CIRRELT
1971 – 2021

CIRRELT-2021-26

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

Xiangyi Zhang
Lu Chen
Michel Gendreau
André Langevin

July 2021

Bureau de Montréal
Université de Montréal
C.P. 6128, succ. Centre-Ville
Montréal (Québec) H3C 3J7
Tél : 1-514-343-7575
Télécopie : 1-514-343-7121

Bureau de Québec
Université Laval,
2325, rue de la Terrasse
Pavillon Palasis-Prince, local 2415
Québec (Québec) G1V 0A6
Tél : 1-418-656-2073
Télécopie : 1-418-656-2624

A Branch-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. In this study, we develop a branch-and-cut algorithm for the vehicle routing problem with two-dimensional loading constraints. A separation algorithm is proposed to simultaneously identify infeasible set inequalities and weak capacity in equalities for fractional solutions. Classic capacitated vehicle routing problem (CVRP) inequalities and a strong-branching-like strategy are adopted as well. A branch-and-cut (B&C) algorithm is built to solve the problem. Experimental results illustrate that the algorithm is competitive. In particular, we solve 6 instances to optimality for the first time. Extensive computational analysis is also conducted to reveal the impact of infeasible set inequalities, the branching strategy, and the packing algorithms on the B&C algorithm

Keywords. Routing, two-dimensional loading constraints, branch and cut, contraction algorithm

Acknowledgements. This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC). Computing facilities were provided by Compute Canada. These supports are gratefully acknowledged.

---

* Corresponding author: xiangyi.zhang@polymtl.ca

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

1 Introduction

Vehicle routing and items-loading are considered as two basic elements for logistics. Making decisions on either of them usually means dealing with NP-hard problems. Nevertheless, the appealing improvement on operational performance (Côté et al. 2017) has motivated many researchers to jointly solve the two problems. A representative integration in this field is the capacitated vehicle routing problem with two-dimensional loading constraints, denoted as 2L-CVRP, which was proposed and addressed by Iori et al. (2007) for the first time. It can be briefly described as follows: given a homogeneous fleet of vehicles and a set of customers with rectangular items to transport, find routes with minimal total traveling distance for the fleet to fulfill the customers such that items associated with a route should be packed in a vehicle while respecting the loading constraints specified by a set of restrictions. Different settings of the loading constraints lead to different versions of the 2L-CVRP. Fuellerer et al. (2009) proposed a taxonomy of the problem. We use the notation "$2|\alpha|L$", where α specifies the restrictions, to denote a problem while *2L-CVRP* represents the problem family.

The parameter α generally specifies two aspects: 1) item orientation; 2) rear loading. The former determines if an item can be rotated or not. In practice, such rotation is not allowed in some cases such as pallets unloaded by forklifts while, in other cases, the restriction is relaxed. The latter imposes the loading sequence to be compatible with the visiting order of customers. In other words, when serving a customer, one does not need to rearrange items belonging to later customers. Such restriction is also called the *last-in-first-out (LIFO)* constraint. Different settings of α yield four variants of the 2L-CVRP.

- $2|RO|L$: rear-loading and oriented items;
- $2|UO|L$: unrestricted loading and oriented items;
- $2|RN|L$: rear-loading and non-oriented items;
- $2|UN|L$: unrestricted loading and non-oriented items.

Over the past decade, a number of papers have worked on the 2L-CVRP and its variants (Pollaris et al. 2015). There are mainly two research streams: 1) developing algorithms for the 2L-CVRP; 2) addressing the practical variants. Our study is in the former stream, focusing on methodology. According to Pollaris et al. (2015), the vast majority of the algorithms developed for the 2L-CVRP are heuristics. To the best of our knowledge, only four studies are related to exact algorithms for the 2L-CVRP (Iori et al. 2007, Hokama et al. 2016, Pinto et al. 2016, Côté et al. 2020). The method presented in Côté et al. (2020) outperforms all the previous exact algorithms but there is a shortage of experiments and analysis on the algorithm, because it is only a minor contribution in their work. Our study not only aims at developing a better exact algorithm for the 2L-CVRP, but also providing more extensive experiments and analysis on the proposed exact algorithm. In particular, we concentrate on $2|RO|L$. The main contribution of the article is five-fold.

1. A separation algorithm is proposed to identify infeasible set inequalities as well as weak capacity inequalities for fractional solutions;
2. A heuristic is developed to lift the infeasible-path inequality, strengthening the linear relaxation;
3. A B&C algorithm is developed based on the above two contributions, integrating classic CVRP valid inequalities and a strong-branching-like strategy;
4. Extensive computational experiments are carried out to perform a more complete analysis compared to Côté et al. (2020);

5. New best lower bounds and upper bounds are provided and 6 open instances are closed.

The remainder of the paper is organized as follows. Section 2 reviews the related works, especially exact algorithms for the 2L-CVRP. Section 3 presents the main problem and the mathematical formulation. Section 4 introduces the separation problem and the valid inequalities. Section 5 presents the separation routines and the lifting heuristic. Section 6 describes other elements of the branch-and-cut algorithm. Section 7 reports the computational experiments and the analyses. Section 8 gives conclusions and future research directions.

2 Literature review

In this section, we first review relevant problems over the past five years. Secondly, existing algorithms for the 2L-CVRP are surveyed.

2.1 Relevant problems

Iori et al. (2007) is the first study to address the 2L-CVRP, specifically on $2|RO|L$. Gendreau et al. (2008) later on investigated both $2|RO|L$ and $2|UO|L$. Besides $2|RO|L$ and $2|UO|L$, two other two versions of the 2L-CVRP were then considered in Fuellerer et al. (2009). Real-life constraints were added to the 2L-CVRP to model practical problems in further studies. Since there is a thorough survey (Pollaris et al. 2015) on the literature before 2015, we merely review related problems that appeared during 2015 - 2020. Note that all the following problems reviewed are solved heuristically if not specified.

Dominguez et al. (2016b) studied $2|UO|L$ and $2|UN|L$ with a heterogeneous fleet motivated by a practical case from the construction industry. Dominguez et al. (2016a) worked on a $2|RN|L$ with cluster backhauls, where delivery and pick-up services are both considered. Along a route, the vehicle has to serve delivery points first and visit pick-up points later. Originating from the same company, Guimarans et al. (2018) addressed a $2|UN|L$ with stochastic travel times and penalty cost due to overtime. Alinaghian et al. (2017) considered time-dependent traveling time for $2|UO|L$ and optimized two objectives: total traveling distance and maximal weight load on vehicles. Zachariadis et al. (2017) introduced a variant of the 2L-CVRP in which vehicles can serve pick-up and delivery customers simultaneously rather than separately. The setting arises in reverse logistics of grocery store chains, where products need to be shipped from warehouse to various store locations while empty pallets and roll cages have to be collected. Annouch et al. (2016) studied a practical problem encountered in the natural gas industry. The operational background is transporting gas cylinders between filling stations and a set of client deposits. The problem has rich features including a heterogeneous fleet, time windows, multi-depot, and split delivery. Song et al. (2019) solved a problem from the food industry. Delivery is performed in the multi-compartment fashion because the loading area of a vehicle is split into parts that can carry multiple types of goods including shelf-stable, chilled, frozen food, vegetables & fruit, and kitchen-cleaning chemicals. Time window constraints are also involved due to the short shelf life of some food. Very recently, Côté et al. (2020) studied a $2|RO|L$ with stochastic items. That is to say, the width and the height of an item is not known until it is loaded. According to the paper, this problem was motivated by a practical application involving retailers of large appliances. The problem is solved by an exact L-shaped method.

Based on the above review, it can be observed that the 2L-CVRP and its variants are widely encountered in practice and solved heuristically. For this reason, it calls for more algorithmic explorations on the basic version i.e., the 2L-CVRP, hoping that better algorithms can be developed to solve the problems.

2.2 Development of algorithms

Iori et al. (2007) proposed the first exact algorithm for the 2L-CVRP. They adapted the classic two-index formulation of the CVRP and solved the problem by a B&C algorithm nested with a novel exact packing algorithm. The classic rounded capacity inequality with a modified right-hand-side is separated. When an integer solution is found, they invoke the exact packing procedure to perform the feasibility check. If the solution is feasible, then it is accepted as a candidate for the best global primal bound, otherwise a cut is added to eliminate the solution from the feasible region. A heuristic algorithm is also embedded in the B&C algorithm to identify upper bounds during the tree search. They created a set of benchmark instances for testing. On the test bed, the B&C algorithm can solve instances with up to 35 customers and 114 items. Hokama et al. (2016) proposed a new packing algorithm based on constraint programming techniques and added valid inequalities for the CVRP to improve the branch-and-cut algorithm from Iori et al. (2007). They achieved a substantial reduction on the CPU time required to reach optimal solutions or best-known solutions. Pinto et al. (2016) formulate $2|RO|L$ as a set partitioning model, which is solved by a column generation approach. The resulting pricing problem is an elementary shortest path problem with resource constraints and two-dimensional loading constraints. No exact dominance rule was found, so the entire algorithm runs in a heuristic fashion. To address the pricing problem, they proposed a variable neighborhood search. When a set of columns are priced out, a packing heuristic is invoked to check their feasibility. If a column is infeasible, a repairing mechanism is triggered. The repairing process for a column is terminated once the reduced cost of the column becomes non-negative. A branch-and-price algorithm was developed based on the column generation procedure. The authors observed that for small size instances, the branch-and-price algorithm is able to improve the initial upper bound very quickly, but finding an integer solution over a larger network (50 customers or more) is not trivial. Very recently, Côté et al. (2020) developed a branch-and-cut algorithm that solved many instances to optimality for the first time and significantly reduced CPU times. The breakthrough is attributed to a packing algorithm proposed earlier (Côté et al. 2014b) and a new family of valid inequalities. The algorithm is the state-of-the-art exact algorithm for $2|RO|L$.

As for meta-heuristics, there is a larger body of literature. Gendreau et al. (2008) proposed a Tabu search heuristic nested with packing heuristics and an exact packing algorithm for $2|RO|L$ and $2|UO|L$. The heuristic found the optimal solutions for most of the closed instances and identified feasible solutions for all the benchmark instances. Zachariadis et al. (2009), Leung et al. (2011) enhanced two different Tabu search heuristics, respectively, by hybridizing guided local search for the same problems. Fuellerer et al. (2009) employed an ant colony optimization algorithm for the four variants of the 2L-CVRP. To deal with the same problems, Wei et al. (2018) designed a simulated annealing heuristic based on a random local search packing heuristic and a new data structure for tracking checked routes. The algorithm can achieve best known solutions for most of the benchmark instances.

The 2L-CVRP has also encouraged studies on relevant packing algorithms. Checking if a route satisfies the two-dimensional loading constraints (for $2|RO|L$) is equivalent to solving a two-dimensional orthogonal packing problem with LIFO constraints (denoted as 2OPPL). The problem is strongly NP-hard. Most studies regarding 2OPPL are conducted under the background of vehicle routing. Heuristics for 2OPPL are usually based on the corresponding algorithms for 2OPP or strip packing problem (SPP) by considering visiting order (Gendreau et al. 2008, Leung et al. 2011, Wei et al. 2015). As for exact algorithms, Iori et al. (2007) proposed a branch-and-bound algorithm based on the algorithm from Martello et al. (2003). A branch-and-cut algorithm was developed by Côté et al. (2014b). The algorithm enumerates solutions for the one-dimensional contiguous bin packing problem, based on which an x-check procedure (Boschetti and Montaletti 2010) is performed to search a solution for the original problem. To our knowledge, this is by far the best exact algorithm for 2OPPL.

We are also aware of studies on three-dimensional loading constraints (Gendreau et al. 2006, Mahvash et al. 2017, Zhang et al. 2015), but these are not related to our study except Mahvash et al. (2017) who developed a column generation heuristic to deal with a VRP with three-dimensional loading constraints. Although the algorithm is not an exact one, their computational results illustrate that the heuristic can improve the best-known solutions for a number of benchmark instances.

3 Problem formulation

In this section, the formal description of $2|RO|L$ and a mixed integer linear programming model are presented.

3.1 Problem description

Let $G = (V, E)$ be an undirected graph, where V is the set of nodes $(0, 1, 2, ..., n)$ with node 0 representing the depot. $V_c = (1, 2, ..., n)$ denotes the set of customers. E is the set of edges linking nodes, with c_e denoting the traveling cost of edge e, $\forall e \in E$. An alternative representation of an edge is (i, j), $i, j \in V$. A set of homogeneous vehicles K is ready at the depot. Any vehicle $k \in K$ has a loading surface whose length and width equal 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 D.

Associated with each customer $i \in V_c$, there is a set of rectangular items M_i to be delivered to i. Each item m in set M_i has a specific width w_{im}, length h_{im} and weight d_{im}. We also respectively use a_i and d_i to denote the total area and total weights of items of customer i, where $a_i = \sum_{m \in M_i} w_{im} h_{im}$ and $d_i = \sum_{m \in M_i} d_{im}$. The number of items is the cardinality of set M_i. It is assumed that all the above input data are positive integers. The problem is to plan a route for each vehicle in fleet K to satisfy the demands of all customers such that the following constraints are respected:

1. Each customer must be visited exactly once by a single vehicle.
2. The total weight of the carried items must not exceed the weight capacity.
3. The items transported in a vehicle must not exceed the loading area.
4. Items must be positioned without being overlapped.
5. Items are not allowed to be rotated.
6. When unloading an item, items of customers served later cannot be moved (the LIFO constraint).

We also consider two conventions used in most articles on 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).

3.2 The two-index formulation

Given a node $i \in V$, $\delta(i)$ denotes the set of edges incident to it. Given $S \subset V_c$, $\delta(S)$ denotes the edges with only one endpoint in S. Let (S, σ) be the route defined by visiting the nodes of 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 be a binary variable for each $e \in E$ where $x_e = 1$ if edge

e is traversed by one of the vehicles and $x_e = 0$ otherwise. We can then formulate the problem as follows.

$$[P] \quad \min \sum_{e \in E} c_e x_e \tag{1}$$

$s.t.$

$$\sum_{e \in \delta(i)} x_e = 2, \ \forall i \in V_c \tag{2}$$

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

$$\sum_{e \in \delta(S)} x_e \geq 2r(S), \ \forall S \subseteq V_c, \ |S| \geq 2 \tag{4}$$

$$\sum_{e \in E(S,\sigma)} x_e \leq |S| - 1, \ \forall (S, \sigma) \text{ such that } \sigma \notin \Sigma(S) \tag{5}$$

$$x_e \in \{0, 1\}, \ \forall e \in E \tag{6}$$

Function (1) indicates that the objective is to minimize the total traveling distance. Constraint sets (2) and (3) indicate the degree on each node. Constraint set (4) is the classic rounded capacity inequality (RCI). Here $r(S) = \max\{\lceil \frac{\sum_{i \in S} d_i}{D} \rceil, \lceil \frac{\sum_{i \in S} a_i}{A} \rceil\}$ accounts for the loading area in addition to what is defined for the CVRP (Lysgaard et al. 2004). It is trivial to see that $r(S)$ is a valid lower bound on the number of vehicles required to serve customers in S. Constraint set (5) imposes the well-known infeasible-path inequalities. Constraint sets (4) and (5) are in exponential number, so the constraints are generated iteratively. In principle, once an infeasible path is identified, it implies the infeasibility of a group of paths. For example, if an infeasible route $2 \to 1 \to 3$ is identified, in theory, one could remove all the routes visiting nodes $2, 1, 3$ in that order but not necessarily contiguously, e.g. $5 \to 2 \to 4 \to 1 \to 6 \to 3$. This calls for adding an exponential number of cuts. However, there is little chance for such a cut to support the convex hull of integer solutions. Therefore, we only eliminate the identified path. Constraint set (6) imposes the integrality of the decision variables.

4 Valid inequalities

In this section, we introduce several valid inequalities to strengthen the model (2) - (6).

4.1 Weak capacity inequalities (WCI)

WCI share the same left-hand-side with RCI, but have a tighter right-hand-side that amounts to the minimal number of bins ($k(S)$) to contain items associated with set S. For a given set $S \subset V \setminus \{0\}$, $|S| \geq 2$, a WCI is defined as:

$$\sum_{e \in \delta(S)} x_e \geq 2k(S) \tag{7}$$

The validity of WCI is trivially established.

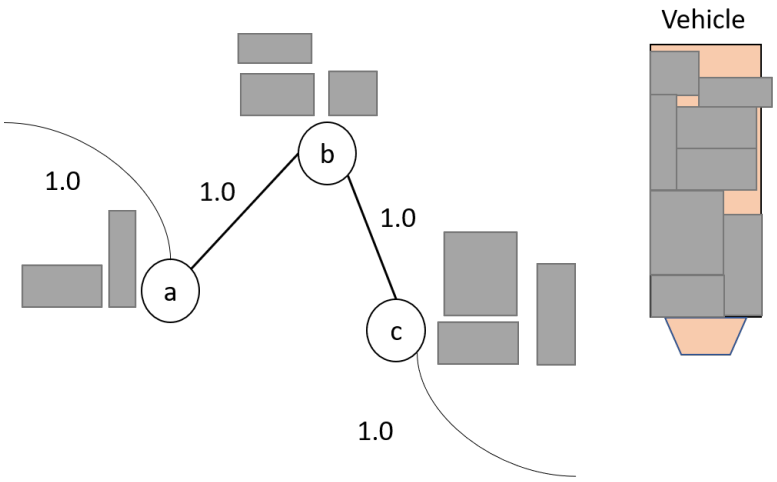

Figure 1: An example of violating ISI

4.2 Infeasible set inequalities (ISI)

For any set of customers $S \in V_c$, the following ISI (8) can be added to formulation P if items associated with S cannot be packed into a vehicle without consideration of the LIFO constraint.

$$\sum_{e \in \delta(S)} x_e \geq 2r'(S), \tag{8}$$

$$r'(S) = \begin{cases} 1 & S \text{ is feasible} \\ 2 & S \text{ is infeasible} \end{cases}$$

ISI essentially says that if a set of customers' items cannot be packed in a single vehicle, there must be at least two vehicles serving the set. If one vehicle can load the items, the inequalities becomes the well-known sub-tour elimination constraints. Figure (1) provides an example where an ISI is identified for an integer solution. In the figure, there is a route $a \to b \to c$ (or $c \to b \to a$), determined by the integer solution. As indicated by the flow, one vehicle serves the route. However, there is no feasible solution for one vehicle (in yellow) to pack the items (in grey). Adding a ISI thus eliminates the solution without losing any feasible integer solution.

Note that ISI can also be easily generalized to the two-dimensional version of WCI where $k(S)$ is calculated by solving a two-dimensional bin packing problem (BPP). This generalized version has been mentioned in the seminal paper of Iori et al. (2007). Due to high computational complexity, the generalization is not applied in our study.

4.3 Strengthened infeasible-path inequalities

For an infeasible path (S, σ) that traverses vertices $v_{\sigma_1}, ..., v_{\sigma_p}$ in order, if another infeasible path $(S', \sigma'), S' = S \setminus \{v_{\sigma_p}\}$, is derived by removing v_{σ_p} from the rear and keeping the sequence of the remaining customers, the following strengthened infeasible-path inequality which dominates $\sum_{e \in E(S,\sigma)} x_e \leq |S| - 1$ can be added.

$$\sum_{e \in E(S',\sigma')} x_e \leq |S'| - 1 \tag{9}$$

4.4 Classic CVRP valid inequalities

Comb inequalities, framed capacity inequalities, (partial) multistar inequalities, generalized multi-star inequalities and hypotour inequalities (Lysgaard et al. 2004) are also valid for model

P. After some preliminary experiments, we chose to add comb inequalities and (partial) multistar inequalities for the reason that the efforts spent on separating the other inequalities outweigh the derived tightness.

5 Separation heuristics

5.1 Separating ISI for integer solutions

Separating ISI for integer solutions is equivalent to calculating $r'(S)$ because a route is easily identified. In the literature, there are some good bounding techniques for the SPP that can be applied to calculate $r'(S)$. The SPP can be roughly described as: given a set of rectangular items and a strip with fixed width, pack all the items with fixed orientation with minimal total height (Martello et al. 2003). That is to say, given a set of customers S and a strip with width W, if the minimal height needed to pack all of their items is h, we can safely have $r'(S) = \lceil \frac{h}{H} \rceil$. We adopt most of the bounds of Côté et al. (2020) as follows except the trivial continuous bound because the capacity inequality (4) renders it redundant:

$$\mathcal{LB}_1(S) = \lceil \frac{\max\{L^{BM}_{dff}(S), L_8(S)\}}{H} \rceil,$$

where $L^{BM}_{dff}(S)$, proposed by Boschetti and Montaletti (2010), is a lower bound based on the *dual feasible function* (Johnson 1973) and $L_8(S)$ is a constructive heuristic proposed by Alvarez-Valdés et al. (2009), transforming an instance in such a way that if the transformed instance cannot fit into a bin with a given lower bound of the height, then the original instance cannot fit, neither, so the lower bound is increased by 1.

$$\mathcal{LB}_2(S) = \lceil \frac{L_4(S)}{H} \rceil,$$

where $L_4(S)$ is calculated by solving (by column generation) the linear relaxation of the non-contiguous bin packing problem.

$$\mathcal{LB}_3(S) = \lceil \frac{L_5(S)}{H} \rceil,$$

where $L_5(S)$ is obtained by solving the corresponding parallel processor scheduling problem with contiguity constraints (relaxation by cutting an item into slices in unit width and the original height (Côté et al. 2014a).

Besides the above lower bounds, an exact algorithm for the 2OPP is also used in our study. The state-of-the-art exact algorithm for the 2OPP was proposed by Côté et al. (2014a); it is composed of a fast branch-and-bound algorithm and a branch-and-cut algorithm. The fast branch-and-bound algorithm is chosen as the exact algorithm for our study because it is more efficient to solve small- and mid-size instances as the ones that we encountered in $2|RO|L$. We denote by $\mathcal{B}\&\mathcal{B}$ the branch-and-bound algorithm. The resulting separation routine is described by **Algorithm 1**.

Algorithm 1 Routine of separating ISI for integer solutions

```
Given an integer solution x'
Identify the set of routes P
for p ∈ P do
    Derive the set of items S associated with p
    if max{LB1(S), LB2(S), LB3(S)} == 2 then
        Go To Add an ISI
    Invoke B&B to calculate r'(S)
    if r'(S) == 2 then
        Go To Add an ISI
Add an ISI:
    Add inequality: x(δ(S)) ≥ 4
```

5.2 Separating WCI and SI for fractional nodes

Identifying an inequality cut for a fractional node is much more difficult than for an integer node since there is an exponential number of routes induced by the solution. We develop a separation routine by combining the well-known Karger's contraction algorithm with the aforementioned lower bounds and $\mathcal{B}\&\mathcal{B}$:

Given an instance and a fractional solution x^* satisfying Eqs (2, 3, 4), find an inequality (8) violated by x^* or prove that none exists. A support graph associated with x^* is defined as $G^* = (V, E^*)$ where $E^* = \{e \in E | x_e^* > 0\}$. Separating an ISI corresponds to finding a set $S \subset V$ such that $r'(S) = 2$ and $2 \leq x^*(\delta(S)) < 4$ on G^*.

Let z_{ij} be a binary variable such that $z_{ij} = 1$ if edge $(i, j) \in \delta(S)$, $z_{ij} = 0$ otherwise. Let y_i be a binary variable such that $y_i = 1$ if customer $i \in S$, $y_i = 0$ otherwise. The separation problem can be formulated as the following feasibility problem:

$$2 \leq \sum_{(i,j)\in E^*} z_{ij} x_{ij}^* < 4, \tag{10}$$

$$y_i = 1, y_j = 0 \text{ or } y_i = 0, y_j = 1 \Longleftrightarrow z_{ij} = 1, \forall i \in V, j \in V, i \neq j, (i, j) \in V \tag{11}$$

$$\sum_{i\in V} y_i \geq 2 \tag{12}$$

$$\sum_{i\in V} y_i \leq |V| - 2 \tag{13}$$

$$r'(S) = 2, \text{where } S = \{i \in V | y_i = 1\} \tag{14}$$

$$z_{ij}, y_i \in \{0, 1\}, \forall i \in V, j \in V \tag{15}$$

Constraint (10) restricts the total amount of flow entering the selected customers to be in the range $[2, 4)$. Constraint set (11) imposes the relation between the binary variables. Constraints (12) and (13) restrict the cardinality of the set. Constraint set (14) imposes that the items associated with the set cannot be packed in a single vehicle. A solution that satisfies the system (10) - (14) identifies an ISI.

We solve the system (10) - (14) as a variant of the global minimum cut problem (Karger 1993). Given the undirected graph G^*, we define a cut of the graph to be a partition of V^* into two non-empty sets A and B. The capacity of a cut (A, B) is the sum of weights of edges with one end in A and the other in B. The cut with the minimum capacity is called "global minimum cut". It is easily seen that the minimum capacity for any support graph in our case equals 2 since there exists no sub-tour after separating Eq. (4). We are interested in the so-called α-minimum cut, i.e., a cut with a capacity within a multiplicative factor of α of

the minimum capacity (Karger and Stein 1996). In our case, the minimum capacity is 2.0 and $\alpha = 2.0$, implied by (10).

We employ Karger's contraction algorithm (Karger 1993) to obtain α-minimum cuts. Karger's contraction algorithm is very simple and efficient for the global minimum cut problem. The "contraction" here indicates an operation that collapses an edge into a super-node (see Figure 2) while the degree of the super-node equals the total weights of edges incident to either of the two contracted nodes. The non-recursive Karger's contraction algorithm was first proposed in Karger (1993); it just performs the contraction operation on a graph until only two super-nodes plus one edge are left.

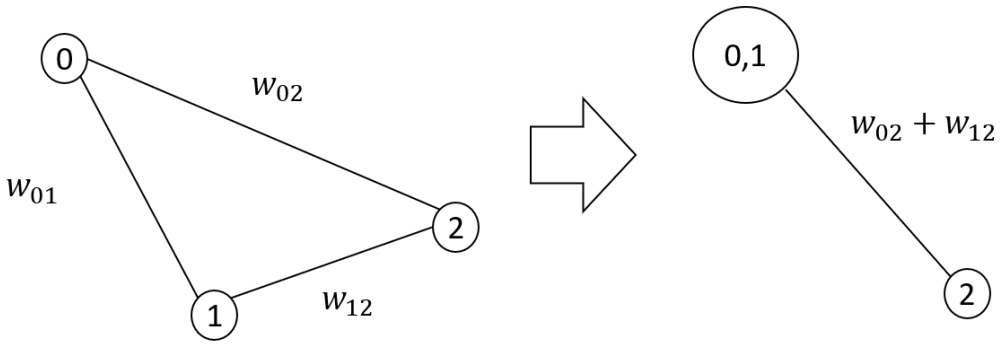

Figure 2: An example of contracting an edge

Because the algorithm randomly selects the edge to contract, its result is not deterministic. Nevertheless, the odd of deriving the global minimum cut can be increased to nearly 1 by running the algorithm in $O(n^2 log(n))$ times (n is the number of vertices). This approach was further improved in Karger and Stein (1996) by recursion. The recursive contraction algorithm can be easily generalized to the α-minimum cut problem that satisfies **Theorem 1**. Accordingly, we can produce all the cuts satisfying Eq.(10) in polynomial time.

Theorem 1. *All cuts that satisfy Eq.(10) can be generated in $O(n^4 log^2(n))$ (see Karger and Stein (1996), Theorem 8.5).*

With all the cuts meeting Eq.(10), the remaining job is to check whether a cut satisfies (14) by invoking the same routine to calculate $r'(S)$ as for integer nodes. Based on the preliminary experiments, however, in practice we choose the non-recursive contraction algorithm because of its computational lightness. Considering the fact that calculating $r'(S)$ is computationally expensive, once a list of candidate sets is derived, the sets are sorted by non-decreasing value of the corresponding capacity. We sweep over the list, and as soon as a violated ISI is found, the separation routine is terminated.

For those sets violating Eq.(10), it is impossible to find a violated ISI. To make better use of them, inspired by Krushinsky and Van Woensel (2015), we try to identify a weak capacity inequality where the right-hand-side (denoted as $k'(S)$) is calculated by solving a one-dimensional BPP associated with weights of items. The BPP is strongly-NP hard, albeit it can be solved quite efficiently in practice. We apply the algorithm from Martello (1990). The entire separation routine for fractional nodes is presented in **Algorithm 2**, where Karger's contraction algorithm is invoked L times independently.

Note that the contraction algorithm naturally fits parallel computing. Therefore, we divide L by five and invoke five independent threads to perform line 3 in **Algorithm 2**. This can decrease the CPU time needed by the algorithm with no impact on its performance.

Algorithm 2 Separation routine for fractional nodes

Initialize L;
Given a fractional solution x^* and the corresponding support graph G^*;
Run the contraction algorithm on G^* for L times and derive a list of sets $\mathcal{S}$;
Sort $\mathcal{S}$ by non-decreasing value of capacity;
for $S \in \mathcal{S}$ **do**
    if $2 \leq x^*(\delta(S)) < 4$ **then**
        Calculate $r'(S)$ as in **Algorithm 1**;
        if $r'(S) == 2$ **then**
            Go to **Add an ISI**;
    else
        Calculate $k'(S)$
        if $x^*(\delta(S)) < 2k'(S)$ **then**
            Add inequality: $x(\delta(S)) \geq 2k'(S)$;
Add an ISI:
    Add inequality: $x(\delta(S)) \geq 4$

5.3 Separating infeasible-path inequalities

At an integer node, a set of routes are identified. On each route, we apply an exact algorithm from Côté et al. (2014b) to check its feasibility regarding the two-dimensional loading constraints. We referred to the packing algorithm as *feasibility checker (FC)*. Once an infeasible-path inequality is found, **Algorithm 3** is invoked to lift it. One may be concerned that **Algorithm 3** slows down the entire process because the FC has to be called more intensively. However, due to the efficiency of the checker, we have not found any negative effect resulting from the lifting procedure.

Algorithm 3 An algorithm to lift an infeasible-path inequality

Given an infeasible route (S, σ);
while True **do**
    Remove the last customer from (S, σ) and check the feasibility of (S, σ);
    if (S, σ) is feasible **then**
        Add the removed customer back to (S, σ);
        Break the loop and return (S, σ)
    else
        Continue the loop

5.4 Separating the other valid inequalities

Because there is an exponential number of constraints (4) and constraints (5), we generate them iteratively to ensure the feasibility as we solve an instance. For an integer solution, rounded capacity inequalities and infeasible-path inequalities are trivially identifiable.

For a fractional solution, the separation is not performed for infeasible-path inequalities. We apply the algorithms from Lysgaard et al. (2004) to separate rounded capacity inequalities, comb inequalities, and (partial) multistar inequalities.

6 The branch-and-cut algorithm

In this section, a branch-and-cut algorithm is presented based on the above valid inequalities and the corresponding separation routines.

6.1 Separation strategy

An efficient branch-and-cut algorithm relies not only on good separation algorithms but also on when to invoke them. We distinguish the root node and the other nodes.

At the root node, when the current solution is fractional, we first try to separate rounded capacity inequalities. If this succeeds, we go to the next iteration. If not, the separation heuristic for comb inequalities is called. When no comb inequality can be found, we run **Algorithm 2** with $L = 5000$ to separate ISI. If no ISI is found, the (partial) multi-star inequalities separation is finally called. When the current solution is integer, a sweep over the corresponding routes is performed to identify violated rounded capacity inequalities, ISI, and infeasible-path inequalities. There is a sequence of calls: rounded capacity inequalities are of the highest priority; if the separation fails, **Algorithm 1** is called for ISI; if the separation fails, perform a sweep over the routes to check the feasibility regarding the two-dimensional loading constraints and try to lift an infeasible-path inequality, if any, by **Algorithm 3**.

For the other nodes, the strategy is the same as for the root node except for the following: when calling **Algorithm 2**, we set L in a hierarchical fashion with respect to the number of nodes explored. Imagine that the enumeration tree has visited a great number of nodes and, as a result, there are even more remaining nodes to explore. Then parameter L can have a huge impact on the accumulated CPU time consumed by **Algorithm 2**. Therefore, we cannot have a uniform L throughout the search. Based on the the preliminary results, we set $L = 100$ for the first $10,000$ nodes (if any). Afterwards, L is reduced by 20 every $10,000$ nodes until it becomes 0 (this means after visiting $50,000$, the separation routine is disactivated).

6.2 Branching strategy

It is acknowledged that selecting the variable to branch on at each node in the enumeration tree affects the speed of the branch-and-cut algorithm and tightness of the lower bounds. This gives rise to a very hard problem – the variable selection problem, which tests each fractional variable and determines which one to select based on some score. A typical scoring mechanism is fully solving the resulting two sub-problems when a tentative variable is branched upon. Others are less computationally intense (see details in Achterberg et al. (2005))). To balance computational efforts and performance of selecting branching variables, we choose the *Pseudo-Shadow Prices* strategy where the score of a variable depends on the dual variables associated with each constraint at the current node (Land and Powell 1979). Note that the branching strategy is crucial here in the sense that it determines the number of nodes to some extent and consequently affects the CPU time of **Algorithm 2**.

6.3 Initial upper bound

A good upper bound allows to accelerate the search. We use the upper bounds from Côté et al. (2020), which are obtained using an adaptive large neighborhood search algorithm.

7 Computational experiments and analyses

This section presents the computational experiments of the branch-and-cut algorithm on the 2L-CVRP benchmark instances from Iori et al. (2007).

Table 1: Parameters associated with the benchmark instances

		Vertical		Homogeneous		Horizontal	
Instance family	$\|M_i\|$	h_{im}	w_{im}	h_{im}	w_{im}	h_{im}	w_{im}
1	1	1	1	1	1	1	1
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]

There are five families of instances (Table 1). The main characteristics distinguishing the five families is the average size of items. Items are unit rectangles in the first family. The other families are sorted by the decreasing order of the size: the second family has the largest items while the fifth family has the smallest items. The benchmark instances were generated based on some CVRP instances from the literature (Toth and Vigo 2002), ranging from 15 to 255 customers. Throughout the rest of the paper, we use ***0x* to denote instance family x. Several groups of comparisons and analyses are conducted.

The entire branch-and-cut algorithm was implemented in C++ under the framework of CPLEX 12.9 and the open source package CVRPSEP. We used the default CPLEX parameter settings except for $CPX\_PARAM\_VARSEL = 4$ specifying the branching strategy. The program was compiled by MSVC 14.0 on Windows 10 operating system. All experiments were carried out on a personal laptop with 8G Ram and Intel(R) Core(TM) i7-6700HQ CPU (2.60GHz).

7.1 The newly-solved instances

We set 4 hours as the time limit and 7 gigabyte as the memory limit. In addition to proving optimality for all the previously-solved instances, our B&C algorithm manages to prove the optimality for 6 new instances as shown in Table 2. The first three columns indicates the name of the instance, number of customers and number of items, respectively. *UB* gives the best upper bound for an instance. *LB* gives the best lower bound. *Time* represents the CPU time in seconds. *Tree size* gives the number of nodes explored by the algorithm.

Table 2: Results for newly-closed instances

Instance	No C	No I	UB	LB	Time	Tree size
1204	30	82	606	606	3031	21,768
1701	40	40	842	842	13876	65,374
1703	40	73	842	842	12409	49,853
1704	40	96	842	842	12542	51,977
1705	40	127	842	842	11592	50,564
1804	44	112	1113	1113	9541	27,700

7.2 Comparison of two algorithms for 2L-CVRP

We compare the proposed B&C algorithm with the state-of-the-art algorithm of Côté et al. (2020). To make a fair comparison, we reproduced the algorithm with CPLEX 12.9 and reran the experiments under the same computational settings.

Table 3 reports the comparison between the two algorithms on the newly-solved instances. In the table, the meaning of the columns is consistent with Table 2. Column *Gap* indicates the percentage gap between *UB* and *LB*. It can be observed that Côté's B&C algorithm fails to prove the optimality for instances *1701, 1703, 1704, 1705* within the time limit. For the other instances, our B&C algorithm costs less CPU time and memory.

Table 4: Comparison between two algorithms on hard instances

Instance	No C	No I	Côté's B&C					Our B&C					Dif-Gap
			UB	LB	Gap	Time	Tree size	UB	LB	Gap	Time	Tree size	
1102	29	43	708.00	673.16	5.17%	8027	623,176	708.00	691.25	2.42%	8034	285,566	2.75%
1402	32	47	1241.00	1204.57	3.03%	7740	125,832	1241.00	1220.33	1.72%	8686	88,584	1.30%
1502	32	48	1101.00	1076.30	2.30%	9066	126,823	1101.00	1080.36	1.91%	8437	79,076	0.38%
1503	32	59	1174.00	1107.89	5.97%	7047	88,586	1174.00	1125.56	4.30%	7964	51,952	1.66%
1702	40	60	851.00	831.51	2.34%	14400	61,015	851.00	837.68	1.59%	14401	36,003	0.75%
1802	44	66	1044.00	991.43	5.30%	8269	102,746	1044.00	1005.00	3.88%	8229	79,684	1.42%
1803	44	87	1101.00	1054.10	4.45%	8987	113,372	1101.00	1060.86	3.78%	8260	87,461	0.67%
1902	50	82	776.00	722.69	7.38%	13935	62,903	776.00	732.60	5.93%	14400	57,430	1.45%
1903	50	103	782.00	739.84	5.70%	14400	50,733	782.00	748.60	4.46%	14400	43,876	1.24%
1904	50	134	797.00	778.42	2.39%	14400	95,330	794.00	781.00	1.66%	14259	62,984	0.72%
2002	71	104	559.00	475.40	17.59%	14401	25,000	559.00	477.12	17.16%	14400	12,944	0.42%
2003	71	151	539.00	496.52	8.56%	14400	31,475	539.00	495.93	8.69%	14400	21,845	-0.13%
2004	71	178	555.00	527.28	5.26%	14400	44,791	555.00	527.50	5.22%	14400	28,392	0.04%
2102	75	114	1079.00	900.77	19.79%	14400	13,810	1079.00	915.56	17.85%	14400	7,849	1.93%
2103	75	164	1168.00	1027.11	13.72%	14400	14,238	1168.00	1043.54	11.93%	14400	8,348	1.79%
2104	75	168	1038.00	982.92	5.60%	14400	24,050	1038.00	987.06	5.16%	14400	17,424	0.44%

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

Instance	No C	No I	Côté's B&C					Our B&C				
			UB	LB	Gap	Time	Tree size	UB	LB	Gap	Time	Tree size
1204	30	82	606.00	606.00	0	2142	33,668	606.00	606.00	0	1970	21,796
1701	75	75	842.00	835.16	0.82%	14400	79,980	842.00	842.00	0	11809	53,409
1703	40	73	842.00	835.87	0.73%	14400	87,359	842.00	842.00	0	9128	50,315
1704	40	96	842.00	836.519	0.65%	14400	83,111	842.00	842.00	0	12542	51,977
1705	40	127	842.00	835.883	0.73%	14400	87,542	842.00	842.00	0	9898	49,646
1804	44	112	1113.00	1113.00	0	7357	114,343	1113.00	1113.00	0	4060	28,762

Table 4 illustrates the results on hard and open instances less than 75 customers because the other instances are too large to tackle. The difficulty stems from two aspects: 1) the gaps between the upper bounds and the lower bounds are still large despite of adding ISI; 2) the resulting tree size is so large that for some instances the algorithm terminates before the time limit due to being out of memory. Instance *2105* is excluded since the FC cannot solve the majority of the packing problems. Column *Dif-Gap* reports the difference between the gaps of the two algorithms. Among 16 instances, there is only one of them (instance *2003*) on which our algorithm performs worse. For the rest, our algorithm provides a lower bound on average 1.0% better and saves more than half of the tree size needed by the other algorithm. The improvement relies on instance families ***02* – 1.23%, ***03* – 0.98%, ***04* – 0.27%. Such observation is in accordance with the fact that in general ***02* tends to have more infeasible routes in terms of the two-dimensional loading constraints because the continuous lower bound has a relatively poor approximation. As a result, there are more ISI separated for the family, leading to better improvement on the lower bound. In light of the newly-solved instances, although family instance ***02* benefits the most from ISI, it is still very hard to close the gap for instances from this family. By contrast, the improvement for the other families are less on average but turns out to be sufficient enough to reach optimality. Table 4 also illustrates that for instances with more than 70 customers, the gap can be as large as more than 15%, which implies the limit of both B&C algorithms.

7.3 Impact of the set inequalities

We aim to analyze the effectiveness of separating ISI for both fractional solutions and integer solutions. Two algorithmic variants are created to this end. The first setting *All Cuts* represents the full version where the separation is triggered for both integer and fractional solutions. The second setting *ISI Cuts Int* corresponds to the version that merely separates infeasible set in-

equalities for integer solutions. The last setting *No ISI Cuts* corresponds to the version without ISI. The sixth column *Dif-LB* gives the relative percentage between the lower bounds of *All Cuts* and *ISI Cuts Int* (the latter is the baseline). The next column *Dif-M* gives the relative difference between the tree sizes. The last two columns give the relative difference between *ISI Cuts Int* and *No ISI Cuts* (the latter is the baseline). To avoid memory-out, we set a time limit of 2000 seconds. One can see that ISI substantially tighten the lower bound. It is more evident for ***02* and ***03*. The inequalities are also able to reduce the size of the tree. Separating the inequalities for fractional solutions provides a better lower bound for 14 instances out of 16 with a smaller tree, on average 0.30% higher lower bounds and 17.58% fewer nodes. The last two columns indicate the impact of the inequalities separated for fractional solutions. On average, ISI for integer solutions can strengthen the lower bound by 1.14% and decrease the tree size by 26.69%.

Table 5: Impact of the set inequalities

Instances	All Cuts		ISI Cuts Int		Dif-LB	Dif-M	No ISI Cuts		Dif-LB	Dif-M
	LB	Tree size	LB	Tree size			LB	Tree size		
1102	680.50	81,804	677.67	10,8470	0.42%	-24.58%	656.90	18,0560	3.16%	-39.93%
1402	1210.00	28,297	1207.07	32,328	0.24%	-12.47%	1196.84	60,246	0.85%	-46.34%
1502	1074.54	28,148	1072.67	33,746	0.17%	-16.59%	1069.72	49,000	0.28%	-31.13%
1503	1122.00	16,382	1110.85	17,920	1.00%	-8.58%	1105.30	38,703	0.50%	-53.70%
1702	829.32	8,508	828.42	9,153	0.11%	-7.05%	823.91	14,128	0.55%	-35.21%
1802	999.57	22,199	995.00	25,310	0.46%	-12.29%	978.81	39,403	1.65%	-35.77
1803	1055.41	26,459	1055.14	23,484	0.03%	12.67%	993.00	32,712	6.26%	-28.21%
1902	725.18	11,622	720.17	12,809	0.70%	-9.27%	712.88	18,955	1.02%	-32.42%
1903	739.74	8,400	735.65	10,858	0.56%	-22.64%	729.13	14,281	0.89%	-23.97%
1904	772.64	10,690	771.69	12,648	0.12%	-15.48%	765.56	18,727	0.79%	-32.46%
2002	475.48	2,921	473.27	5,209	0.47%	-43.92%	470.80	7,500	0.52%	-30.55%
2003	494.82	4,621	494.83	6,745	0.00%	-31.49%	494.78	6,979	0.01%	-3.35%
2004	526.43	6,673	526.47	8,500	-0.01%	-21.49%	526.30	11,059	0.03%	-23.14%
2102	909.83	1,644	906.53	1,920	0.36%	-14.38%	897.42	3,802	1.02%	-49.50%
2103	1038.61	1,645	1032.26	2,418	0.62%	-31.97%	1020.58	3,779	1.14%	-36.01
2104	983.64	5,034	980.44	6,155	0.33%	-18.21%	976.80	6,003	0.37%	2.53%

7.4 Impact of branching strategies

To analyze the impact of the branching strategy on our algorithm, we create a variant of the B&C by setting the branching strategy as the default of CPLEX 12.9 MIP solver. We set 2000 seconds as the time limit to avoid memory out. The comparison is tested over the same open instances as shown by **Table 6**. Column *Default* gives results of the default branching strategy. Column *PSP* (Pseudo-Shadow Prices) gives the results of the chosen branching strategy. *Dif-LB* presents the relative difference. *Dif-M* presents the relative difference of the tree sizes. In terms of tightening the lower bound, the effort for performing the chosen branching strategy pays off. The lower bound is 0.55% higher on average. Unlike separating ISI, the average of the M-Gap is nearly zero.

Table 6: Impact of the branching strategy

Instances	Default		PSP		Dif-LB	Dif-M
	LB	Tree size	LB	Tree size		
1102	658.33	79,732	680.50	81,804	3.37%	2.60%
1402	1201.71	31,404	1210.00	28,297	0.69%	-9.89%
1502	1068.81	31,306	1074.54	28,148	0.54%	-10.09%
1503	1114.50	22,424	1122.00	16,382	0.67%	-26.94%
1702	826.73	12,782	829.31	8,508	0.31%	-33.44%
1802	992.66	28,090	999.57	22,199	0.70%	-20.97%
1803	1046.33	20,676	1055.41	26,459	0.87%	27.97%
1902	721.96	11,283	725.18	11,622	0.45%	3.00%
1903	738.56	9,004	739.74	8,400	0.16%	-6.71%
1904	768.20	10,769	772.64	10,690	0.58%	-0.73%
2002	477.65	3,763	475.48	2,921	-0.46%	-22.38%
2003	494.60	4,349	494.82	4,621	0.04%	6.25%
2004	525.56	5,995	526.43	6,673	0.17%	11.31%
2102	907.45	1,824	909.83	1,644	0.26%	-9.87%
2103	1036.60	1,792	1038.61	1,645	0.19%	-8.20%
2104	977.11	3,598	983.64	5,034	0.67%	39.91%
2105	964.41	4,688	965.20	6,616	0.08%	41.13%

7.5 Observation on the packing algorithms

The importance of the FC is self-evident. Over the past decade, there have been many studies on developing efficient heuristics and exact algorithms for similar packing problems. A question naturally rises: is the state-of-the-art exact packing algorithm good enough for the 2L-CVRP?

Based on the experiments, we think the FC is no longer a bottleneck. Over the course of the B&C algorithm, the FC never fails for instance families ***02*, ***03* and ***04*. The average solution time on an instance from these families is very low as shown by Figure 3. A large computational effort is only needed for family ***05*. Nevertheless, the computational effort can be drastically reduced by having a good upper bound. Table 7 gives two typical examples. Column *Time on the FC* gives the accumulated time spent on the FC throughout the algorithm. When we set a better initial upper bound for both instances, one can see a sharp drop in total time. The reason for the large amount of time is that the FC occasionally cannot solve a packing problem to optimality within 1200 seconds (the time limit for the FC). But a majority of these unsolved packing problems are associated with sub-optimal solutions. With a better initial upper bound, the packing problems are avoided by the bounding mechanism of the B&C algorithms. Since there are many powerful heuristics for 2L-CVRP in the literature, it can be realistic to have a very good solution (even the optimum) at the very beginning.

Table 7: Solution time on two special instances

Instance	UB	LB	Total time	Time on the FC
1505	1336.00	1336.00	6999	6986
1505[a]	1336.00	1336.00	41	31
1805	937.00	937.00	5294	5219
1805[a]	937.00	937.00	133	84

[a] With a better initial upper bound

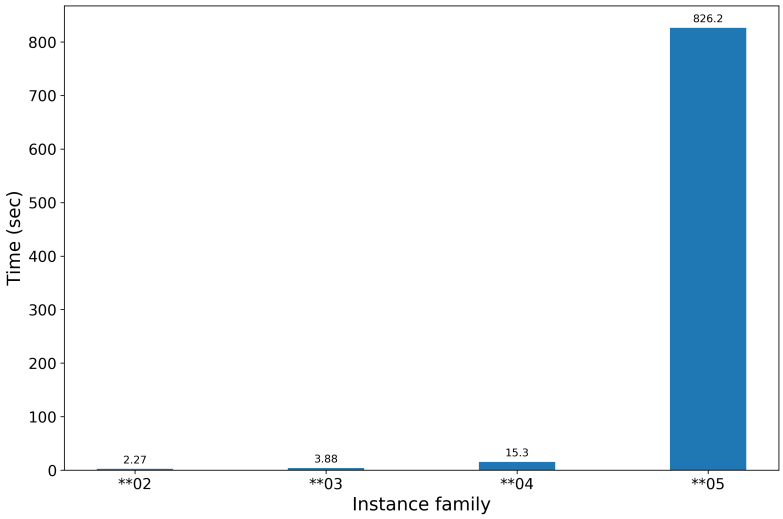

Figure 3: The CPU time cost by the FC

Besides the FC, $\mathcal{BB}$ also plays a critical part in the B&C algorithm in the sense that it determines the quality of the linear relaxation as well as it is invoked many times during a search. The time limit for the $\mathcal{BB}$ is 2 seconds. If it fails to solve a packing problem within the limit, we count it as one failure. The failure rate is a good indicator showing us if the algorithm is efficient enough. Figure 4 presents the average failure rates to the four families. The figure is in a ladder form which indicates that the risk of failure grows as the size of items decreases. But the worst failure rate is relatively small, under 10.0%, so we believe that $\mathcal{BB}$ is not a bottleneck.

Although both the FC and $\mathcal{BB}$ perform the best for ***02*, the majority of the open instances (with fewer than 75 customers) come from ***02*. This observation also supports the argument that $\mathcal{BB}$ and the FC are no longer bottlenecks.

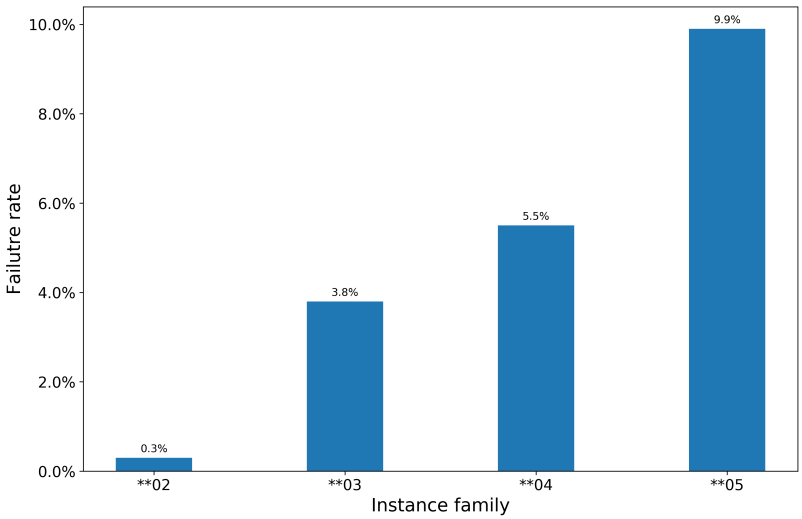

Figure 4: The average failure rates of $\mathcal{BB}$

8 Conclusions and future research

In this paper, we investigated the capacitated vehicle routing problem with two-dimensional loading constraints. We proposed a separation routine to identify infeasible set inequalities and weak capacity inequalities for fractional solutions. A lifting heuristic for infeasible-path inequalities was also developed. Finally, a branch-and-cut algorithm was proposed to solve the problem. We also reproduced the state-of-the-art algorithm and ran the same experiments with it.

Computational results on the benchmark instances illustrate that the newly-introduced inequalities can tighten the lower bound and reduce the tree size for the majority of the hard instances. Six instances are solved to optimality for the first time. Among these instances, there are: four instances exclusively solved by the proposed B&C algorithm; two instances solved by the proposed algorithm and the reproduced algorithm. For hard instances, our algorithm increases the lower bound by 1.0% on average and saves more than half of the tree size. The impact of the chosen branching strategy was also analyzed. On hard instances, the strategy pays off in terms of strengthening the lower bound, but there is no clear reduction on the tree size. Afterwards, the impact of the ISI (infeasible set inequalities) is verified. Fractional ISI strengthens the lower bounds by 0.30% and trims 17.58% nodes on average. Finally, analyses are carried out for the packing algorithms. It seems that the packing algorithms are no longer the biggest bottleneck as they were a decade ago. For one thing, they are not the bottleneck for instances from the ***02*, ***03* and ***04* families with fewer than 75 customers, though better packing algorithms can lessen the CPU time by a moderate amount. Interestingly, in light of the fact that most open instances are from the ***02* and ***03* families, we think one needs to work on something other than the packing algorithms to close these instances. This naturally brings up a future direction which is to develop exact algorithms by applying the column generation approach and cut generation together.

Acknowledgements

This work was supported by the Canadian Natural Sciences and Engineering Research Council (NSERC). Computing facilities were provided by Compute Canada. These supports are gratefully acknowledged.

References

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

Alinaghian, M., Zamanlou, K., Sabbagh, M. S., 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.

Alvarez-Valdés, R., Parreño, F., Tamarit, J. M., 2009. A branch and bound algorithm for the strip packing problem. OR spectrum 31 (2), 431–459.

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

Boschetti, M. A., Montaletti, L., 2010. An exact algorithm for the two-dimensional strip-packing problem. Operations Research 58 (6), 1774–1791.

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

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

Côté, J.-F., Gendreau, M., Potvin, J.-Y., 2020. The vehicle routing problem with stochastic two-dimensional items. Transportation Science 54 (2), 453–469.

Côté, J.-F., Guastaroba, G., Speranza, M. G., 2017. The value of integrating loading and routing. European Journal of Operational Research 257 (1), 89–105.

Dominguez, O., Guimarans, D., Juan, A. A., 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, A. A., 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.

Fuellerer, G., Doerner, K. F., Hartl, R. F., 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, A. A., 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, F. K., Xavier, E. C., 2016. A branch-and-cut approach for the vehicle routing problem with loading constraints. Expert Systems with Applications 47, 1–13.

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

Johnson, D. S., 1973. Near-optimal bin packing algorithms. Ph.D. thesis, Massachusetts Institute of Technology.

Karger, D. R., 1993. Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In: SODA. Vol. 93. pp. 21–30.

Karger, D. R., Stein, C., 1996. A new approach to the minimum cut problem. Journal of the ACM (JACM) 43 (4), 601–640.

Krushinsky, D., Van Woensel, T., 2015. An approach to the asymmetric multi-depot capacitated arc routing problem. European Journal of Operational Research 244 (1), 100–109.

Land, A., Powell, S., 1979. Computer codes for problems of integer programming. In: Annals of Discrete Mathematics. Vol. 5. Elsevier, pp. 221–269.

Leung, S. C., 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, A. N., Eglese, R. W., 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., 1990. Knapsack problems: algorithms and computer implementations. Wiley-Interscience series in discrete mathematics and optimiza tion.

Martello, S., Monaci, M., Vigo, D., 2003. An exact approach to the strip-packing problem. INFORMS journal on Computing 15 (3), 310–319.

Pinto, T., Alves, C., de Carvalho, J. V., 2016. A branch-and-price algorithm for the vehicle routing problem with 2-dimensional loading constraints. In: International Conference on Computational Logistics. Springer, pp. 321–336.

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

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.

Toth, P., Vigo, D., 2002. An overview of vehicle routing problems. The vehicle routing problem, 1–26.

Wei, L., Zhang, Z., Zhang, D., Leung, S. C., 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.

Wei, L., Zhang, Z., Zhang, D., Lim, A., 2015. A variable neighborhood search for the capacitated vehicle routing problem with two-dimensional loading constraints. European Journal of Operational Research 243 (3), 798–814.

Zachariadis, E. E., Tarantilis, C. D., Kiranoudis, C. T., 2009. A guided tabu search for the vehicle routing problem with two-dimensional loading constraints. European Journal of Operational Research 195 (3), 729–743.

Zachariadis, E. E., Tarantilis, C. D., Kiranoudis, C. T., 2017. Vehicle routing strategies for pick-up and delivery service under two dimensional loading constraints. Operational Research 17 (1), 115–143.

Zhang, Z., Wei, L., Lim, A., 2015. An evolutionary local search for the capacitated vehicle routing problem minimizing fuel consumption under three-dimensional loading constraints. Transportation Research Part B: Methodological 82, 20–35.