta^{\text{num}} = 50$, $\delta^{\text{gap}} = 1\%$, $\delta^{\text{num}} = 5$, $\zeta_1^{\text{estim}} = 1.0$, $\zeta_2^{\text{num}} = 5$. In the distance matrix for ng-sets, the distance between two required edges is defined as the sum of costs of four paths between midpoints of the edges. Each path is the shortest path starting from a given vertex incident to the first edge and ending at a given vertex incident to the second edge.

The branching is done on the aggregation of x variables corresponding to node degrees in the original graph or on the aggregation of x variables corresponding to whether required two edges are served immediately one after another by the same route or not. The same priority is used for both branching strategies.

The Eglese dataset (36) is standard in the literature and it is used in all recent works on the CARP. We have used the same initial upper bounds as in (63). The performance of our solver is similar to the most recent exact algorithm (63) for the problem. Other algorithms in the literature are significantly less efficient. The generality of our solver opens a way to quickly obtain excellent computational results for many variants of the arc routing problems.

7. Conclusions

We proposed a new generic way of modeling VRPs and related problems, so that they can be solved by a BCP algorithm that already includes most state-of-the-art elements introduced for the most classical VRP variants. It combines existing modeling concepts, like the use of RCSPs for defining the valid routes, with new ones, the most important being packing sets. The experiments show that the generic solver has a performance either comparable or better than the specific algorithms for all VRP variants tested. The cases where the performance was much better can be explained by the fact that previous works on some problems did not use some of those advanced elements, possibly due to the complexity of their implementation. However, if generic VRP solvers become publicly available, we believe that their use may become standard, at least for the purpose of having baseline results to be compared with results of proposed specialized algorithms.

The presented generic solver is available for academic use at `https://vrpsolver.math.u-bordeaux.fr/`. The optimization algorithms and a Julia–JuMP (35) user interface are being given as pre-compiled docker image. Modeling a typical VRP variant, like those in our tests, requires around 100 lines of Julia code (not counting input/output code). This means that a user can already have a good working algorithm in a day. After that, several days of

computational experiments for parameter tuning may yield an improved performance. However, there are variants where additional work on separation routines for problem specific cuts may be needed for top performance.

Furthermore, we believe that there is plenty of room for "creative modeling", where users may find original ways of fitting new problems into the proposed model. In fact, as already demonstrated on generalized assignment problem and on bin/vector packing problems, not only VRP variants can be efficiently treated. It may be also possible to model of problems from scheduling, from network design and from other discrete optimization subareas. As the VRP technology available in the solver is quite advanced, there is a chance of obtaining a good performance.

As future work, we plan to further extend the modeling capabilities of the VRP solver. We believe that the most promising course for that is to add the possibility of using other types of resources in the models. This may include resources with arc consumption dependent on its own accumulated consumption or even dependent on accumulated consumption of other resources, resources with soft or multiple interval limits, non-linear and stochastic resources, and others, as discussed in (46) and in (58). However, devising and implementing algorithms that support any of those more complex resources, in an efficient way and preserving the compatibility with all the existing features of our solver, will be a major challenge.

Acknowledgements

We would like to thank Teobaldo Bulhoes and Guillaume Marques for a large part of the implementation of the Julia–JuMP interface to the solver; Teobaldo Bulhoes, Guillaume Marques and Eduardo Queiroga for implementing, over that interface, the models corresponding to the examples of this paper; and Laurent Facq for a general support of the computing environment.

Experiments presented in this paper were carried out using the PlaFRIM (Federative Platform for Research in Computer Science and Mathematics), created under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux, CNRS and ANR in accordance to the "Programme d'Investissements d'Avenir".

References

[1] LEMON: Library for Efficient Modeling and Optimization in Networks, 2014.

[2] Boost C++ libraries, 2019.

[3] Tobias Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universitat Berlin, 2007.

[4] C. Archetti, N. Bianchessi, and M.G. Speranza. Optimal solutions for routing problems with profits. *Discrete Applied Mathematics*, 161(4–5):547–557, 2013.

[5] C. Archetti, D. Feillet, A. Hertz, and M G Speranza. The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society*, 60(6):831–842, Jun 2009.

[6] Pasquale Avella, Maurizio Boccia, and Igor Vasilyev. A computational study of exact knapsack separation for the generalized assignment problem. *Computational Optimization and Applications*, 45(3):543–555, 2010.

[7] Roberto Baldacci, Enrico Bartolini, and Aristide Mingozzi. An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, 59(2):414–426, 2011.

[8] Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115:351–385, 2008.

[9] Roberto Baldacci and Aristide Mingozzi. A unified exact method for solving different classes of vehicle routing problems. *Mathematical Programming*, 120(2):347–380, 2009.

[10] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011.

[11] M.L. Balinski and R.E. Quandt. On an integer program for a delivery problem. *Operations Research*, 12(2):300–304, 1964.

[12] Enrico Bartolini, Jean-François Cordeau, and Gilbert Laporte. Improved lower bounds and exact algorithm for the capacitated arc routing problem. *Mathematical Programming*, 137(1):409–452, Feb 2013.

[13] J. E. Beasley. OR-Library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[14] J. Belenguer and E. Benavent. The capacitated arc routing problem: Valid inequalities and facets. *Computational Optimization & Applications*, 10(2):165–187, 1998.

[15] G. Belov and G. Scheithauer. A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting. *European Journal of Operational Research*, 171(1):85 – 106, 2006.

[16] Nicola Bianchessi, Renata Mansini, and M. Grazia Speranza. A branch-and-cut algorithm for the team orienteering problem. *International Transactions in Operational Research*, 25(2):627–635, 2018.

[17] C. Bode and S. Irnich. Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research*, 60(5):1167–1182, 2012.

[18] Filipe Brandão and João Pedro Pedroso. Bin packing and related problems: General arc-flow formulation with graph compression. *Computers & Operations Research*, 69:56 – 67, 2016.

[19] Teobaldo Bulhoes, Minh Hoàng Hà, Rafael Martinelli, and Thibaut Vidal. The vehicle routing problem with service level constraints. *European Journal of Operational Research*, 265(2):544 – 558, 2018.

[20] Teobaldo Bulhões, Artur Pessoa, Fábio Protti, and Eduardo Uchoa. On the complete set packing and set partitioning polytopes: Properties and rank 1 facets. *Operations Research Letters*, 46(4):389–392, 2018.

[21] Teobaldo Bulhoes, Ruslan Sadykov, and Eduardo Uchoa. A branch-and-price algorithm for the minimum latency problem. *Computers & Operations Research*, 93:66–78, May 2018.

[22] Alberto Caprara and Paolo Toth. Lower bounds and algorithms for the 2-dimensional vector packing problem. *Discrete Applied Mathematics*, 111(3):231 – 262, 2001.

[23] I-Ming Chao, Bruce L. Golden, and Edward A. Wasil. The team orienteering problem. *European Journal of Operational Research*, 88(3):464 – 474, 1996.

[24] N. Christofides and S. Eilon. An algorithm for the vehicle-dispatching problem. *Operational Research Quarterly*, 20:309–318, 1969.

[25] N. Christofides, A. Mingozzi, and P. Toth. *Combinatorial Optimization*, chapter The vehicle routing problem, pages 315–338. Wiley, Chichester, 1979.

[26] François Clautiaux, Saïd Hanafi, Rita Macedo, Marie Émilie Voge, and Cláudio Alves. Iterative aggregation and disaggregation algorithm for pseudo-polynomial network flow models with side constraints. *European Journal of Operational Research*, 258(2):467 – 477, 2017.

[27] Claudio Contardo and Rafael Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129 – 146, 2014.

[28] Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30(2):105–119, 1997.

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

[30] Maxence Delorme and Manuel Iori. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, Forthcoming, 2018.

[31] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, 2016.

[32] Guy Desaulniers, Jacques Desrosiers, Marius M Solomon, François Soumis, Daniel Villeneuve, et al. A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In *Fleet management and logistics*, pages 57–93. Springer, 1998.

[33] Guy Desaulniers, François Lessard, and Ahmed Hadjar. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404, 2008.

[34] I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[35] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[36] R. W. Eglese and L. Y. O. Li. Efficient routeing for winter gritting. *Journal of the Operational Research Society*, 43(11):1031–1034, 1992.

[37] Racha El-Hajj, Duc-Cuong Dang, and Aziz Moukrim. Solving the team orienteering problem with cutting planes. *Computers & Operations Research*, 74:21 – 30, 2016.

[38] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[39] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106(3):491–511, 2006.

[40] Hermann Gehring and Jörg Homberger. Parallelization of a two-phase metaheuristic for routing problems with time windows. *Journal of Heuristics*, 8(3):251–276, 2002.

[41] Sylvie Gélinas, Martin Desrochers, Jacques Desrosiers, and Marius M Solomon. A new branching strategy for time constrained routing problems with application to backhauling. *Annals of Operations Research*, 61(1):91–109, 1995.

[42] Timo Gschwind, Stefan Irnich, Ann-Kathrin Rothenbächer, and Christian Tilk. Bidirectional labeling in column-generation algorithms for pickup-and-delivery problems. *European Journal of Operational Research*, 266(2):521 – 530, 2018.

[43] LLC Gurobi Optimization. Gurobi optimizer reference manual, version 7.5, 2017.

[44] Stephan Held, William Cook, and Edward C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, 2012.

[45] Katrin Heßler, Timo Gschwind, and Stefan Irnich. Stabilized branch-and-price algorithms for vector packing problems. *European Journal of Operational Research*, 271(2):401 – 419, 2018.

[46] Stefan Irnich. Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, 30(1):113–148, 2008.

[47] Stefan Irnich, Guy Desaulniers, Jacques Desrosiers, and Ahmed Hadjar. Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing*, 22(2):297–313, 2010.

[48] Mads Jepsen, Bjorn Petersen, Simon Spoorendonk, and David Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

[49] B. Kallehauge, J. Larsen, and O.B.G. Madsen. Lagrangian duality applied to the vehicle routing problem with time windows. 33(5):1464–1487, 2006.

[50] O Kullmann. *Handbook of Satisfiability*, chapter Fundaments of branching heuristics, pages 205–244. IOS Press, Amsterdam, 2009.

[51] G. Laporte and Y. Nobert. A branch and bound algorithm for the capacitated vehicle routing problem. *Operations-Research-Spektrum*, 5(2):77–85, Jun 1983.

[52] Pierre Le Bodic and George Nemhauser. An abstract model for branching and its application to mixed integer programming. *Mathematical Programming*, 166(1):369–405, Nov 2017.

[53] Haibing Li and Andrew Lim. A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, 12(02):173–186, 2003.

[54] Humberto Longo, Marcus Poggi De Aragao, and Eduardo Uchoa. Solving capacitated arc routing problems using a transformation to the cvrp. *Computers & Operations Research*, 33(6):1823–1837, 2006.

[55] J. Lysgaard. *CVRPSEP: A package of separation routines for the capacitated vehicle routing problem.* Aarhus School of Business, Department of Management Science and Logistics, 2003.

[56] Jens Lysgaard, Adam N. Letchford, and Richard W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445, Jun 2004.

[57] Robert M. Nauss. Solving the generalized assignment problem: An optimizing and heuristic approach. *INFORMS Journal on Computing*, 15(3):249–266, 2003.

[58] Axel Parmentier. Algorithms for non-linear and stochastic resource constrained shortest path. *Mathematical Methods of Operations Research*, 89(2):281–317, 2019.

[59] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. In *Proceedings of the XVII IPCO*, volume 8494 of *Lecture Notes in Computer Science*, pages 393–403. Springer, 2014.

[60] Diego Pecin, Claudio Contardo, Guy Desaulniers, and Eduardo Uchoa. New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 29(3):489–502, 2017.

[61] Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, 2017.

[62] Diego Pecin, Artur Pessoa, Marcus Poggi, Eduardo Uchoa, and Haroldo Santos. Limited memory rank-1 cuts for vehicle routing problems. *Operations Research Letters*, 45(3):206 – 209, 2017.

[63] Diego Pecin and Eduardo Uchoa. Comparative analysis of capacitated arc routing formulations for designing a new branch-cut-and-price algorithm. *Transportation Science*, (Forthcoming), 2019.

[64] Artur Pessoa, Ruslan Sadykov, and Eduardo Uchoa. Enhanced branch-cut-and-price algorithm for heterogeneous fleet vehicle routing problems. *European Journal of Operational Research*, 270:530–543, 2018.

[65] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 30(2):339–360, 2018.

[66] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. In Andrea Lodi and Viswanath Nagarajan, editors, *Integer Programming and Combinatorial Optimization*, volume 11480, pages 354–369. Springer, 2019.

[67] Bjørn Petersen, David Pisinger, and Simon Spoorendonk. Chvátal-gomory rank-1 cuts used in a dantzig-wolfe decomposition of the vehicle routing problem with time windows. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 397–419. Springer, 2008.

[68] David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.

[69] M. Poggi de Aragão and E. Uchoa. Integer program reformulation for robust branch-and-cut-and-price. In L. Wolsey, editor, *Annals of Mathematical Programming in Rio*, pages 56–61, Búzios, Brazil, 2003.

[70] Marius Posta, Jacques A. Ferland, and Philippe Michelon. An exact method with variable fixing for solving the generalized assignment problem. *Computational Optimization and Applications*, 52:629–644, 2012.

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

[72] Roberto Roberti and Aristide Mingozzi. Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, 48(3):413–424, 2014.

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

[74] Stefan Ropke and Jean-François Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

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

[76] Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa. A bucket graph based labeling algorithm with application to vehicle routing. Technical Report L-2017-7, Cadernos do LOGIS-UFF, Niterói, Brazil, October 2017.

[77] Ruslan Sadykov, François Vanderbeck, Artur Pessoa, Issam Tahiri, and Eduardo Uchoa. Primal heuristics for branch-and-price: the assets of diving methods. *INFORMS Journal on Computing*, 31(2):251–267, 2019.

[78] J. E. Schoenfield. Fast, exact solution of open bin packing problems without linear programming. Technical report, US Army Space and Missile Defense Command, 2002.

[79] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[80] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Anand Subramanian, and Thibaut Vidal. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017.

[81] François Vanderbeck, Ruslan Sadykov, and Issam Tahiri. BaPCod — a generic Branch-And-Price Code, 2018.

[82] François Vanderbeck and Laurence A Wolsey. Reformulation and decomposition of integer programs. In *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer, 2010.

[83] Laguna Wei, Zhixing Luo, Roberto Baldacci, and Andrew Lim. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, Forthcoming, 2019.