CIRRELT-2021-27

A Branch-and-Price-and-Cut Algorithm for the Vehicle Routing Problem with Two-Dimensional Loading Constraints

Xiangyi Zhang
Lu Chen
Michel Gendreau
André Langevin

July 2021

A Branch-and-Price-and-Cut Algorithm for the Vehicle Routing Problem with Two-Dimensional Loading Constraints

Xiangyi Zhang[1], Lu Chen[2], Michel Gendreau[1], André Langevin[1]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), and Department of Mathematics and Industrial Engineering, Polytechnique Montréal

[2] School of Mechanical Engineering, Shanghai Jiao Tong University,Shanghai 200240, China

Abstract. The vehicle routing problem with two-dimensional loading constraints (2L-CVRP) is a practical variant of the classic capacitated vehicle routing problem. A number of algorithms have been developed for the problem, but it is very difficult for the existing exact methods to optimally solve instances featuring with large rectangular items. To address this issue, a branch-and-price-and-cut (BPC) algorithm is proposed in this study. A novel data structure and a new dominance rule are developed to build an exact pricing algorithm which takes the loading constraints into account. Several valid inequalities are introduced to strengthen the linear re-laxation. Extensive computational experiments were conducted on the bench mark instances of the 2L-CVRP, showing that the BPC algorithm outperforms all the existing exact methods for the problem in terms of the solution quality. Fourteen instances are solved to optimality for the first time. In particular, the size of solvable instances with large items is nearly doubled. Moreover, managerial insights about the impact of respecting the last-in-first-out constraint are also obtained.

Keywords. Capacitated vehicle routing, loading constraints, column generation, trie.

Acknowledgements. The authors are grateful to Professor Jean-François Côté from Université Laval for the codes of the packing algorithms and valuable suggestions on the subject. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

* Corresponding author: xiangyi.zhang@polymtl.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2021

1 Introduction

The vehicle routing problem with two-dimensional loading constraints (2L-CVRP) is an extension of the classical capacitated vehicle routing problem (CVRP) in which vehicles transport rectangular items characterized by length, width, and weight. The problem was introduced by Iori et al. (2007) who solve it by a branch-and-cut (B&C) algorithm. Two-dimensional loading constraints are mainly characterized by the orientation of items and a sequential restriction. The *orientation* concerns whether an item can be rotated or not. For example, items that are loaded by forklifts cannot be rotated. As for the *sequential restriction*, which is also called the *Last-In-First-Out (LIFO)* or *rear loading* constraint, it specifies that items belonging to successive customers along a route are not allowed to be moved. Besides, other characteristics of the loading constraints refer to *non-overlapping loading* and *orthogonal loading*, which respectively mean that all the items loaded in a vehicle cannot be overlapped and that the four edges of a loaded item should be parallel to the sides of the vehicle. Both constraints must be satisfied in the context of the 2L-CVRP. Fuellerer et al. (2009) categorized the 2L-CVRP into four variants based on different configurations of the loading constraints.

a) $2|RO|L$: sequential restriction (R) and oriented items (O) ;

b) $2|UO|L$: unrestricted (U) and oriented items (O) ;

c) $2|RN|L$: sequential restriction (R) and non-oriented items (N);

d) $2|UN|L$: unrestricted (U) and non-oriented items (N).

The 2L-CVRP has great practical value. It is commonly found in the transportation of furniture, large industrial equipment, and fragile goods. Items, in these cases, cannot be stacked for some reason. For example, kitchen appliances are not stacked in the course of delivery to avoid being crushed. Large industrial equipment like excavators cannot be stacked for stability reason. A concrete business case can be seen at Opein (`www.opein.com`), a medium-sized company which distributes large equipment to their customers. The practical value of the 2L-CVRP has motivated a number of studies on algorithm design, including both exact algorithms and heuristics, and solving new practical variants (Pollaris et al. 2015).

Besides the practical value of the 2L-CVRP, there are two additional motivations for the study. The state-of-the-art exact algorithm, the B&C algorithm developed by Côté et al. (2020), performs less efficiently as items become relatively large with respect to the loading area of the vehicle (Zhang et al. 2021a). By contrast, practical problems often feature large items. The motivation is to develop a new exact algorithm to address this issue. Another reason triggering the research is related to the performance of the column generation (CG) approach in the 2L-CVRP. CG-based exact algorithms have been proved to be very effective in vehicle routing problems (Costa et al. 2019); however, the only existing CG algorithm (Pinto et al. 2016) for the 2L-CVRP works in a heuristic fashion due to the loading constraints. Its performance is not satisfactory compared to meta-heuristics such as the ones proposed by Fuellerer et al. (2009) or Leung et al. (2011). Based on the great success of CG algorithms in VRPs, we believe that CG has the potential to be the key component of an efficient exact algorithm for the 2L-CVRP.

The main contributions of the study are as follows.

a) We present an exact pricing algorithm based on a novel data structure and an exact dominance rule.

b) We propose a novel strategy for variable selection using the new data structure to build the branch-and-price-and-cut (BPC) algorithm, which turns out to be effective for the 2L-CVRP.

c) Extensive computational experiments are carried out showing that the algorithm outperforms the state-of-the-art exact algorithm. Fourteen instances are solved to optimality for the first time. The solvable sizes of some families of instances are nearly doubled.

d) Instances without the LIFO constraint are also solved. The resulting optimal solutions enable us to conduct precise analyses on the impact of respecting the LIFO constraint, which provide valuable insights from a managerial point of view.

The remainder of the paper is organized as follows. Section 2 reviews the existing literature. Section 3 presents the formal description of the problem and the mathematical formulations. Section 4 elaborates on the set partitioning formulation, the new dominance rule, and the pricing algorithm. Section 5 presents the novel data structure and the BPC algorithm. Computational results are reported in Section 6, followed by conclusions in the last section.

2 Literature review

In this section, the main literature related to the 2L-CVRP and its variants is reviewed. Section 2.1 presents a variety of 2L-CVRPs that have been addressed so far. In Section 2.2, we review the literature on the algorithms for the 2L-CVRP is reviewed.

2.1 Literature on existing 2L-CVRPs

The first study on the 2L-CVRP was carried out by Iori et al. (2007) in which the problem addressed is $2|RO|L$. Gendreau et al. (2008) further investigated $2|RO|L$ and $2|UO|L$. Later on, Fuellerer et al. (2009) considered all the four variants of the 2L-CVRP. There are also some studies in which real-life constraints are added. As Pollaris et al. (2015) wrote a thorough review on the relevant problems published before 2015, we merely focus on problems that were studied from 2015 to 2020. Dominguez et al. (2016b) explored $2|UO|L$ and $2|UN|L$ with heterogeneous fleet, which stemmed from the construction industry. Dominguez et al. (2016a) studied $2|RN|L$ with backhauls, in which, a route may include both delivery customers and pick-up customers. The study was motivated by the daily operation of *Opein*. For the same company, Guimarans et al. (2018) addressed $2|UN|L$ with stochastic travel times and overtime penalty, which usually happens in practice. Alinaghian et al. (2017) addressed $2|UO|L$ with linearly time-dependent traveling times and two objectives minimizing: respectively the total traveling distance and the maximal load of the vehicles. A practical 2L-CVRP was proposed by Annouch et al. (2016), issued from the natural gas industry. Its background is that gas cylinders have to be transported from stations and to deposits. The problem features an heterogeneous fleet, time windows, several depots, and split deliveries. Another appealing practical problem was tackled by Song et al. (2019) where vehicles have multi-compartments. The loading area of a vehicle could carry vegetables and fruits, chemicals, frozen food and so forth, which are not allowed to be stored together. Due to the short shelf life of some food, time windows are also considered. Very recently, Côté et al. (2020) studied $2|RO|L$ where the width and the length of an item remains uncertain until it is loaded. The problem is motivated by retailers selling large appliances. Except for Iori et al. (2007) and Côté et al. (2020), in the above studies, heuristics are developed to solve the problems.

2.2 Literature on algorithms for the 2L-CVRP

After the seminal work of Iori et al. (2007), Gendreau et al. (2008) developed a Tabu Search heuristic for the 2L-CVRP. This heuristic finds optimal solutions for most of the closed instances and identifies feasible solutions for all the benchmark instances much more quickly than the algorithm of Iori et al. (2007). The Tabu Search heuristic was then incorporated with Guided Local Search by Zachariadis et al. (2009), leading to several new best solutions. The best heuristic algorithm for the 2L-CVRP so far is from Wei et al. (2018); it is composed of a simulated annealing heuristic with a local-search-based packing algorithm. The algorithm surpasses all the other heuristics in terms of solution quality.

There is also a branch-and-price (BP) heuristic for the $2|RO|L$ (Pinto et al. 2016). The authors formulated the problem as a set partitioning model, where the two-dimensional loading constraint is considered in the pricing problem. Firstly, they generate a set of columns without considering the loading constraints. Secondly, all the columns are checked by a packing algorithm. If a column is feasible, then it is added to the master problem. If a column is infeasible, a rectifier is invoked to make the column feasible. If the rectifier fails, then the column is discarded. To accelerate the CG approach, the same authors also proposed a variable neighborhood search heuristic (Pinto et al. 2015) to solve the pricing problem. Because the packing algorithm and the dominance rule used in the labeling algorithm are not exact, one cannot guarantee the exactness of the entire BP algorithm. Although the authors explored some interesting branching strategies for the 2L-CVRP, the overall performance of the algorithm is not competitive. No new best-known solution was derived. It turns out that the best integer solutions derived by the algorithm for instances with more than 26 customers are worse than the solutions obtained by Zachariadis et al. (2009). There is one BP algorithm for a variant of the 2L-CVRP. Zhang et al. (2021b) developed an exact BP for $2|UO|L$ with time window constraints. The authors proposed an exact dominance rule for the pricing problem which includes the loading constraints. They trained a feasibility predictor by machine learning techniques and installed the predictor into the CG algorithm to mitigate the computational complexity brought by the loading constraints. The resulting BP algorithm could solve instances with 50 customers and 103 items to optimality and the majority of the best-known solutions derived by Khebbache-Hadji et al. (2013) were improved. In addition, there is a BP algorithm for the CVRP with three-dimensional loading constraints (3L-CVRP) (Mahvash et al. 2017). Columns are generated by solving the classic elementary shortest path problem. Then an extreme point-based algorithm is leveraged to check the feasibility of the columns with regards to the loading constraints. The BP algorithm outperforms the methods proposed by Gendreau et al. (2006) and Tarantilis et al. (2009) on both solution quality and speed.

Iori et al. (2007) proposed the first exact algorithm for $2|RO|L$. The classic two-index formulation of the CVRP is adjusted to address the loading constraints. An exact branch-and-bound algorithm was developed to solve the packing problem. The classic rounded capacity inequality was employed where the right-hand-side considered the continuous bound of the two-dimensional bin packing problem (Martello and Vigo 1998). As the B&C algorithm progresses, when an integer solution is found, the exact packing algorithm verifies the loading constraints. If the solution is feasible, it is accepted as a candidate to update the incumbent global upper bound, otherwise it is discarded. The authors created a set of benchmark instances for testing, of which the algorithm could solve to optimality instances with up to 35 customers and 114 items to optimality. Hokama et al. (2016) proposed an exact packing algorithm for the same problem using constraint programming techniques and added some classic valid inequalities for CVRP (Lysgaard et al. 2004) to improve the B&C algorithm of Iori et al. (2007). Substantial reduction on CPU time was achieved, however, no open instance was closed. It was only with the publication of Côté et al. (2020) that some new open instances were solved. The authors proposed infeasible set inequalities which worked for integer nodes of the branch-and-bound tree. With state-of-the-art packing algorithms (Côté et al. 2014a,b), all the instances solved by Iori et al. (2007) and 26 open instances were solved to optimality within two hours of CPU time, compared to the 24-hour limit in the seminal study. Zhang et al. (2021a) further proposed a heuristic to separate infeasible set inequalities for fractional nodes. The resulting B&C algorithm closed 6 open instances and improved the dual bounds for other open instances by 1.0%

on average. Their computational analysis shows that although the current framework of the B&C algorithm significantly surpasses the B&C algorithm of Iori et al. (2007), the algorithm struggles to solve instances with large items.

3 Mathematical formulation

The problem addressed in this study is $2|RO|L$, which is formally defined as follows. We are given a complete undirected graph $G = (V, E)$, where $V = \{0, 1, 2, ..., n\}$ is a set of vertices including customers $V_c = \{1, 2, ..., n\}$ and the depot 0. Vertex $n + 1$ represents a copy of the depot. E is the set of edges. $\forall e \in E$, c_e is the traveling cost associated with edge e. An alternative representation of an edge (v_i, v_j) is also used throughout the article. A set K of homogeneous vehicles is available at the depot. A vehicle $k \in K$ has a loading surface whose length and width are equal to H and W, respectively. Thus, the total area of the loading surface is equal to $A = H \times W$. Each vehicle also has a weight capacity denoted as Q.

Each customer $i \in V_c$ is associated with a set of rectangular items M_i. Each item m in set M_i has a specific width w_{im}, length h_{im} and weight q_{im}. ν_i and c_i denote the total area and total weights of the items belonging to customer i, respectively, where $\nu_i = \sum_{m \in M_i} w_{im} h_{im}$ and $q_i = \sum_{m \in M_i} q_{im}$. The number of items is the cardinality of set M_i. Set M is the union of all the items, i.e., $M = \{m | m \in M_i, \forall i \in V_c\}$. The problem is to plan a route for each vehicle in the fleet K to cover the demands of all customers such that the following constraints are respected:

1. Each customer is visited exactly once.
2. Total weight of the carried items must not exceed the vehicle capacity.
3. Transported items in a vehicle must not exceed the loading area.
4. Items must be positioned without being overlapped.
5. Rotating items is not allowed.
6. Items of customers served later cannot be moved (the LIFO rule).

The LIFO constraint can be formally described as follows. $\forall m \in M$, let $\mathcal{X}(m)$ and $\mathcal{Y}(m)$ stand for the x-coordinate and y-coordinate (left-bottom wise) of item m in the loading area, respectively. For any pair of items $m, m' \in M$ such that $\mathcal{N}(m) < \mathcal{N}(m')$ where $\mathcal{N}(m)$ denotes the sequence number of item m (item m' has to be delivered before m), the following logical constraint should be respected.

$$\mathcal{X}(m) \geq \mathcal{X}(m') + w_{m'} \quad or \quad \mathcal{X}(m') \geq \mathcal{X}(m) + w_m \quad or \quad \mathcal{Y}(m') \geq \mathcal{Y}(m) + h_m \tag{1}$$

There are also two conventions in the field of developing exact algorithms for the 2L-CVRP: any route serving a single customer is forbidden and the number of used vehicles equals the fleet size (Iori et al. 2007, Côté et al. 2020).

The three-index formulation

To present the three-index formulation of the problem, we first define some necessary notations. Given a node $i \in V$, let $\delta(i)$ represent the set of edges incident to it. Given a subset S of V, $\delta(S)$ denotes the edges with only one endpoint in S. Let (S, σ) be the route constructed by set S in order σ. We denote by $E(S, \sigma)$ the set of edges in route (S, σ). We denote by $\Sigma(S)$ the set of feasible sequences for set S. Let x_e^k be a binary variable for each $e \in E$ and $k \in K$, where $x_e^k = 1$ if edge e is traversed by vehicle k and $x_e^k = 0$ otherwise. Let y_i^k be a binary variable for each vertex $i \in V_c$ and vehicle $k \in K$. y_i^k takes the value 1 if vehicle k serves customer i and 0 otherwise. The three-index formulation is as follows:

$$[P] \quad \min \sum_{k \in K} \sum_{e \in E} c_e x_e^k \tag{2}$$

$$s.t.$$

$$\sum_{e \in \delta(0)} \sum_{k \in K} x_e^k = 2|K| \tag{3}$$

$$\sum_{e \in \delta(i)} x_e^k = 2y_i^k, \ \forall i \in V_c, \ \forall k \in K \tag{4}$$

$$\sum_{e \in \delta(S)} x_e^k \geq 2y_i^k, \ \forall S \subset V_c, 1 < |S| < n-1, \ \forall i \in S, \ \forall k \in K \tag{5}$$

$$\sum_{k \in K} y_i^k = 1, \ \forall i \in V_c \tag{6}$$

$$\sum_{i \in V_c} q_i y_i^k \ \leq Q, \ \forall k \in K \tag{7}$$

$$\sum_{i \in V_c} \nu_i y_i^k \ \leq A, \ \forall k \in K \tag{8}$$

$$\sum_{e \in E(S,\sigma)} \sum_{k \in K} x_e^k \leq |S| - 1, \ \forall (S, \sigma) \text{ such that } \sigma \notin \Sigma(S), \tag{9}$$

$$x_e \in \{0,1\}, \ \forall e \in E; \ y_i^k \in \{0,1\}, \ \forall i \in V \tag{10}$$

This formulation is derived from the three-index formulation for the CVRP proposed by Fischetti et al. (1995). Expression (2) indicates the objective is to minimize the total traveling distance. Constraint (3) states the total degree of the depot. Constraint set (4) connects the two groups of decision variables and imposes the degree associated with a customer or the depot in terms of a single vehicle. Constraint set (5) eliminates subtours for each vehicle. Constraint set (6) imposes that each customer is visited exactly once. Constraint sets (7 - 8) impose that the vehicle capacity and the loading area should not be exceeded on a route. Constraint set (9) eliminates infeasible routes from the feasible region. Constraint set (10) defines the domain of the variables.

4 The column generation algorithm

In this section, the three-index formulation is transformed into a set partitioning model. The CG algorithm is proposed to solve the model. It is described in the following order: the overall framework, the route relaxation technique, the labeling algorithm with the exact dominance rule, the new completion bounds, and the heuristic pricing strategy.

4.1 Set partitioning formulation

If we apply the Danzig-Wolfe decomposition to the three-index formulation, a set partitioning formulation for $2|RO|L$ is derived. Let Ω be the collection of all possible feasible routes. Let λ_r be a binary variable, where $\lambda_r = 1$ if a route $r \in \Omega$ is selected 0 otherwise. Let c_r be the total distance traveled on route r and a_{ir} be a constant indicating whether or not vertex i is visited in route r. Then, the set partitioning formulation can be written as follows.

$$\min \sum_{r\in\Omega} c_r\lambda_r \tag{11}$$

$$s.t.$$

$$\sum_{r\in\Omega} \lambda_r = |K| \tag{12}$$

$$\sum_{r\in\Omega} a_{i,r}\lambda_r = 1,\ \forall i \in V_c \tag{13}$$

$$\lambda_r \in \{0,1\},\ \forall r \in \Omega \tag{14}$$

The objective function (11) minimizes the total cost of the selected routes. Constraint (12) restricts the number of selected routes to the fleet size. Constraint set (13) imposes that each customer is served exactly once. We call model (11 - 14) the *integer programming master problem (IPM)*. By relaxing (14) as

$$0 \leq \lambda_r \leq 1,\ \forall r \in \Omega$$

a lower bound of the IPM can be obtained. Usually the LP relaxation (called *linear programming master problem (LPM)*) is solved by a CG-based algorithm (Desaulniers et al. 2006) because it is unpractical to enumerate all columns. To solve the LMP, one starts from a subset of columns $\bar{\Omega} \subset \Omega$ that forms a restricted LMP. At each iteration, a pricing problem is solved to generate columns with negative reduced cost. The pricing problem of the $2|RO|L$ is

$$[PP] \quad \min \sum_{e\in E} \bar{d}_e x_e - \pi_f \tag{15}$$

$$s.t. \tag{16}$$

$$\sum_{e\in\delta(i)} x_e = 2 \quad \forall i \in V \tag{17}$$

$$\sum_{e\in\delta(S)} x_e \geq 2 \quad \forall S \subset V_c,\ 1 < |S| < n-1,\ \forall i \in S,\ \forall k \in K \tag{18}$$

$$\sum_{(i,j)\in E} x_{ij}(q_i + q_j) \leq 2Q \tag{19}$$

$$\sum_{(i,j)\in E} x_{ij}(\nu_i + \nu_j) \leq 2A \tag{20}$$

$$\sum_{e\in E(S,\sigma)} x_e \leq |S| - 1,\ \forall (S,\sigma) \text{ such that } \sigma \notin \Sigma(S), \tag{21}$$

$$x_e \in \{0,1\} \forall e \in E \tag{22}$$

where $\bar{d}_e$ is defined as

$$\bar{d}_{ij} = c_{ij} - \frac{1}{2}\pi_i - \frac{1}{2}\pi_j,\ \forall (i,j) \in E$$

π_i and π_f are the dual values associated with constraints (13) for node i and the constraint (12), respectively. By removing constraint (21), PP is reduced to the classic elementary shortest path problem with resource constraints (ESPPRC), which has been addressed in many studies (Feillet et al. 2004, Righini and Salani 2008, Desaulniers et al. 2008, Martinelli et al. 2014). Details on the characteristics of the ESPPRC can be found in these papers.

4.2 The framework of the column generation algorithm

A difference with conventional CG algorithms for the CVRP (Desaulniers et al. 2006), due to the two-dimensional loading constraints, is that, once a column (route) is identified by the pricing algorithm, the column should be checked for the loading constraints by a feasibility checker. Throughout this study, the state-of-the-art exact algorithm developed in Côté et al. (2014b) is applied as the checker. When the checker approves the feasibility of a column, the column is added to the master, otherwise it is discarded. Figure 1 provides a diagram of the CG algorithm used in this study.

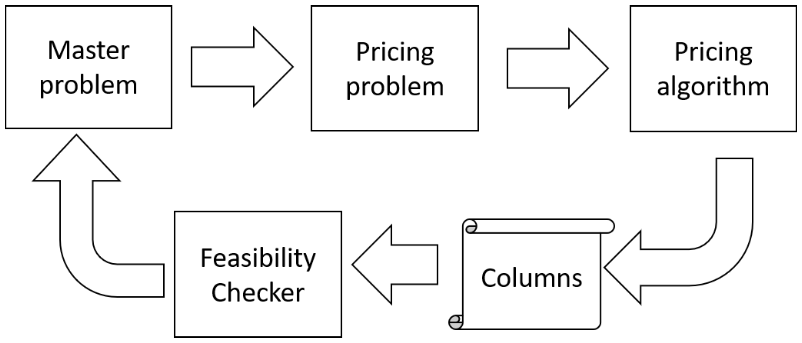


Figure 1: The diagram of the column generation algorithm

4.3 The Path Relaxation

In the literature, the *elementarity* of a path can be relaxed to achieve easier pricing problems. There are many CG approaches based on such relaxation (Desrochers et al. 1992, Righini and Salani 2008, Desaulniers et al. 2008, Baldacci et al. 2011). We apply the well-known ng-route relaxation according to the state-of-the-art pricing algorithms for CVRP. The ng-route relaxation was proposed by Baldacci et al. (2011); it remains to be one of the key component of the state-of-the-art CG approaches for CVRP. Associated with a customer i, there is a set of pre-selected customers $N_i \subseteq V_c$ (including i itself), such that $|N_i| \leq \Delta(N_i)$, where $\Delta(N_i)$ is a parameter to specify the maximal cardinality of the set. Given a partial path $P = (0, i_1, i_2, ..., i_k)$, any customer in set $\Pi(P)$ is not allowed to be visited next to i_k, where $\Pi(P)$ is defined as:

$$\Pi(P) = \{i_r : i_r \in \bigcap_{s=r+1}^{k} N_{i_s}, r = 1, ..., k-1\} \bigcup \{i_k\}$$

By the restriction imposed by $\Pi(P)$, the ng-route relaxation can obtain much tighter lower bounds than those found fully relaxing the *elementarity* conditions (Baldacci et al. 2011).

For example, let $r = [0, 4, 1, 2, 3]$ be a partial path, and let $N_1 = \{1, 3, 4\}$, $N_2 = \{2, 4, 5\}$, $N_3 = \{1, 3, 4\}$, $N_4 = \{1, 4\}$. Then $1 \notin N_2 \cap N_3$, $2 \notin N_3$, $3 \in N_3$, $4 \in N_1 \cap N_2 \cap N_3$. Therefore, $\Pi(r) = \{3, 4\}$, meaning that the the partial path cannot be extended to customer 3 or customer 4.

4.4 The definition of labels and the exact dominance rule

Labeling algorithms have proved to be very successful in solving pricing problems when it comes to CG methods for vehicle routing problems (Desaulniers et al. 2006). We apply the framework of the labeling algorithm in Martinelli et al. (2014), which is one of the best pricing algorithms for the CVRP.

A label is the main entity to manipulate in a labeling algorithm. It represents a partial path starting from the depot to some vertex while recording critical information based on which further path extension is performed. Let $(\eta(L), c(L), q(L), \mu(L), s(L))$ be the representation of a label L. $\eta(L)$ stands for the end vertex of the label. $c(L)$ is the total cost accumulated along the partial path. $q(L)$ stands for the accumulated weight of the label. $\mu(L)$ represents the unreachable nodes. $s(L)$ stores the rectangular items collected so far while memorizing the collecting sequence. Figure 2 illustrates an example. The label representing partial path $0-1-2$ is defined as $(2, 8, 15, \{1, 2\}, \{m_{11}, m_{12}, m_{21}\})$ since node 1 and node 2 are not allowed to be visited onward.

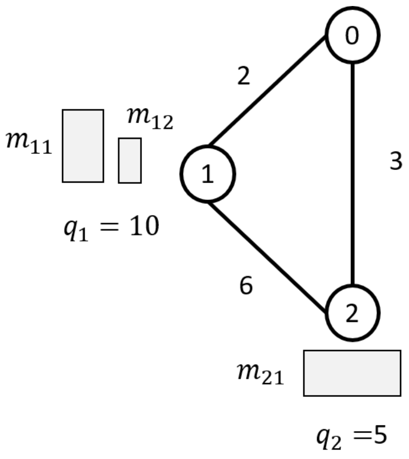


Figure 2: An example of the label representation

Label L_1 dominates L_2 if for an arbitrary legal extension of L_2 towards the depot, one can always find a legal extension of L_1 towards the depot with less or equal cost. $L_1 \prec L_2$ denotes "L_1 dominates L_2". Dominance rules specify the dominance relation between two labels. The efficiency of a labeling algorithm heavily relies on which dominance rules are applied. A tight dominance rule enables a labeling algorithm to prune the majority of labels so that the computational effort can be massively reduced. The following dominance rule is proposed for the $2|RO|L$.

Theorem 1. *$L_i \prec L_j$ if the following conditions are satisfied:*
1) $\eta(L_i) = \eta(L_j)$;
2) $c(L_i) \leq c(L_j)$;
3) $q(L_i) \leq q(L_j)$;
4) $\mu(L_i) \subseteq \mu(L_j)$;
5) $s(L_i) \sqsubseteq s(L_j)$.

Proof. See Lemma 3. □

Conditions 1-4 of Theorem 1 are conventional, but the last condition $s(L_i) \sqsubseteq s(L_j)$ regards the loading constraints and is explained in Statement 1. In the rest of this subsection, we show that checking the last condition is equivalent to solving a two-dimensional sequential variable size bin packing problem (2DSVSBPP). We also propose a fast heuristic to address it.

Given two labels L_i and L_j, if the last condition is skipped and $L_i \prec L_j$ holds in terms of the other four conditions, one could end up with a complete path extending from L_i to the depot that has a negative reduced cost but violates the loading constraints. If the same extension from L_j to the depot does not violate the loading constraints and has a negative reduced cost, then an improving column would be lost if L_j was eliminated by the dominance test. To prevent such mis-dominance from happening, the last condition must be considered.

The general idea behind the way to check the last condition is that items in $s(L_j)$ are seen as bins and then items in $s(L_i)$ are packed in the bins. If a feasible packing is found, the condition is satisfied.

Given two sets of items $s(L_i)$ and $s(L_j)$, items in $s(L_j)$ are treated as bins. For any item (bin) $k \in s(L_i)(s(L_j))$, there is a sequence number $\mathcal{N}(k)$ attached. Let $B(k)$, belonging to $s(L_j)$, be the bin holding item $k, \forall k \in s(L_i)$. $\mathcal{N}(B(k))$ represents the sequence number of the bin holding item k. $(\mathcal{X}(k), \mathcal{Y}(k))$ represents the coordinate of item k on the loading surface. $(\mathcal{X}(B(k)), \mathcal{Y}(B(k)))$ stands for the coordinate of bin $B(k)$ if it is placed as an item on the loading surface. Let $\mathcal{P}(s') = 1$ $(\mathcal{P}(s') = 0)$ denote that a set of items s' can (cannot) be packed in terms of the loading constraints in the loading area of a vehicle.

Statement 1. *$s(L_i) \sqsubseteq s(L_j)$ denotes that the items in $s(L_i)$ can be packed into the bins in $s(L_j)$ while respecting: for any pair of items h, k with different sequence numbers $\mathcal{N}(h) < \mathcal{N}(k)$ such that 1) $\mathcal{N}(B(h)) \leq \mathcal{N}(B(k))$; 2) If $\mathcal{N}(B(h)) = \mathcal{N}(B(k))$, the LIFO constraint should be considered for packing items in $B(h)$.*

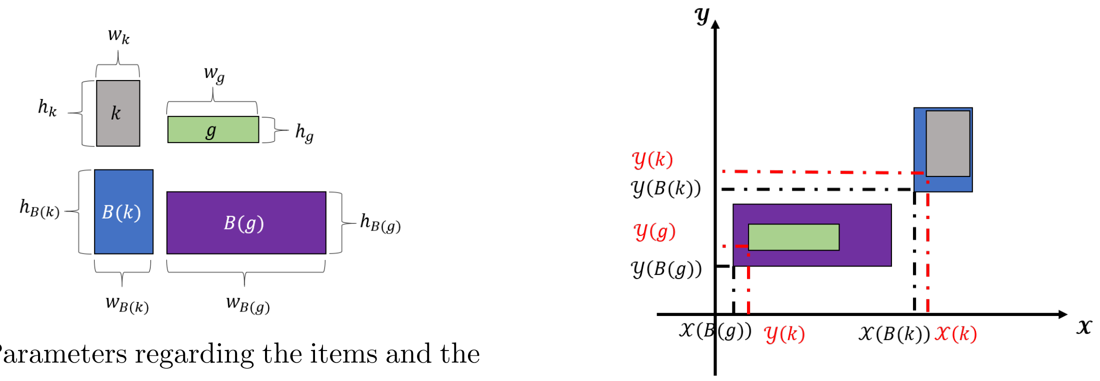
Lemma 1. *Given that $s(L_i) \sqsubseteq s(L_j)$, for any pair of items $k, g \in s(L_i)$ and $B(k), B(g) \in s(L_j)$ if $\mathcal{X}(B(k)) \geq \mathcal{X}(B(g)) + w_{B(g)}$, then $\mathcal{X}(k) \geq \mathcal{X}(g) + w_g$.*

Proof. Because $B(k)$ holds item k, $\mathcal{X}(k) \geq \mathcal{X}(B(k))$. Because $B(g)$ holds item g, $\mathcal{X}(g) + w_g \leq \mathcal{X}(B(g)) + w_{B(g)}$. Then we have $\mathcal{X}(k) \geq \mathcal{X}(B(k)) \geq \mathcal{X}(B(g)) + w_{B(g)} \geq \mathcal{X}(g) + w_g$. □

Lemma 2. *Given that $s(L_i) \sqsubseteq s(L_j)$, for any pair of items $k, g \in s(L_i)$ and $B(k), B(g) \in s(L_j)$ if $\mathcal{Y}(B(k)) \geq \mathcal{Y}(B(g)) + h_{B(g)}$, then $\mathcal{Y}(k) \geq \mathcal{Y}(g) + h_g$.*

Proof. Because $B(k)$ holds item k, $\mathcal{Y}(k) \geq \mathcal{Y}(B(k))$. Because $B(g)$ holds item g, $\mathcal{Y}(B(g)) + h_{B(g)} \geq \mathcal{Y}(g) + h_g$. Then we have $\mathcal{Y}(k) \geq \mathcal{Y}(B(k)) \geq \mathcal{Y}(B(g)) + h_{B(g)} \geq \mathcal{Y}(g) + h_g$. □

Figure 3 graphically interprets Lemma 1 and Lemma 2, the coordinate relation when items from $s(L_i)$ and bins from $s(L_j)$ placed on the loading surface.



(a) Parameters regarding the items and the bins

(b) Coordinates of the items and the bins

Figure 3: Graphical meaning of Lemma 1 and Lemma2

Lemma 3. *Given a set of items s', such that $\forall k \in s'$, $\mathcal{N}(k) > \mathcal{N}(h), \forall h \in s(L_i) \cup s(L_j)$, if $s(L_i) \sqsubseteq s(L_j)$, then $\mathcal{P}(s' \cup s(L_j)) = 1 \Rightarrow \mathcal{P}(s' \cup s(L_i)) = 1$.*

Proof. To prove $s' \cup s(L_i)$ is packable amounts to showing that the coordinates of any pair of items satisfy constraint (1). For any pair of items $k \in s'$ and $g \in s' \cup s(L_i)$, it is trivial.

Suppose there is an pair of items $k, g \in s(L_i)$ such that $\mathcal{N}(k) < \mathcal{N}(g)$ and $\mathcal{N}(B(k)) \neq \mathcal{N}(B(g))$. Without loss of generality, it is assumed that $\mathcal{N}(B(k)) < \mathcal{N}(B(g))$. If $\mathcal{X}(B(g)) \geq \mathcal{X}(B(k)) + w_{B(k)}$, then $\mathcal{X}(g) \geq \mathcal{X}(k) + w_k$ (Lemma 1); if $\mathcal{Y}(B(g)) \geq \mathcal{Y}(B(k)) + h_{B(k)}$, then $\mathcal{Y}(g) \geq \mathcal{Y}(k) + h_k$ (Lemma 2). Hence, Constraint (1) is respected.

For any pair of items $k, g \in s(L_i)$ such that $\mathcal{N}(k) < \mathcal{N}(g)$ and $\mathcal{N}(B(k)) = \mathcal{N}(B(g))$, since items k, g are packed into the same bin, Constraint (1) is naturally satisfied. □

We are aware of efficient algorithms for the two-dimensional variable size bin packing problem (2DVSBPP) (Kang and Park 2003, Hong et al. 2014), however, they are not suitable in the context of the special structure of our problem: 1) the number of bins is not infinite; 2) the size of an item can be very close to the size of a bin; 3) it is only needed to answer whether a feasible solution exists or not; 4) we have a LIFO constraint. Moreover, since this is merely a sub-routine of the entire algorithm, its time budget is very limited. Considering these factors, a simple construction heuristic is developed to solve the 2DVSBPP.

While constructing a solution for the 2DVSBPP, there are two decisions to make: 1) Which item should be assigned to which bin? 2) How to arrange the items assigned to a bin? We borrow the backbone idea of the well-known best-fit algorithm (BFA) proposed by Burke et al. (2004) which builds a solution in an iterative fashion. At each step, the lowest niche (a niche is a segment as shown in Figure 4) is selected and then the most fitting item is chosen to be placed over the niche.

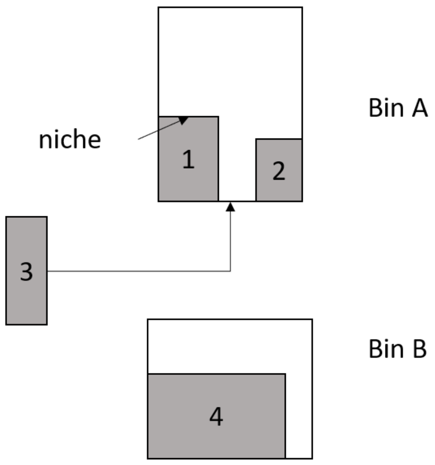


Figure 4: A typical step of placing an item

A generalized best-fit algorithm (GBFA) is developed, which shares the idea of choosing the best item for a niche at each step. Let B be the list of bins, I be the list of *bubbles*, each of which contains the items with the same sequence number. The pseudo-code of the algorithm is in Algorithm 1.

Algorithm 1 The GBFA

```
1:  Sort $B$ and $I$ by the non-decreasing order of the sequence number;
2:  while $I$ is not empty and $B$ is not empty do
3:      Select the first bin of $B$
4:      Select the remaining items in the first bubble of $I$.
5:      Pack the selected items into the selected bin by the BFA.
6:      if all the selected items are packed then
7:          Remove the first bubble of $I$
8:      else
9:          Remove the first bin of $B$
10: Return True if $I$ is empty else return False
```

The central idea of the GBFA is that bins are filled one by one in accordance with the sequence numbers. For each bin being filled, items are packed by the classical BFA, which naturally satisfies Constraint (1). Let n_I and n_B be the number of items in I and the number of bins in n_B, respectively. The worse time complexity happens when a bin can pack all the items. Hence, in light of the time complexity of the BFA Burke et al. (2004), the worse time complexity is $O({n_I}^2 + n_I B)$. On average, the classical BFA is performed $\frac{n_I}{n_B}$ times, leading to an average time complexity of $O(\frac{{n_I}^2}{n_B} + n_I W)$. If the GBFA can find a feasible packing solution for B and I, the algorithm returns *True* else it returns *False*. Therefore, with the GBFA we can efficiently check whether $s(L_i) \sqsubseteq s(L_j)$ holds for two given labels.

Algorithm 2 gives the pseudo-codes of the exact labeling algorithm. $\mathcal{M}(\nu, i)$ is a bucket that includes labels ending at customer i with accumulated area ν. Before a newly-generated label is inserted into a bucket, its unreachable set should be updated from two perspectives: customers that lead to violation of the weight limit Q or the loading constraints become unreachable; the details regarding how to efficiently detect infeasible routes are presented in Section 5. At the end of Algorithm 2, Algorithm 3 is invoked to build routes that satisfy the loading constraints.

Algorithm 2 The exact labeling algorithm

```
 1: Initialize matrix $\mathcal{M}$, ng-sets $N_i \in V_c, \forall i \in V_c \cup \{n+1\}$
 2: $\mathcal{M}(\nu, i) \leftarrow$ empty list, $\forall i \in V_c, \nu = 0, ..., A$
 3: $\mathcal{M}(\nu, i) \leftarrow \{i, \bar{d}_{0,i}, q_i, \{i\}, M_i\}$
 4: for $\nu = 1, ..., A$ do
 5:     for $i \in V_c \cup \{n+1\}$ do
 6:         if $\nu - \nu_i > 0$ then
 7:             for $j \in V_c$ do
 8:                 for $L \in \mathcal{M}(\nu - \nu_i, j)$ do
 9:                     $L' = (i, c(L) + \bar{d}_{j,i}, q(L) + q(i), \mu(L) \cap N_i \cup \{i\}, s(L) \cup M_i)$
10:                     InsertFlag ← True
11:             for $\tilde{\nu} = 1, ..., \nu$ do
12:                 for $\forall \tilde{L} \in \mathcal{M}(\tilde{\nu}, i)$ do
13:                     if $\tilde{L}$ dominates $L'$ then
14:                         InsertFlag ← True
15:                         break
16:                     else
17:                         if $L'$ dominates $\tilde{L}$ then
18:                             delete $\tilde{L}$
19:             if InsertFlag then
20:                 Update unreachable customers $\mu(L')$
21:                 $\mathcal{M}(\nu, i) \leftarrow \mathcal{M}(\nu, i) \cup \{L'\}$
22: Return BuildRoutes($\mathcal{M}$)
```

Algorithm 3 BuildRoutes($\mathcal{M}$)

```
1: Initialize $\mathcal{R} = \emptyset$
2: for $\nu = 1, ..., A$ do
3:     for $\forall L \in \mathcal{M}(\nu, n+1)$ do
4:         Parse label $L$ as route $r$
5:         if $r$ satisfies Constraints (9) then
6:             $\mathcal{R} = \mathcal{R} \cup \{r\}$
7: Return $\mathcal{R}$
```

4.5 Completion bounds

A completion bound is used to prune the labels that cannot produce negative reduced cost. As shown in Feillet et al. (2007), a tight bounding technique is able to significantly accelerate the labeling algorithm. Two completion bounds addressing the loading constraints are proposed to accelerate the labeling algorithm.

The first bound is based on the full relaxation of the loading constraints when solving the 2DVSBPP. Let $PP_j(Q', A')$ be the problem to find the shortest ng-route with respect to the given ng-sets. An ng-route should start from the depot and end up at customer j while the accumulated area and the accumulated weights must be less or equal to Q' and A', respectively. Moreover, the loading constraints should be satisfied. We denote the optimal cost of $PP_j(Q', A')$ as $T^*_j(Q', A')$. Let $\tilde{PP}_j(Q', A')$ be the problem variant of $PP_j(Q', A')$ which excludes the loading constraints. We denote the optimal cost of $\tilde{PP}_j(Q', A')$ as $\tilde{T}^*_j(Q', A')$. We then have Lemma 4 because the relaxation of the loading constraints broadens the solution space.

Lemma 4. $T_j^*(Q', A') \geq \tilde{T}_j^*(Q', A')$

Figure 5 shows an example of extending a label from customer i to customer j. Let l be the extending label and $a(l)$ be the total area of items in $s(l)$, the cost becomes $c(l) + \tilde{d}_{i,j}$ after the extension. With Lemma 4, we have $c(l) + \tilde{d}_{i,j} + T_j^*(Q - q(l) - q_j, A - a(l) - \nu_j) \geq c(l) + \tilde{d}_{i,j} + \tilde{T}_j^*(Q - q(l) - q_j, A - a(l) - \nu_j)$. Hence, if $\tilde{d}_{i,j} + \tilde{T}_j^*(Q - q(l) - q_j, A - a(l) - \nu_j) \geq 0$, then there is no need to perform the extension because it never leads to a path with a negative reduced cost.

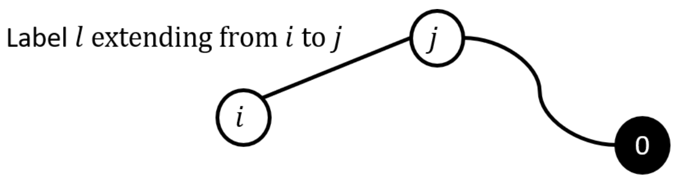


Figure 5: A demonstration of the completion bound

The second completion bound is based on the relaxation of the LIFO constraint when performing the labeling algorithm. Its validity is similar to the first completion bound. Note that when the LIFO constraint is relaxed, all the sequence number attached to items become 0. Therefore, Algorithm 1 still performs correctly.

4.6 A hierarchical labeling routine

Due to the loading constraints, using dominance rules is not as effective as it is for the CVRP. Running the exact labeling algorithm is thus much more computationally expensive. To address this issue, we propose a hierarchical labeling routine (shown in Algorithm 4), in which five heuristic versions of Algorithm 2 based on different combinations of settings (see Table 1) are applied before the exact version is finally activated.

Table 1: Five heuristic labeling algorithms

Version	Dominance rule	Route type	Bucket size
①	Theorem 1 condition 1	elementary	1
②	Theorem 1 condition 1	ng-route	1
③	Theorem 1 conditions 1-3	ng-route	unlimited
④	Theorem 1	ng-route	10
⑤	Theorem 1, relaxing LIFO	ng-route	unlimited

Algorithm 4 The hierarchical labeling routine

```
1: Initialize $i = 1$
2: while $i <= 5$ do
3:     Invoke Version ⓘ
4:     if No improving columns then
5:         $i \leftarrow i + 1$
6:     else
7:         Return the columns
8: Invoke Algorithm 2
```

The first completion bound can be derived by Version ③ when no columns are priced-out. The best cost of extending a label l from customer i to customer j is the sum of $c(l) + \tilde{d}_{i,j}$ and $\tilde{T}_j^*(Q - q(l) - q_j, A - a(l) - \nu_j)$ which is the optimal objective value of the following optimization problem. It can be solved by simply enumerating the labels in the bucket $\mathcal{M}(\nu, j)$.

$$\min_{L \in \mathcal{M}(\nu,j)} c(L) \tag{23}$$

$$s.t.$$

$$0 \leq \nu \leq A - a(l) - \nu_j \tag{24}$$

$$0 \leq q(L) \leq Q - q(l) - q_j \tag{25}$$

The second completion bound is derived from Version ⑤. It resembles the calculation of the first completion bound, so the details are not presented.

5 The L-Trie data structure

Even armed with the aforementioned hierarchical labeling routine and the exact dominance rule, the CG algorithm still suffers from the loading constraints because there are still many labels generated. To prune more labels, a new data structure named *L-Trie* is proposed, where the prefix "L" represents "loading constraints" to distinguish it from the traditional *Trie* in the field of string search (Brass 2008). In this section, the structure of *L-Trie* is described in the first place. Furthermore, functions related to *L-Trie* are introduced. Finally, the way to deploy *L-Trie* in the labeling algorithm is discussed.

5.1 Definitions and attributes

Definition 1. *Given a route $r = [i_1, i_2, ..., i_n]$, $[j_1, j_2, ..., j_{n'}]$ is a sub-route of r if we can find a sequence of indices $(k_1, k_2, ..., k_{n'}), 1 \leq k_1 < k_2, ..., k_{n'} \leq n$ such that $i_{k_1} = j_1, i_{k_2} = j_2, ..., i_{k_{n'}} = j_{n'}$.*

Definition 2. *A route r is a master-route of route r' if and only if r' is one of the sub-routes of r.*

For example, route 0-1-2-3-0 has the following sub-routes: 0-1-0, 0-2-0, 0-3-0, 0-1-2-0, 0-1-3-0, 0-2-3-0, 0-1-2-3-0. Route 0-1-2-3-0 is a master-route of all these sub-routes.

L-Trie is a container to store the information on checked routes. It is inspired by the idea of *Trie*, which has been successfully applied in string search (Brass 2008). *L-Trie* is an ordered tree data structure. A node γ on a *L-Trie* has the following attributes:

1. γ_1: customer id, indicating which customer is associated with the node.
2. γ_2: father node, indicating the immediately-preceding node. A node has only one father node.
3. γ_3: a list of child nodes, indicating all its children.
4. γ_4: a set of customers, indicating all the customers associated with the node and the children.
5. γ_5: an individual boolean sign, indicating if there exists a route ending up at the node.

Figure 6 gives an example of a *L-Trie* with 0-3-1-7-0, 0-3-8-1-0, 0-7-4-0, 0-9-6-1-0 stored. The empty node is the root of the *L-Trie*. The attributes of the node in red are presented in details.

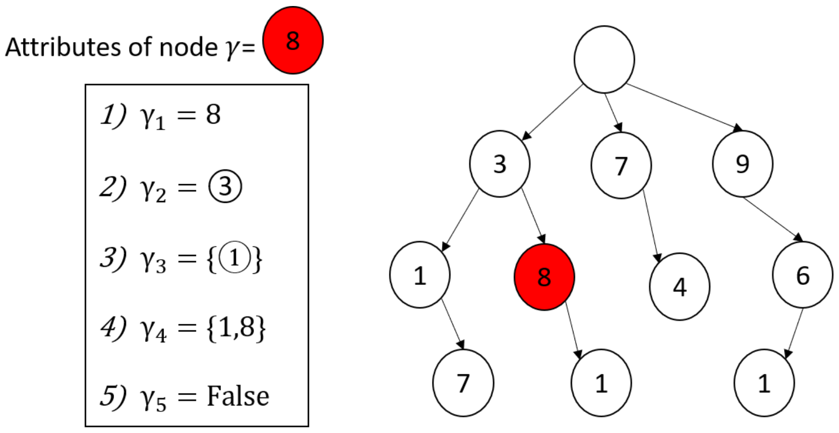


Figure 6: An example of *L-Trie*

5.2 Functions of *L-Trie*

To utilize the information stored in *L-Trie*, two important functions are defined upon the data structure.

1. *L-Trie::findSubRoute*, for a given route, determine if there exists some route stored in the *L-Trie* that is one of the sub-routes of the given route;
2. *L-Trie::findMasterRoute*, for a given route, determine if there exists some route stored in the *L-Trie* that is a master-route of the given route.

The value of having both functions can be justified by the following example. Suppose route 0-1-2-3-0 is passed to *L-Trie::findSubRoute* and route 0-1-3-0 is detected as a checked route labeled as infeasible in terms of the loading constraints, then it is safe to say that 0-1-2-3-0 is also infeasible without calling the feasibility checker. On the other hand, if route 0-1-2-3-0 is passed to *L-Trie::findMasterRoute* and some route, say 0-1-4-2-5-3-0 is detected as a checked route labeled as feasible, then it is safe to say 0-1-2-3-0 is feasible without invoking the feasibility checker. By calling the functions, we can check the feasibility of a route without invoking the feasibility checker.

Let α be the number of routes stored in a *L-Trie*, β be the maximal length of a route. The worst space complexity of the *L-Trie* in big-O notation is $O(\alpha\beta n)$. Besides *L-Trie::findSubRoute* and *L-Trie::findMasterRoute*, like the classic Trie, *L-Trie* has a search function to identify if a route has already been stored. It takes $O(\beta)$ to search a route with β customers. Furthermore, inserting a route with β customers in a *L-Trie* has the same time complexity.

L-Trie::findSubRoute The pseudo-codes of the function is shown in Algorithm 5. $r[i:]$ represents the $(i+1)^{th}$ customer through the last customer of r. Function *findPos(r, γ)* returns the index of the first customer equal to γ_1 in r. If there exists no such customer, the function returns -1. The central idea of *L-Trie::findSubRoute* is to seek a sub-route in the depth-first manner. Set γ_4 is critical for the efficiency of Algorithm 5 because it prevents the search from visiting those nodes that never leads to a sub-route. The worst time complexity of Algorithm 5 is $O(\alpha\beta)$, when all the routes stored in the tree are enumerated. The best time complexity of Algorithm 5 is $O(\omega)$ if there exists a sub-route with ω as the number of customers. Because of γ_4, the worst time complexity rarely occurs, in practice. In general, the algorithm works extremely fast.

Statement 2. *For a given route r, if L-Trie::findSubRoute finds a route r', then r' must be a sub-route of r.*

Statement 3. *For a given route r, if there exists a sub-route r' of r stored in the tree and r' is the only sub-route stored in the L-Trie, L-Trie::findSubRoute will find r'.*

The proofs for Statements 2 and 3 are presented in Appendix A.

Algorithm 5 L-Trie::findSubRoute(r, γ)

```
1: if some customer in $r$ is in $\gamma_4$ then
2:     for $\bar{\gamma}$ in $\gamma_3$ do
3:         pos = findPos($r$, $\bar{\gamma}_1$)
4:         if pos $\geq 0$ and L-Trie::findSubRoute($r[pos :]$, $\bar{\gamma}$) then
5:             Return True
6: Return $\gamma_5$
```

L-Trie::findMasterRoute The pseudo-codes of the function is shown in Algorithm 6. $r[1]$ denotes the first customer of route r. The backbone idea of the algorithm is to match the customers of a route one customer after another. Set γ_4 can effectively stops exploring the nodes that cannot produce a master-route. Likewise, the worst time complexity of Algorithm 6 is also $O(\alpha\beta)$. The best time complexity is $O(\omega)$ if there exists a master-route with ω customers.

Algorithm 6 L-Trie::findMasterRoute(r, γ)

```
1: if $r$ is empty then
2:     Return True
3: else
4:     for $\bar{\gamma}$ in $\gamma_3$ do
5:         if all customers in $r$ in $\bar{\gamma}_4$ then
6:             $flag \leftarrow$ False
7:             if $r[1] == \bar{\gamma}_1$ then
8:                 $flag \leftarrow$ L-Trie::findMasterRoute($r[1 :], \bar{\gamma}$)
9:             else
10:                $flag \leftarrow$ L-Trie::findMasterRoute($r, \bar{\gamma}$)
11:            if flag then
12:                Return True
13:    Return False
```

Statement 4. *For a given route r, if L-Trie::findMasterRoute finds a route r', then r' must be a master-route of r.*

Statement 5. *For a given route r, if there exists a master-route r' of r stored in the tree and r' is the only master-route of r stored in the L-Trie, L-Trie::findMasterRoute will find r'.*

The proofs for Statements 4 and 5 are presented in Appendix A.

5.3 The application of *L-Trie* in the labeling algorithm

To utilize the *L-Trie* in the context of $2|RO|L$, in principle, one should create two *L-Tries*, one for the feasible routes denoted as *FL-Trie* and the other for the infeasible routes denoted as *IL-Trie*. Given a route r, invoke Algorithm 6 upon *FL-Trie*, if the algorithm returns *True*, r is feasible. On the other hand, invoke Algorithm 5 upon *IL-Trie*, if the algorithm returns *True*, r is infeasible. If the feasibility of r is still pending after going through both algorithms, r is checked by the feasibility checker. Afterwards, it is stored in *FL-Trie* or *IL-Trie*, depending on its feasibility.

However, in practice we found that just creating the *IL-Trie* is more cost-effective in terms of speed and memory. There are two reasons for that. First, it is generally easier to verify the feasibility of feasible routes by a heuristic packing algorithm. Therefore, the saved time brought by the *FL-Trie* is less significant than the time saved by using the *IL-Trie.* Second, the data structure is very memory-intensive, so using both the *L-Tries* could quickly use up all memory.

In the labeling algorithm proposed in Section 4.6, for any label L, the *IL-Trie* is applied to verify whether extending L to a customer produces an infeasible route. With such a technique, the labeling algorithm can prune many labels by directly prohibiting extensions.

5.4 Features of the branch-and-price-and-cut algorithm

Valid inequalities
The following valid inequalities are applied for the $2|RO|L$: the rounded capacity inequalities (RCI) and the infeasible set inequalities (ISI). How to account the dual values associated with these inequalities in the pricing problem are not discussed in the study for the sake of conciseness. Readers can refer to Desaulniers et al. (2011) for details.

The RCI is widely used for vehicle routing problems. It can eliminate sub-tours and ensure the capacity limit of any vehicle to be respected. Let $S \subset V_c$ be a subset of customers and λ^* be the optimal solution of the restricted master problem. Let $\Omega(e,r)$ be the number of times for which edge e is traversed in route r, then the following inequality is valid

$$\sum_{e\in\delta(S)}\sum_{r\in\Omega}\Omega(e,r)\lambda^*_r \geq \max\{\lceil\frac{\sum_{i\in S} q_i}{Q}\rceil, \lceil\frac{\sum_{i\in S}\nu_i}{A}\rceil\}$$

The well-known CVRPSEP package (Lysgaard et al. 2004) is applied to separate the RCIs.

The ISI is a family of inequalities proposed by Côté et al. (2020). For a set of customers S whose items are not packable in a vehicle, then the following inequality is valid

$$\sum_{e\in\delta(S)}\sum_{r\in\Omega}\Omega(e,r)\lambda^*_r \geq 2$$

To identify ISIs, the separation algorithm in Zhang et al. (2021a) is used.

Other classic inequalities such as comb inequalities (Lysgaard et al. 2004) are not introduced since they are not cost-effective in the BPC algorithm according to the preliminary experiments.
The branching strategy

The classic branching strategy, branching on edge, is applied because it does not alter the structure of the pricing problem. It is acknowledged that choosing on which variable to branch is highly critical to the efficiency of the branch-and-bound algorithm (Achterberg et al. 2005). Hence, a novel variable selection strategy is proposed to determine which edge (variable) is to branch upon. With the *IL-Trie* storing the infeasible routes that have been checked, the number of occurrences of each edge in the *IL-Trie* is counted. Among those edges with a fractional amount of flow, the one with the largest number of occurences is selected. The intuition behind the strategy is as follows. The *IL-Trie* maintains a set of routes that are promising in terms of the negative reduced cost but infeasible with respect to the loading constraints. Hence, branch $x_e \leq 0$ is likely to produce a high lower bound that is higher than the incumbent primal bound while the other branch $x_e \geq 1$ is also prone to increase the lower bound since edge e is likely to generate infeasible paths. Further experiments illustrate that this novel variable selection strategy effectively reduces the size of the tree and accelerates the convergence of the BPC algorithm.

6 Computational experiments and analysis

In this section, we report the computational results on the benchmark instances (Iori et al. 2007) for both the versions with and without the LIFO constraint. Managerial insights about the cost of respecting the LIFO constraint are thus derived. We also compare the novel variable selection strategy against a conventional one. All the computational experiments were carried out on a Windows computer equipped with an Intel i5-10600KF processor (4.10 GHz). The BPC algorithm was implemented in C++ using the CVRPSEP (Lysgaard et al. 2004) library and CPLEX 12.9 as the LP solver. CPU times reported throughout the experiments are in seconds if not specified.

6.1 The instances

All the instances tested were created by Iori et al. (2007). To make the study self-contained, some important details regarding the instances are described in this section. Table 2 lists the detailed settings of the four families of the benchmark instances. Instances of *family 1* is omitted since this family is equivalent to the classical CVRP.

Column *Vertical*, *Homogeneous* and *Horizontal* represents three different types of items, respectively: an item whose height is statistically greater than its width, an item whose height is statistically equal to its width and an item whose height is statistically less than its width. What can be observed in Table 2 is that *family 2* is featured with the largest items on average while *family 5* is featured with the smallest items on average. It can be expected that instances with large items tend to have more infeasible paths in terms of the LIFO constraint because there is less space to adjust. By contrast, instances with very small items are less likely to have infeasible paths. As a result, instances of *family 2,3* and *4* are solved by the proposed BPC algorithm while instances from *family 5* are solved by a simple variant of the BPC algorithm, in which the loading constraints are moved from the pricing problem to the master problem. By doing so, the computational complexity of the pricing problem is strikingly reduced with very little loss on the tightness of the lower bound. Details on the algorithm is presented in Appendix B.

Table 2: Settings for generating items

		Vertical		Homogeneous		Horizontal	
Instance family	$\lvert M_i \rvert$	h_{im}	w_{im}	h_{im}	w_{im}	h_{im}	w_{im}
2	[1,2]	[4H/10, 9H/10]	[W/10, 2W/10]	[2H/10, 5H/10]	[2W/10, 5W/10]	[H/10, 2H/10]	[4W/10, 9W/10]
3	[1,3]	[3H/10, 8H/10]	[W/10, 2W/10]	[2H/10, 4H/10]	[2W/10, 4W/10]	[H/10,2H/10]	[3W/10, 8W/10]
4	[1,4]	[2H/10, 7H/10]	[W/10, 2W/10]	[H/10, 4H/10]	[W/10, 4W/10]	[H/10,2H/10]	[2W/10, 7W/10]
5	[1,5]	[H/10, 6H/10]	[W/10, 2W/10]	[H/10, 3H/10]	[W/10, 3W/10]	[H/10,2H/10]	[W/10, 6W/10]

6.2 Solved open instances

The computational results of the newly-solved instances are reported in Table 3. The BPC algorithm is compared with the B&C algorithm in Zhang et al. (2021a). For both algorithms, 86,400 minutes (24 hours) and 8 Gigabyte are set as the time limit and the memory limit, respectively. Both algorithms used the same initial upper bounds from Côté et al. (2020).

Table 3: Comparison between the two algorithms on the newly-solved instances

Instance	No C	No I	The BPC			The B&C				
			Opt	Time	Tree size	UB	LB	Gap	Time	Tree size
1102	29	43	708.0	34382.7	736	708.0	691.0	2.46%	4,139.0$^+$	285,831
1402	32	47	1241.0	16104.0	207	1241.0	1218.6	1.83%	4,741.0$^+$	77,738
1502	32	48	1098.0	40461.0	328	1101.0	1080.3	1.91%	5,098.0$^+$	82,310
1702	40	60	851.0	11.56	0	851.0	839.1	1.42%	13,658.0$^+$	44,765
1802	44	66	1041.0	62449.0	585	1044.0	1005.5	3.83%	4,560.0$^+$	77,866
1803	44	87	1085.0	6326.0	31	1101.0	1061.4	3.73%	4,752.0$^+$	94,114
1902	50	82	774.0	7928.0	122	776.0	732.7	5.90%	9,109.0$^+$	60,052
1903	50	103	782.0	3601.0	62	782.0	749.8	4.29%	11,243.0$^+$	52,220
1904	50	134	793.0	1551.4	48	800.0	780.5	2.49%	8,625.45.0$^+$	61,542
2004	71	178	538.0	5173.0	51	555.0	527.9	5.22%	13,117.4$^+$	42,823
2104	75	168	1021.0	62420.0	362	1038.0	991.2	4.72%	29,831.0$^+$	52,495
2402	75	124	1193.0	17661.0	143	1193.0	1057.0	12.86%	34,048.9$^+$	23,909
2404	75	195	1111.0	7152.0	72	1115.0	1038.7	7.34%	28,526.7$^+$	39,490
2405	75	215	1044.0	3072.0	8051	1047.0	1018	2.84%	37,472.9$^+$	56,760

+ Out of memory

In Table 3, column *No C* gives the number of customers. Column *No I* gives the total number of items in the instance. Columns *Opt*, *Time*, *Tree size*, *UB*, *LB*, *Gap* gives the optimal objective value, the CPU time, the number of nodes explored, the global upper bound, the global lower bound, and the percentage gap between the upper bound and the lower bound, respectively. The BPC algorithm is compared with the state-of-the-art exact algorithm for $2|RO|L$ from Zhang et al. (2021a). There are two observations from the table.

1. Instances with 50 customers or less except for *1503* are all closed. In terms of *family*, before our study, the largest size of solvable instances in *family 2* was 35 customers and 56 items (*1602*). Now, the BPC algorithm can solve instances of *family 2* with up to 75 customers and 124 items. For *family 3*, the largest solvable instance had 40 customers and 73 items (*1703*) while the BPC algorithm improves the solvable size up to 50 customers and 103 items. With respect to *family 4*, the largest solvable network used to have 44 customers and 112 items. The BPC algorithm increases the solvable size up to 75 customers and 195 items. As for *family 5*, the landscape is not changed significantly, as we go from instances with 71 customers to instances with 75 customers.

2. The tree size of the BPC algorithm is significantly less than that of the B&C algorithm, because the linear relaxation of the set partitioning formulation is much tighter than the two-index formulation. Economical usage of memory enables the algorithm to run long enough to prove the optimality.

The full results on the benchmark instance are provided in Appendix C, which illustrates that all the instances optimally solved by the B&C algorithm can also be solved by the BPC algorithm. However, a drawback of the BPC algorithm is that it has to invoke the exact checker for much more times than the B&C algorithm due to the primal property of the CG algorithm. As a result, the BPC algorithm needs much more CPU time on some instances, especially on instances with less than 35 customers. Nevertheless, as the size of the instance grows, the speed of the BPC algorithm becomes more competitive. Furthermore, because CG-based algorithms are particularly good at solving instances with many vehicles, the BPC algorithm runs significantly faster on instances *1703,1704,1705* than the B&C algorithm.

6.3 The effectiveness of the *IL-Trie* on root nodes

Two groups of experiments studying the effectiveness of the *IL-Trie* were carried out as well. In the first group, we compare the CG algorithm (with the valid inequalities) with and without the

IL-Trie in terms of solving root nodes. The detailed results are reported in Table 4. Column *T-Imp %* provides the percentage improvement of the CG with the *IL-Trie* over the CG algorithm without the *IL-Trie*. On average, the *IL-Trie* can save 36.98% of CPU time to solve root nodes for the CG algorithm. The most significant impact occurs for *family 2*, where the average saving reaches up to 55.63%. On the other hand, the *IL-Trie* has less effect on other families of instances. It can be explained by the fact that instances with larger items tend to have infeasible routes, so much more extensions can be prohibited by the use of *IL-Trie*.

Table 4: The effectiveness of the *IL-Trie*

Instance	CG without the *IL-Trie*	CG with the *IL-Trie*	T-Imp %
	Time	Time	
1102	564.1	167.7	70.28%
1402	407.2	200.0	50.87%
1502	1111.2	300.8	72.92%
1702	5.4	6.6	-21.56%
1802	3980.5	849.3	78.66%
1803	345.2	313.8	9.09%
1902	319.4	163.4	48.85%
1903	317.4	271.8	14.36%
1904	177.3	190.5	-7.39%
2004	729.5	488.3	33.06%
2402	4172.7	443.6	89.37%
2404	446.5	423.2	5.23%
Average	1048.1	318.3	36.98%

In the second group, the impact of the *IL-Trie* on the *no-loading property* of the pricing algorithm (PA) is investigated. The concept of the *no-loading property* is defined as the quality of the solution derived by the PA before considering the loading constraints while performing label extensions, i.e., only running the hierarchical labeling routine including Versions ①, ②, ③. It is a very important indicator to evaluate the PA, because after Version ③, the number of labels increases exponentially due to the consideration of the loading constraints. Better optimal objective values obtained before invoking Version ④ implies that fewer labels lead to negative reduced cost. As a result, more labels are bounded by the completion bounds so that the number of generated labels even considering the loading constraints can be controlled at a low level. We present the *no-loading property* of the PA without the *IL-Trie* and with the *IL-Trie* in Table 5. Column *Obj* gives the optimal objective value of the *LPM* while Column *Obj-Red %* gives the percentage reduction of the optimal objective value derived by the PA with the *IL-Trie* on the value derived by the PA without the *IL-Trie*. On average, the percentage reduction is 1.55%. For the majority of the instances, the *IL-Trie* enables the PA algorithm to obtain better objective values before involving the loading constraints. On the flip side, without the *IL-Trie*, the hierarchical labeling algorithm invokes Version ④ with a higher objective value, which could cause massive computational effort. One might argue that the PA with the *IL-Trie* took more CPU time, but the extra time pays off if compared to the amount of extra CPU time needed by the onward label extensions without the *IL-Trie*.

Table 5: The effectiveness of the *IL-Trie* on the *no-loading property*

Instance	PA without the *IL-Trie*		PA with the *IL-Trie*		Obj-Red %
	Obj	Time	Obj	Time	
1102	703.1	65.0	682.7	116.6	-2.90%
1402	1234.6	43.5	1227.9	124.9	-0.54%
1502	1101.8	62.3	1079.2	149.7	-2.04%
1702	851.0	2.8	851.0	3.4	0.00%
1802	1069.3	104.7	1026.3	334.9	-4.02%
1803	1085.8	70.0	1078.8	161.3	-0.65%
1902	774.3	127.1	763.9	153.5	-1.36%
1903	778.9	181.2	774.6	237.2	-0.55%
1904	784.9	138.4	783.5	151.0	-0.18%
2004	546.1	526.0	537.0	488.3	-1.71%
2104	1014.1	192.5	1013.0	317.7	-0.10%
2402	1254.3	189.7	1181.6	423.5	-5.80%
2404	1109.0	222.7	1105.2	426.6	-0.34%
Average	946.7	148.2	931.2	237.6	-1.55%

6.4 The impact of the new variable selection strategy

A group of computational analyses is also conducted for the new variable selection strategy (VSS). We evaluate a VSS by the global lower bound obtained after exploring a certain number of nodes. To make a comparison, a classic VSS, which is choosing the fractional variable with the highest coefficient in the objective function, is adopted. The BPC algorithm is terminated when the size of the tree is greater than 100 nodes. The computational results are shown in Table 6. Column *LB* gives the global lower bounds. Column *LB-Imp %* gives the percentage improvement of the new VSS over the classical VSS. On all the newly-solved instances, the new VSS performs consistently better than the classical VSS in terms of the objective values. The amount of the enhancement by the new VSS is up to 0.26% on average. Although the new VSS generally consumes more CPU time, the extra CPU time is likely compensated by the tighter lower bounds if the BPC algorithm continues to solve the instances to optimality.

Table 6: The effectiveness of the new variable selection strategy

Instance	The classical VSS		The new VSS		LB-Imp %
	LB	Time	LB	Time	
1102	690.8	2668.8	695.6	2536.5	0.69%
1402	1234.4	4058.4	1236.1	5796.9	0.14%
1502	1087.94	5690.1	1091.0	7743.1	0.28%
1702	851.0	7.9	851.0	9.3	0.00%
1802	1032.5	10415.7	1032.6	10733.9	0.01%
1803	1084.1	8979.0	1085.0	6979.0	0.08%
1902	768.0	2334.6	772.1	2835.5	0.52%
1903	779.0	2492.1	781.2	4196.6	0.28%
1904	787.1	1543.3	793.0	1931.9	0.74%
2004	538.0	3603.3	538.0	5335.1	0.00%
2104	1016.5	5895.1	1017.7	11023.9	0.12%
2402	1186.9	6467.2	1190.5	8067.4	0.31%
2404	1108.5	3915.8	1110.6	6530.5	0.19%
Average	935.8	4467.7	938.1	5672.3	0.26%

6.5 The cost of respecting the LIFO constraint

Respecting the LIFO constraint, from the practical standpoint, is not always necessary except for some situations where unloading items of successive customers is technically impossible at the serving customer (e.g., driving a loaded excavator off the vehicle). For other situations, whether or not implementing the LIFO rule relies on the management strategy. Hence, it is insightful to study the cost of following the LIFO constraint, as the basis for practitioners to make relevant decisions.

The proposed BPC algorithm is run over the benchmark instances while relaxing the LIFO constraint, i.e., $2|UN|L$. The computational experiment is only performed on the instances with less than or equal to 50 customers since the vast majority of these instances are solved optimally. The results are summarized in Table 7. *MORL* stands for the mean optimal objective values with the LIFO constraint relaxed while Column *MOWL* represents the mean optimal objective values with the LIFO constraint. Column *Mean Obj-Diff %* gives the corresponding average relative percentage difference between the objective values. It can be observed that for instances of *family 2*, following the LIFO constraint produces 2.6% more cost on average. As the average size of the items is smaller, the cost due to the LIFO constraint gradually decreases. Especially for *family 5*, the LIFO constraint brings no extra cost.

For practitioners, according to Table 7, if the items are relatively large with respect to the vehicles, the LIFO constraint could significantly raise the operational cost. To address the issue, the following policies could provide some insights for managers.

1. **Hierarchize service level of delivery.** Generally, the LIFO constraint is respected to make speedy delivery for customers. In a distribution network, service level can be set in a hierarchical fashion. That is to say, only for customers that need speedy delivery, the LIFO constraint is respected. One can even charge for the speedy delivery as the compensation for the extra cost.

2. **Purchase vehicles with larger loading surface.** The policy can reduce the extra cost effectively as implied by Table 7. Although a large amount of investment is required at the very beginning, it can be covered by the saved extra cost in the long run. Moreover,

the average extra cost can be a good reference for the purchase.

Table 7: The results for the analysis on the cost of respecting the LIFO constraint

Instance family	MORL	MOWL	Mean Obj-Diff %
2	758.3	778.2	2.60%
3	739.5	749.4	1.10%
4	786.1	794.2	1.00%
5	777.4	777.4	0.00%

7 Conclusion

This study has developed a state-of-the-art exact method to solve the 2L-CVRP. The problem is formulated as a set partitioning model of which the linear relaxation is solved by a column generation algorithm boosted by a new data structure and a new dominance rule. The branch-and-price-and-cut algorithm is enhanced by a novel variable selection strategy and some existing valid inequalities. The algorithm solved 14 instances optimally for the first time. For the benchmark instances from *family 2*, *family 3* and *family 3*, the solvable size is substantially increased by the study. Besides methodological contributions, valuable managerial insights regarding the LIFO constraint is also derived. Future work will explore the effectiveness of the new data structure in meta-heuristics for the 2L-CVRP. Also, it is appealing to address variants of the 2L-CVRP with stochasticity.

Acknowledge

The authors are grateful to Professor Jean-Franqis Côté from Université Laval for the codes of the packing algorithms and valuable suggestions on the subject. This work was supported by the Canadian Natural Sciences and Engineering Research Council (NSERC).

References

Achterberg T, Koch T, Martin A (2005) Branching rules revisited. *Operations Research Letters* 33(1):42–54.

Alinaghian M, Zamanlou K, Sabbagh MS (2017) A bi-objective mathematical model for two-dimensional loading time-dependent vehicle routing problem. *Journal of the Operational Research Society* 68(11):1422–1441.

Annouch A, Bellabdaoui A, Minkhar J (2016) Split delivery and pickup vehicle routing problem with two-dimensional loading constraints. *2016 11th International Conference on Intelligent Systems: Theories and Applications (SITA)*, 1–6 (IEEE).

Baldacci R, Mingozzi A, Roberti R (2011) New route relaxation and pricing strategies for the vehicle routing problem. *Operations research* 59(5):1269–1283.

Brass P (2008) *Advanced data structures*, volume 193 (Cambridge University Press Cambridge).

Burke EK, Kendall G, Whitwell G (2004) A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research* 52(4):655–671.

Costa L, Contardo C, Desaulniers G (2019) Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science* 53(4):946–985.

Côté JF, Dell'Amico M, Iori M (2014a) Combinatorial benders' cuts for the strip packing problem. *Operations Research* 62(3):643–661.

Côté JF, Gendreau M, Potvin JY (2014b) An exact algorithm for the two-dimensional orthogonal packing problem with unloading constraints. *Operations Research* 62(5):1126–1141.

Côté JF, Gendreau M, Potvin JY (2020) The vehicle routing problem with stochastic two-dimensional items. *Transportation Science* 54(2):453–469.

Desaulniers G, Desrosiers J, Solomon MM (2006) *Column generation*, volume 5 (Springer Science & Business Media).

Desaulniers G, Desrosiers J, Spoorendonk S (2011) Cutting planes for branch-and-price algorithms. *Networks* 58(4):301–310.

Desaulniers G, Lessard F, Hadjar A (2008) Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science* 42(3):387–404.

Desrochers M, Desrosiers J, Solomon M (1992) A new optimization algorithm for the vehicle routing problem with time windows. *Operations research* 40(2):342–354.

Dominguez O, Guimarans D, Juan AA, de la Nuez I (2016a) A biased-randomised large neighbourhood search for the two-dimensional vehicle routing problem with backhauls. *European Journal of Operational Research* 255(2):442–462.

Dominguez O, Juan AA, Barrios B, Faulin J, Agustin A (2016b) Using biased randomization for solving the two-dimensional loading vehicle routing problem with heterogeneous fleet. *Annals of Operations Research* 236(2):383–404.

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

Feillet D, Gendreau M, Rousseau LM (2007) New refinements for the solution of vehicle routing problems with branch and price. *INFOR: Information Systems and Operational Research* 45(4):239–256.

Fischetti M, González JJS, Toth P (1995) Experiments with a multi-commodity formulation for the symmetric capacitated vehicle routing problem. *in Proceedings of the 3rd Meeting of the EURO Working Group on Transportation, 169–173* (Citeseer).

Fuellerer G, Doerner KF, Hartl RF, Iori M (2009) Ant colony optimization for the two-dimensional loading vehicle routing problem. *Computers & Operations Research* 36(3):655–673.

Gendreau M, Iori M, Laporte G, Martello S (2006) A tabu search algorithm for a routing and container loading problem. *Transportation Science* 40(3):342–350.

Gendreau M, Iori M, Laporte G, Martello S (2008) A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks: An International Journal* 51(1):4–18.

Guimarans D, Dominguez O, Panadero J, Juan AA (2018) A simheuristic approach for the two-dimensional vehicle routing problem with stochastic travel times. *Simulation Modelling Practice and Theory* 89:1–14.

Hokama P, Miyazawa FK, Xavier EC (2016) A branch-and-cut approach for the vehicle routing problem with loading constraints. *Expert Systems with Applications* 47:1–13.

Hong S, Zhang D, Lau HC, Zeng X, Si YW (2014) A hybrid heuristic algorithm for the 2d variable-sized bin packing problem. *European Journal of Operational Research* 238(1):95–103.

Iori M, Salazar-González JJ, Vigo D (2007) An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation science* 41(2):253–264.

Kang J, Park S (2003) Algorithms for the variable sized bin packing problem. *European Journal of Operational Research* 147(2):365–372.

Khebbache-Hadji S, Prins C, Yalaoui A, Reghioui M (2013) Heuristics and memetic algorithm for the two-dimensional loading capacitated vehicle routing problem with time windows. *Central European Journal of Operations Research* 21(2):307–336.

Leung SC, Zhou X, Zhang D, Zheng J (2011) Extended guided tabu search and a new packing algorithm for the two-dimensional loading vehicle routing problem. *Computers & Operations Research* 38(1):205–215.

Lysgaard J, Letchford AN, Eglese RW (2004) A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming* 100(2):423–445.

Mahvash B, Awasthi A, Chauhan S (2017) A column generation based heuristic for the capacitated vehicle routing problem with three-dimensional loading constraints. *International Journal of Production Research* 55(6):1730–1747.

Martello S, Vigo D (1998) Exact solution of the two-dimensional finite bin packing problem. *Management science* 44(3):388–399.

Martinelli R, Pecin D, Poggi M (2014) Efficient elementary and restricted non-elementary route pricing. *European Journal of Operational Research* 239(1):102–111.

Pinto T, Alves C, de Carvalho JV (2015) Variable neighborhood search for the elementary shortest path problem with loading constraints. *International Conference on Computational Science and Its Applications*, 474–489 (Springer).

Pinto T, Alves C, de Carvalho JV (2016) A branch-and-price algorithm for the vehicle routing problem with 2-dimensional loading constraints. *International Conference on Computational Logistics*, 321–336 (Springer).

Pollaris H, Braekers K, Caris A, Janssens GK, Limbourg S (2015) Vehicle routing problems with loading constraints: state-of-the-art and future directions. *OR Spectrum* 37(2):297–330.

Righini G, Salani M (2008) New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal* 51(3):155–170.

Song X, Jones D, Asgari N, Pigden T (2019) Multi-objective vehicle routing and loading with time window constraints: a real-life application. *Annals of Operations Research* 1–27.

Tarantilis CD, Zachariadis EE, Kiranoudis CT (2009) A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. *IEEE Transactions on Intelligent Transportation Systems* 10(2):255–271.

Wei L, Zhang Z, Zhang D, Leung SC (2018) A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research* 265(3):843–859.

Zachariadis EE, Tarantilis CD, Kiranoudis CT (2009) A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research* 195(3):729–743.

Zhang X, Chen L, Gendreau M, Langevin A (2021a) A branch-and-cut algorithm for the vehicle routing problem with two-dimensional loading constraints .

Zhang X, Chen L, Gendreau M, Langevin A (2021b) Learning-based branch-and-price algorithms for a vehicle routing problem with time windows and two-dimensional loading constraints .

A Proofs for *L-Trie*

Statement 2.

Proof. We prove the statement by contradiction. Suppose we are given a route $r = [i_1, i_2, ..., i_n]$ and the root node γ of the *L-Trie*. The route found by *L-Trie::findSubRoute(r,γ)* is $r' =$

$[i_{k_1}, i_{k_2}, ..., i_{k_m}]$ and r' is not a sub-route. By Definition 1, we can deduce that $1 \le k_1 < k_2, ..., k_m \le n$ is not true otherwise r' would be a valid sub-route. However, the function *findPos* in Algorithm 5 ensures that the found route must respect $1 \le k_1 < k_2, ..., k_m \le n$, which achieves the contradiction. □

Statement 3.

Proof. We prove the statement by contradiction. Suppose we are given a route $r = [i_1, i_2, ..., i_n]$. There exists a sub-route $r' = [i_{k_1}, i_{k_2}, ..., i_{k_m}]$ stored in the *L-Trie*. *L-Trie::findSubRoute(r, γ)* fails to find r'. Suppose the algorithm finds node "$i_{k_{m-1}}$" as shown in Figure 7, then it will continue to search node "i_{k_m}" according to Algorithm 5. By recursion, nodes $i_{k_1}, i_{k_2}, ..., i_{k_m}$ must be all visited. Then the only possible reason that can justifies the failure to find r' is that the algorithm ends at some children of node "i_{k_m}". Because r' is the only sub-route of r, so no child of node "i_{k_m}" leads to a sub-route. Eventually, *Line 6* in Algorithm 5 is performed, which is to return *True* since r' ends at node "i_{k_m}". Hence, the contradiction is achieved. □

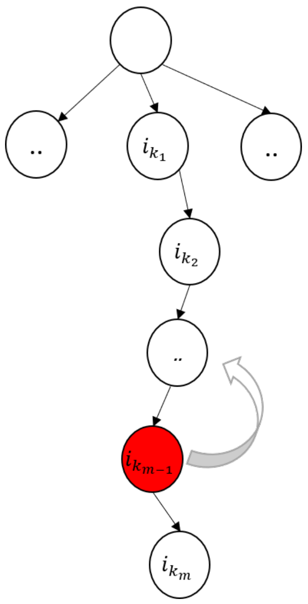


Figure 7: Interpretation of the proof for Statement 3

Statement 4.

Proof. We prove the statement by contradiction. Suppose we are given a route $r = [i_{k_1}, i_{k_2}, ..., i_{k_m}]$ and the root node of the *L-Trie* is γ. *L-Trie::findMasterRoute(r, γ)* finds route $r' = [i_1, i_2, ..., i_n]$ and r' is not a master-route of r. By Definition 2, there are two cases: i) r' includes some customer that is not visited in r; ii) all customers in r' are visited in r as well, but $1 \le k_1 < k_2, ..., k_m \le n$ is not true. The first case is prohibited by *Line 5* in Algorithm 6. As for the second case, it is also impossible because the search is performed in the order of route r as shown in *Line 8*. Hence, the contradiction is achieved. □

Statement 5.

Proof. Suppose we are given a route $r = [i_{k_1}, i_{k_2}, ..., i_{k_m}]$ and the root node of the *L-Trie* is γ. $r' = [i_1, i_2, ..., i_n]$ is the only master-route of r stored in the *L-Trie*. As shown by Figure 8, once Algorithm 6 starts to explore node "i_1", it will eventually reach node i_{k_m}, which exactly corresponds to route r'. Since node "i_1" is bound to be explored by Algorithm 6, r' can be surely found. □

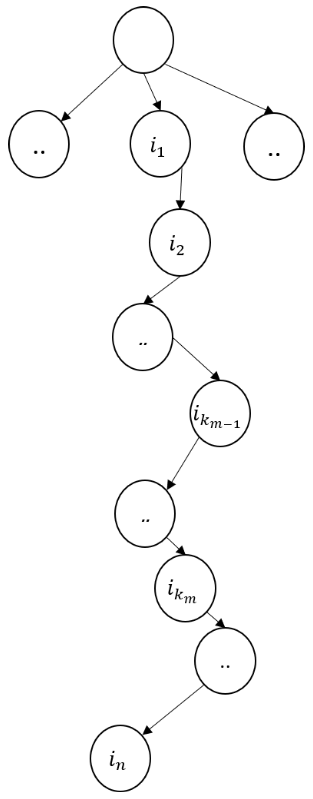


Figure 8: Interpretation of the proof for **Statement 5**

B The variant of the BPC

Generally, the simple variant of the BPC is the version where the two-dimensional loading constraints are excluded in the pricing problem but moved to the master problem. Let $\Omega(e)$ be the set of routes that traverse edge e. Formally, the resulting set partitioning formulation and the pricing problem are as follows.

$$\min \sum_{r \in \Omega} c_r \lambda_r \tag{26}$$

$s.t.$

$$\sum_{r \in \Omega} \lambda_r = |K| \tag{27}$$

$$\sum_{r \in \Omega} a_{i,r} \lambda_r = 1, \ \forall i \in V_c \tag{28}$$

$$\sum_{e \in E(S,\sigma)} \sum_{r \in \Omega(e)} \lambda_r \leq |S| - 1, \ \forall (S,\sigma) \text{ such that } \sigma \notin \Sigma(S) \tag{29}$$

$$\lambda_r \in \{0,1\}, \ \forall r \in \Omega \tag{30}$$

$$\min \sum_{e \in E} \bar{d}_e x_e - \pi_f \tag{31}$$

$$s.t. \tag{32}$$

$$\sum_{e \in \delta(i)} x_e = 2 \quad \forall i \in V \tag{33}$$

$$\sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V_c,\ 1 < |S| < n-1,\ \forall i \in S,\ \forall k \in K \tag{34}$$

$$\sum_{(i,j) \in E} x_{ij}(q_i + q_j) \leq 2Q \tag{35}$$

$$\sum_{(i,j) \in E} x_{ij}(\nu_i + \nu_j) \leq 2A \tag{36}$$

$$x_e \in \{0,1\} \forall e \in E \tag{37}$$

Hence, the pricing problem is the classic ESPPRC, which is solved by the dynamic programming proposed in Martinelli et al. (2014). Other components of the BPC algorithm are not changed.

C Full results of the BPC algorithm on the benchmark instances

Instance	No C	No I	The BPC			The B&C				
			Opt	Time	Tree size	UB	LB	Gap	Time	Tree size
0102	15	24	285.0	65.3	3	285.0	285.0	0.0%	1.9	314
0103	15	31	280.0	43.4	0	280.0	280.0	0.0%	1.9	97
0104	15	37	288.0	15.0	0	288.0	288.0	0.0%	0.3	0
0105	15	45	279.0	0.8	0	279.0	279.0	0.0%	0.2	0
0202	15	25	342.0	6.3	10	342.0	342.0	0.0%	0.2	71
0203	15	31	347.0	7.9	6	347.0	347.0	0.0%	0.6	127
0204	15	37	336.0	16.5	23	336.0	336.0	0.0%	0.5	59
0205	15	48	329.0	0.3	3	329.0	329.0	0.0%	0.03	11
0302	15	37	396.0	7.5	2	396.0	396.0	0.0%	0.3	148
0303	20	46	387.0	13.5	0	387.0	387.0	0.0%	0.6	18
0304	20	46	374.0	16.3	5	374.0	374.0	0.0%	3.9	36
0305	20	49	369.0	0.5	0	369.0	369.0	0.0%	0.15	9
0402	20	32	434.0	8.3	0	434.0	434.0	0.0%	0.6	70
0403	20	46	432.0	8.3	2	432.0	432.0	0.0%	0.8	53
0404	20	50	438.0	14.1	0	438.0	438.0	0.0%	0.8	151
0405	20	62	423.0	0.3	0	423.0	423.0	0.0%	0.1	2
0502	21	31	380.0	20.3	2	380.0	380.0	0.0%	0.6	24
0503	21	37	373.0	28.8	0	373.0	373.0	0.0%	0.2	7
0504	21	41	377.0	27.4	0	377.0	377.0	0.0%	0.5	13
0505	21	57	389.0	0.2	0	389.0	389.0	0.0%	0.1	0
0602	21	33	491.0	16.5	0	491.0	491.0	0.0%	0.5	79
0603	21	40	496.0	17.2	3	496.0	496.0	0.0%	3.3	234
0604	21	57	489.0	31.4	0	489.0	489.0	0.0%	0.5	3
0605	21	56	488.0	1.2	4	488.0	488.0	0.0%	0.2	40
0702	22	32	724.0	27.5	0	724.0	724.0	0.0%	0.5	24
0703	22	41	698.0	39.9	0	698.0	698.0	0.0%	1.4	7
0704	22	51	714.0	50.8	0	714.0	714.0	0.0%	1.1	6

0705	22	55	742.0	4.7	9	742.0	742.0	0.0%	0.8	0
0802	22	29	720.0	49.7	0	720.0	720.0	0.0%	5.6	342
0803	22	45	730.0	32.1	0	730.0	730.0	0.0%	3.0	10
0804	22	48	701.0	38.9	0	701.0	701.0	0.0%	0.3	0
0805	22	52	721.0	0.6	0	721.0	721.0	0.0%	0.8	0
0902	25	40	612.0	4.0	0	612.0	612.0	0.0%	0.3	26
0903	25	61	615.0	7.9	0	615.0	615.0	0.0%	0.7	41
0904	25	63	626.0	66.3	8	626.0	626.0	0.0%	3.5	331
0905	25	91	609.0	0.9	0	609.0	609.0	0.0%	0.17	18
1002	29	43	687.0	124.1	2	687.0	687.0	0.0%	4.7	567
1003	29	49	637.0	279.6	4	637.0	637.0	0.0%	17.9	1,316
1004	29	72	738.0	817.2	37	738.0	738.0	0.0%	4.7	12
1005	29	86	704.0	21.2	13	704.0	704.0	0.0%	13.6	64
1102	29	43	708.0	34,382.7	736	708.0	691.0	2.5%	4,139.5	285,831
1103	29	62	743.0	749.4	46	743.0	743.0	0.0%	83.7	18,651
1104	29	74	781.0	1,893.3	49	781.0	781.0	0.0%	104.1	2,583
1105	29	91	681.0	33.2	3	681.0	681.0	0.0%	79.7	5
1202	30	50	605.0	41.3	0	605.0	605.0	0.0%	405.2	1,1631
1203	30	56	596.0	99.0	6	596.0	596.0	0.0%	145.2	5,624
1204	30	82	606.0	22.8	2	606.0	606.0	0.0%	901.3	19,990
1205	30	101	596.0	4.7	6	596.0	596.0	0.0%	166.7	5,208
1302	30	101	2,714.0	444.8	16	2,714.0	2,714.0	0.0%	40.8	7,284
1303	32	56	2,574.0	229.2	8	2,574.0	2,574.0	0.0%	23.5	2,202
1304	32	78	2,668.0	1,435.9	54	2,668.0	2,668.0	0.0%	175.6	2,3445
1305	32	102	2,632.0	55.8	5	2,632.0	2,632.0	0.0%	94.1	4
1402	32	47	1,241.0	16,104.0	207	1,241.0	1,218.6	1.83%	4,740.8	77,738
1403	32	57	1,190.0	5,592.5	57	1,190.0	1,190.0	0.0%	65.4	5,474
1404	32	65	1,166.0	464.3	0	1,166.0	1,166.0	0.0%	2.7	17
1405	32	87	1,288.0	1.4	0	1,288.0	1,288.0	0.0%	0.7	27
1502	32	48	1,098.0	40,461	328	1,101.0	1,080.3	1.91%	5,098.2	82,310
1504	32	84	1,348.0	3,027.3	64	1,348.0	1,348.0	0.0%	163.0	7,550
1505	32	114	1,335.0	24.3	0	1,335.0	1,335.0	0.0%	6,756.8	128
1602	35	56	682.0	9.0	3	682.0	682.0	0.0%	15.9	533

1603	35	74	682.0	12.3	2	682.0	682.0	0.0%	17.1	654
1604	35	93	691.0	17.2	0	691.0	691.0	0.0%	33.2	1,492
1605	35	114	682.0	1.6	2	682.0	682.0	0.0%	16.6	651
1702	40	60	851.0	11.5	0	851.0	839.1	1.42%	13,658.7	44,765
1703	40	60	842.0	7.6	0	842.0	842.0	0.0%	5,860.6	53,227
1704	40	60	842.0	7.9	0	842.0	842.0	0.0%	5,291.2	50,236
1705	40	60	842.0	2.6	0	842.0	842.0	0.0%	5,675.7	51,226
1802	44	66	1,041.0	62,449.0	585	1,044.0	1,005.5	3.83%	4,560.3	77,866
1803	44	87	1,085.0	6326.0	31	1101.0	1061.4	3.73%	4,752.0	94,114
1804	44	112	1,113.0	2,334.8	17	1,113.0	1,113.0	0.0%	2,167.2	31,823
1805	44	122	937.0	364.8	114	937.0	937.0	0.0%	4,880.9	612
1902	50	82	774.0	7,928.0	122	776.0	732.7	5.9%	9,109.7	60,052
1903	50	103	782.0	3,601.0	62	782.0	782.0	4.3%	11,243.0	52,220
1904	50	134	793.0	1,551.4	48	800.0	780.6	2.49%	8,625.4	61,542
1905	50	157	724.0	136.0	47	724.0	724.0	0.0%	1,029.7	115
2004	71	178	538.0	5,173.0	51	555.0	527.9	5.22%	13,117.4	42,823
2005	71	226	505.0	3,623.1	264	505.0	505.0	0.0%	1,865.2	7,082
2104	75	168	1021.0	62,420.0	362	1,038.0	991.2	4.72%	29,831.0	52,495
2402	75	124	1,193.0	17,661.0	143	1,193.0	1,057.0	12.86%	34,048.9	23,909
2404	75	195	1,111.0	7,152.0	1,470	1,115.0	1,038.75	7.34%	28,526.7	39,490
2405	75	215	1,044.0	16,100	30,72	1,047.0	1,018.05	2.84%	37,472.9	56,760