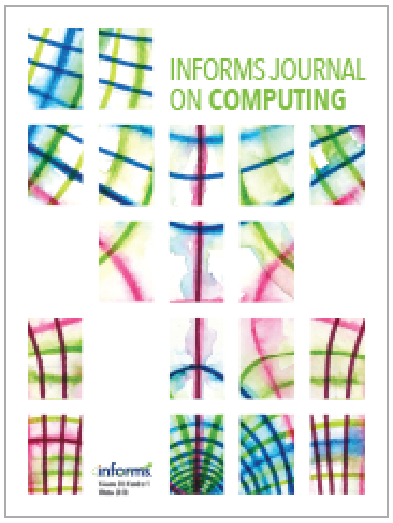


INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Learning-Based Branch-and-Price Algorithms for the Vehicle Routing Problem with Time Windows and Two-Dimensional Loading Constraints

Xiangyi Zhang, Lu Chen, Michel Gendreau, André Langevin

To cite this article:
Xiangyi Zhang, Lu Chen, Michel Gendreau, André Langevin (2022) Learning-Based Branch-and-Price Algorithms for the Vehicle Routing Problem with Time Windows and Two-Dimensional Loading Constraints. INFORMS Journal on Computing 34(3):1419-1436. https://doi.org/10.1287/ijoc.2021.1110

http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 34, No. 3, May–June 2022, pp. 1419–1436
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

Learning-Based Branch-and-Price Algorithms for the Vehicle Routing Problem with Time Windows and Two-Dimensional Loading Constraints

Xiangyi Zhang,[a,b] **Lu Chen,**[c] **Michel Gendreau,**[a,b] **André Langevin**[a,b]

[a] Département de mathématiques et de génie industriel, Polytechnique Montréal, Montreal, Quebec H3C 3A7, Canada; [b] Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, Montreal, Quebec H3C 3J7, Canada; [c] School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

Contact: xiangyi.zhang@polymtl.ca, https://orcid.org/0000-0003-4395-2776 (XZ); chenlu@sjtu.edu.cn (LC); michel.gendreau@polymtl.ca, https://orcid.org/0000-0002-9262-3648 (MG); andre.langevin@polymtl.ca (AL)

Received: August 12, 2020
Revised: March 12, 2021; May 28, 2021
Accepted: June 2, 2021
Published Online in Articles in Advance: December 29, 2021

https://doi.org/10.1287/ijoc.2021.1110

Abstract. A capacitated vehicle routing problem with two-dimensional loading constraints is addressed. Associated with each customer are a set of rectangular items, the total weight of the items, and a time window. Designing exact algorithms for the problem is very challenging because the problem is a combination of two NP-hard problems. An exact branch-and-price algorithm and an approximate counterpart are proposed to solve the problem. We introduce an exact dominance rule and an approximate dominance rule. To cope with the difficulty brought by the loading constraints, a new column generation mechanism boosted by a supervised learning model is proposed. Extensive experiments demonstrate the superiority of integrating the learning model in terms of CPU time and calls of the feasibility checker. Moreover, the branch-and-price algorithms are able to significantly improve the solutions of the existing instances from literature and solve instances with up to 50 customers and 103 items.

Summary of Contribution: We wish to submit an original research article entitled "Learning-based branch-and-price algorithms for a vehicle routing problem with time windows and two-dimensional loading constraints" for consideration by IJOC. We confirm that this work is original and has not been published elsewhere, nor is it currently under for publication elsewhere. In this paper, we report a study in which we develop two branch-and-price algorithms with a machine learning model injected to solve a vehicle routing problem integrated the two-dimensional packing. Due to the complexity brought by the integration, studies on exact algorithms in this field are very limited. Our study is important to the field, because we develop an effective method to significantly mitigate computational burden brought by the packing problem so that exactness turns to be achievable within reasonable time budget. The approach can be generalized to the three-dimensional case by simply replacing the packing algorithm. It can also be adapted for other VRPs when high-dimensional loading constraints are concerned. Broadly speaking, the study is a typical example of adopting supervised learning to achieve acceleration for operations research algorithms, which expands the envelop of computing and operations research. Hence, we believe this manuscript is appropriate for publication by IJOC.

History: Accepted by Andrea Lodi, Area Editor for Design & Analysis of Algorithms—Discrete.
Funding: This work was supported by the Canadian Natural Sciences and Engineering Research Council (NSERC) through its Discovery Grants program. Computing facilities were provided by Compute Canada. These supports are gratefully acknowledged.
Supplemental Material: The e-companion and data are available at https://doi.org/10.1287/ijoc.2021.1110.

Keywords: vehicle routing problem • column generation • supervised learning • loading constraints

1. Introduction

In recent years, there has been a growing interest in the integration of other decisions in vehicle routing problems. The trend is driven by three factors: increasing capability of modern computing resources, economic benefits, and practical needs. As pointed out by Côté et al. (2017), combining routing and loading decisions can save up to 50% of the cost incurred if these decisions are made separately. The related problems often occur in practice, for instance, when transporting items such as porcelain, furniture, and kitchen appliances with stacking being not allowed due to the fragility of items (Gendreau et al. 2008). On the other hand, the integration of routing and multi-dimensional loading is computationally challenging because evaluating the feasibility of the loading constraints is generally a strongly-NP hard problem. The potential benefits and the algorithmic challenge motivate this study.

In this paper, we devote to developing column generation-based algorithms to solve a problem described as follows: given a set of customer requests, each of which is characterized by a total weight, a set of rectangular items and a hard time window, a set of routes for a homogeneous fleet that satisfy all the requests with minimal travel distance, while respecting all operational constraints (e.g., loading constraints, time windows) is sought-after. It is referred to as the vehicle routing problem with time windows and two-dimensional loading constraints (2L-VRPTW).

The feature of two-dimensional loading constraints in the context of VRPs is typically defined by item orientation and loading restriction (Fuellerer et al. 2009). *Item Orientation* specifies whether an item can be rotated or not. For some cases, items can be placed by either longer side or shorter side, whereas fixed orientation generally happens when items are only accessible to a specific side (Bortfeldt and Homberger 2013). *Loading restriction* specifies whether the order of packing items should account for the visiting order of customers. The restriction becomes critical when a vehicle's container can only be opened from the rear. In our problem, rotation of items is forbidden and loading restrictions are not considered.

The majority of exact algorithms for the VRPTW or its variants are built on the branch-and-price methodology (B&P) (Costa et al. 2019). A typical B&P algorithm uses a branch-and-bound (B&B) tree where the linear relaxation of the subproblem considered at each node is solved by column generation, calling for solving a pricing problem to find columns with negative reduced cost. The pricing problem is usually tackled as an elementary shortest path problem with resource constraints (ESPPRC), a strongly NP-hard problem. It is typically solved by a labeling algorithm (Desaulniers et al. 2006). To mitigate the complexity of the pricing problem, some relaxation techniques (Desrochers et al. 1992, Righini and Salani 2008, Baldacci et al. 2011) were proposed. However, when it comes to the 2L-VRPTW, the pricing problem is structurally altered in the sense that any priced-out column has to be checked by solving a strip packing problem (SPP) (Martello et al. 2003) in order to ensure the loading feasibility. From the complexity point of view, whichever existing relaxation technique is used, the pricing problem remains strongly NP-hard. Such feasibility check is also very crucial to other algorithms for VRPs considering multidimensional loading constraints. To speed up the checking, a variety of packing heuristics and bounding techniques are used to evaluate routes (Gendreau et al. 2006, Iori et al. 2007, Zhang et al. 2015, Pinto et al. 2016, Mahvash et al. 2017). In our study, we explore a fundamentally different strategy to achieve acceleration. The idea is to decrease the usage frequency of the feasibility checker. That is, before invoking the feasibility checker, columns are classified by a machine learning model. Thereafter, they are checked selectively based on the classification result.

Machine learning also introduces several issues to column generation. One concern is that a machine learning model works in a stochastic manner, which can cause infeasible solutions. Another concern is that the model deployed into an OR algorithm should be computationally light otherwise the benefit brought by the model may be outweighed. In other words, when designing the architecture of a model, one needs to take both compactness and capability into consideration. The last concern is the generalization ability of the model. The model should be effective on instances from different distributions to obtain a robust algorithm. All these issues are discussed and the corresponding solutions are presented in the paper. In summary, there are four contributions in this paper:

1. A new mechanism of column generation for the 2L-VRPTW is proposed. Inside the traditional column generation framework, a supervised learning model and a procedure to rectify the learning model is embedded to guarantee the validity of the entire algorithm. The resulting column generation algorithm can effectively mitigate the computational burden caused by the loading constraints.
2. An exact dominance rule is proposed considering the two-dimensional loading constraints in the pricing problem rather than relaxing the constraints. A good approximate dominance rule is also proposed for the sake of time budget.
3. A new supervised learning task is established. A feedforward neural network with hand-crafted features as the input is built to address the task.
4. An exact B&P algorithm and its approximate variant are developed to solve the instances from the literature. Extensive computational experiments are reported regarding the performance of the B&P algorithms as well as the impact of the learning model.

The remainder of the paper is organized as follows. Section 2 reviews algorithms related to the 2L-VRPTW and applications of machine learning to combinatorial optimization (CO) problems. Section 3 describes the mathematical formulation, the pricing algorithm, and the feasibility checker for the two-dimensional loading constraints. Section 4 describes the new column generation mechanism and the B&P algorithms. Computational experiments are reported in Section 5, followed by the conclusion in the last section.

2. Literature Review

In the following sections, we use 2L-VRPs and 3L-VRPs to denote vehicle routing problems with two- and three-dimensional loading constraints, respectively. One-dimensional loading occasionally occurs in pick-up and delivery (Cordeau et al. 2010), but studies in this stream are not considered in the review. We are aware that there is a large body of studies on algorithms for two-dimensional packing problems, which is closely related to the loading constraints, but they are excluded in this section for the sake of compactness. Readers are referred to a recent survey (Iori et al. 2021) for details.

2.1. Exact Algorithms for Vehicle Routing Problems with Loading Constraints

2L-VRPs and 3L-VRPs are mostly solved by heuristics and meta-heuristics in the literature. By contrast, studies on exact algorithms in this field are very limited. According to Pollaris et al. (2015), only one study (Iori et al. 2007) out of 32 before 2014 proposes an exact algorithm. They designed a branch-and-cut algorithm (B&C) for a capacitated vehicle routing problem with 2D loading constraints (2L-CVRP). They introduced the “infeasible-path” constraints to the classic two-index vehicle flow model for the CVRP (Toth and Vigo 2002). In the course of B&B search, the feasibility checker is only invoked when an integer node is detected. Instances with up to 35 customers and more than 100 items can be solved optimally. Very recently, Côté et al. (2020) developed a branch-and-cut algorithm based on a cutting-edge exact packing algorithm from Côté et al. (2014b) and finally achieved a significant improvement on the benchmark instances given by Iori et al. (2007). The size of optimally solved instances is nearly doubled. To the best of our knowledge, Côté et al. (2020) is the state-of-the-art exact algorithm for 2L-CVRP.

2.2. Column Generation for Vehicle Routing Problems with Loading Constraints

Column generation, whether exactness is ensured or not, is an important algorithmic component of solution methods for 2L-VRP and 3L-VRP according to Mahvash et al. (2017). Their column generation-based heuristics manage to improve solutions from Gendreau et al. (2006) and Tarantilis et al. (2009) by 5.04% and 1.5%, respectively. Literature focusing on column generation for 2L-VRPs or 3L-VRPs is quite limited. There are two main related works. Pinto et al. (2013) addressed the 2L-CVRP with unloading sequential constraints. They proposed a column-generation-based heuristic where a route is priced out in an incremental fashion. Initially, the loading constraints are removed from the pricing problem and the relaxed subproblem is solved exactly. Afterward, a generated route is checked calculating the lower bound of the formulated strip packing problem. If the lower bound is larger than the volume of the vehicle container, then the route is eliminated. If not, a heuristic constructs a feasible packing solution for the route. Finally, if the heuristic fails, the route is rectified by removing some customers based on several rules until a feasible packing is obtained or the route no longer has negative reduced cost. Addressing the same problem, Pinto et al. (2016) developed a B&P algorithm where the pricing problem is solved via a variable neighborhood search (VNS) algorithm (Pinto et al. 2015). The VNS kicks off from a constructive heuristic offering a high-quality feasible solution and later conducts local searches with respect to seven neighborhoods associated with packing or routing. Mahvash et al. (2017) addressed the 3L-CVRP by a B&P algorithm. The pricing problem is solved as follows: relax the loading constraints and price out optimal routes; check the feasibility of the loading constraints for each route by an extreme-point heuristic (Crainic et al. 2008); if a route is denied by the checker, it is rectified by a savings algorithm; finally, those routes either infeasible or with nonnegative reduced cost after the rectification are discarded. They also proposed a greedy algorithm as a computationally lighter heuristic, which is invoked before running the aforementioned procedure.

It can be observed that a common mechanism to deal with the loading constraints is usually as follows: Phase I—relax the loading constraints and run a column generator to derive a set of columns; Phase II—run the feasibility checker on all columns; Phase III—repair or discard infeasible columns. This mechanism has three main drawbacks. The first one is that the dominance rule is not exact; the second is that repairing an infeasible column could be time-consuming in the sense that each time a customer is removed from a column, the feasibility checker has to be invoked; lastly, it is not necessary to check all the generated columns considering the fact that only the variables (columns) selected in the optimal basis should be guaranteed to be feasible. Nonbasic variables (columns) can be infeasible, having no effect on the solution.

2.3. Time Window Constraints in 2L-VRPs

Besides 2L-CVRP, there have also been some efforts on the 2L-VRPTW. Martínez and Amaya (2013) address a real-life problem for transporting perishable food that cannot be stacked. The problem is solved by a tabu search integrated with a packing heuristic algorithm. Song et al. (2019) also address a practical 2L-VRPTW from the food service industry, in which one is required to load pallets and individual rectangular items on vehicles. A generalized variable neighborhood search algorithm is designed for the problem. The only study dealing with the exact same problem as ours is Khebbache-Hadji et al. (2013) who develop two heuristics and a memetic algorithm.

2.4. Machine Learning Models in Combinatorial Optimization

Applying machine learning (ML) to solve CO problems is not a fresh idea. It can date back to the 1980s (Smith 1999). A recent review by Bengio et al. (2021) classifies the role that ML models play in operations research (OR) algorithms: (1) End-to-End; (2) learning valuable information; (3) learning subroutines. "End-to-End" algorithms apply a ML model to directly solve a CO problem by mapping the problems domain to the solutions domain (Khalil et al. 2017, Kool et al. 2018, Nair et al. 2018). "Learning valuable information" consists in using ML models to extract useful data for modeling or solving a CO problem. It operates in a fashion of "predict first, optimize second" where valuable information is predicted by a ML model and then an OR algorithm is used with the predicted information (Kruber et al. 2017, Bonami et al. 2018). The last category uses a ML model to learn subroutines. When solving a CO problem, an algorithm may encounter different decision points. For instance, a B&B algorithm has to select a variable to branch, which is a crucial decision to make. Inspired by ML techniques, Alvarez et al. (2017) and Khalil et al. (2016) propose different ML-based branching rules embedded in a B&B tree to achieve better efficiency. There are also studies on training ML models to select optimality cut for the L-shaped method (Jia and Shen 2019) and Gomory cuts for a cutting plane algorithm (Tang et al. 2019). As for column generation, to the best of our knowledge, there is only one study linking ML to column generation. Triggered by the repetition of pricing problems in column generation, Václavík et al. (2018) apply a fast regression model that predicts a tight upper bound for a pricing problem based on historical data. Because the pricing problems discussed in the paper are solved by a MIP solver, the prediction can be used as a very good upper bound to accelerate the MIP. It saves nearly 40% and 22% CPU time for the resulting B&P algorithm in two different types of scheduling problems.

3. Mathematical Formulation and the Pricing Algorithms

In this section, the three-index formulation and the set partitioning formulation of the problem are presented. The pricing problem is also analyzed. An exact dominance rule and an approximate dominance rule are proposed.

3.1. Notation and Problem Description

The 2L-VRPTW is defined on a complete directed graph $G = (V, A)$, where the triangle inequality holds. $V = \{v_0, v_1, v_1, v_2, \ldots, v_n, v_{n+1}\}$ is the set of nodes on the graph, where v_0 and v_{n+1}, the source and sink nodes, stand for two copies of the depot appearing at the two ends of a route. Nodes $N = \{v_1, v_2, \ldots, v_n\}$ stands for the set of customers and A is the set of arcs, representing direct links between any of two nodes on the graph. A nonnegative value d_a is associated with any arc $a \in A$, which depicts the travel distance between the two end nodes of the arc. In addition, an arc from node v_i to node v_j can be denoted as (i, j).

A homogeneous fleet of vehicles is denoted as K. Each vehicle has a loading area characterized by width and height: W and H, respectively. The weight capacity of a vehicle is Q and the total area equals $\tilde{v} = W \times H$.

For each customer $v_i \in N$, there is a demand consisting of a set of rectangular items M_i. Each item $m \in M_i$ has a specific width w_{im}, height h_{im}, and weight q_{im}. We also respectively use ν_i and q_i to denote the total area and total weight of items for customer v_i, 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. A hard time window $[a_i, b_i]$ is associated with each node $i \in V$, where a_i, b_i, represent, respectively, the earliest and the latest time at which service at node v_i has to begin. Each customer has a service duration denoted as u_i. Figure 1 is provided for illustrative purpose. 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. The total weight of the carried items should not exceed the vehicle capacity.
3. Items transported in a vehicle must not exceed the loading area.
4. Items must be positioned without overlapping.
5. Service at a customer should begin within the time window. An earlier arrival is allowed, but the service is not performed until the earliest time.

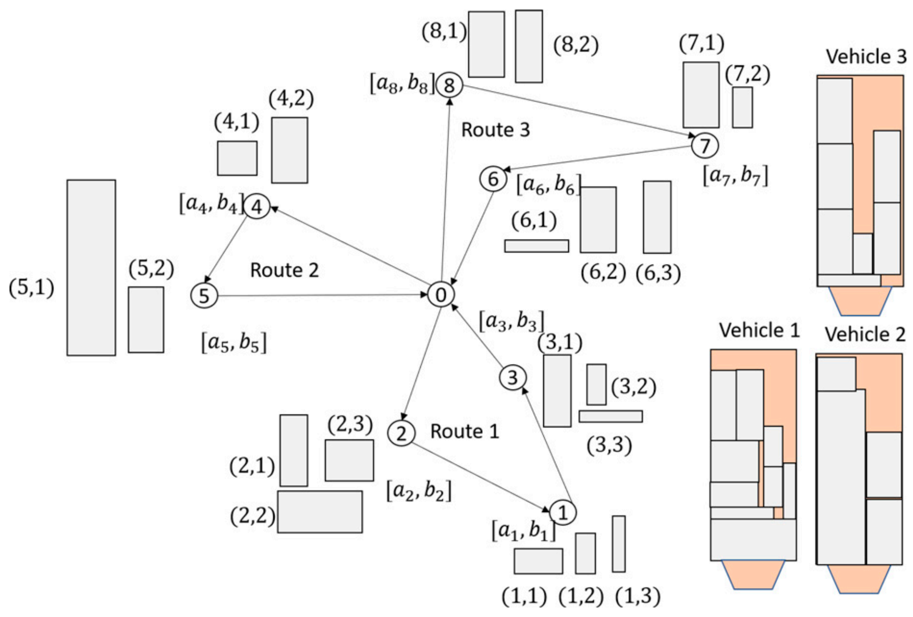
The objective is to minimize the total distances travelled by the fleet. Furthermore, the following assumptions are made throughout the study.

1. All the information is deterministic and known before planning.
2. Heights, widths, and weights are integers.
3. The demand of any customer does not exceed the vehicle capacity.
4. Items are not allowed to be rotated and there is no restriction on the unloading sequence.
5. A vehicle executes at most one route.

3.2. Three-Index Formulation

We first present a three-index formulation for the problem, which uses two families of decision variables. For each arc (i, j) and each vehicle k, binary variable $x^k_{i,j}$ takes value 1 if arc (i, j) is traversed by vehicle k; continuous variable s^k_i indicates the starting time of service at customer v_i by vehicle k. Representing the

Figure 1. (Color online) Example of a 2L-VRPTW



loading constraints requires the following notation: Γ is the set of all the routes that satisfy the loading constraints and the standard VRPTW constraints; $\bar{\Gamma}$ stands for the routes that satisfy the standard VRPTW constraints; $A(\gamma)$ denotes the set of arcs in a route $\gamma \in \Gamma$. The 2L-VRPTW can thus be modeled as follows:

$$[\text{P}] \quad \min \sum_{(i,j)\in A} \sum_{k\in K} x_{i,j}^k d_{i,j} \tag{1}$$

s.t.

$$\sum_{(0,j)\in A} x_{0,j}^k \le 1, \quad \forall k \in K, \tag{2}$$

$$\sum_{k\in K} \sum_{(i,j)\in A} x_{i,j}^k = 1, \quad \forall j \in N \tag{3}$$

$$\sum_{(i,j)\in A} x_{i,j}^k - \sum_{(j,i)\in A} x_{i,j}^k = 0, \quad \forall j \in V, \forall k \in K \tag{4}$$

$$s_i^k + u_i + d_{i,j} - s_j^k + M x_{i,j}^k \le M, \quad \forall i \in N, \ \forall k \in K \tag{5}$$

$$a_i \le s_i^k \le b_i, \quad \forall i \in N \tag{6}$$

$$\sum_{(i,j)\in A} x_{i,j}^k q_j \le Q, \quad \forall k \in K \tag{7}$$

$$\sum_{a\in A(\gamma)} x_a^k \le |A(\gamma)| - 1, \ \forall k \in K, \ \forall \gamma \in \bar{\Gamma} \backslash \Gamma \tag{8}$$

$$x_{i,j}^k \in \{0,1\}, \ \forall (i,j) \in A, \ \forall k \in K \tag{9}$$

Objective Function (1) minimizes the total distance travelled by the fleet. Constraint Set (2) represents that a vehicle has at most one route to execute. Constraint Set (3) enforces that each customer is served exactly once. Constraint Set (4) contains the flow conservation constraints. Constraint Sets (5–6) respect the temporal sequence of visiting customers as well as the hard time window constraints. We assume that the speed of a vehicle equals 1 so that the amount of consumed time is equivalent to the travelled distance. Constraint Set (7) enforces the weight limit. Constraint Set (8) indicate that any route violating the two-dimensional loading constraints is excluded from the feasible region. Constraint Set (9) enforces the *integrality* of the decision variables.

Constraints (8) call for identifying routes that are infeasible in terms of the loading constraints. Checking whether a route violates the loading constraints is equivalent to solving the SPP, a strongly-NP hard problem. Therefore, the existence of constraints (8) dramatically increases the difficulty of the original VRPTW. One possible method for solving Formulation (1)–(9) is a branch-and-cut algorithm akin to the algorithm proposed by Iori et al. (2007). Constraints (8) and (9) are relaxed to derive a lower bound of the original problem, which is embedded in a branch-and-bound tree. For an integer solution found in the tree, if some route γ associated with the solution violates the loading constraints, then a group of cuts $\sum_{a\in A(\gamma)} x_a^k \le |A(\gamma)| - 1, \forall k \in K$, is added to remove the integer solution from the feasible region. Otherwise, the integer solution is treated as a candidate to update the global upper bound. However, due to the big-M constraints brought by Constraints (5), the linear relaxation of the formulation provides a weak lower bound, leading to a large enumeration tree. Hence, we apply Danzig-Wolfe decomposition to the three-index formulation to derive the following set partitioning formulation (Desrosiers et al. 1995).

3.3. Set-Covering Formulation of the 2L-VRPTW

Let λ_r be the decision variables where

$$\lambda_r = \begin{cases} 1 & \text{if route } r \text{ is selected,} \\ 0 & \text{otherwise.} \end{cases}$$

Let $\tilde{d}_r$ be the total distance travelled on route r and Ω be the set of feasible routes. $a_{i,r}$ is a binary constant specifying whether customer i is visited in route r. It takes value 1 if i is visited, otherwise it takes value 0. We have the following formulation.

$$[\text{RMP}] \qquad \min_{r\in\Omega} \tilde{d}_r\lambda_r, \tag{10}$$

$$\text{s.t.} \quad \sum_{r\in\Omega}\lambda_r \le |K| \tag{11}$$

$$\sum_{r\in\Omega} a_{i,r}\lambda_r \ge 1 \qquad \forall i\in N \tag{12}$$

$$\lambda_r \in \{0,1\} \qquad \forall r\in\Omega \tag{13}$$

Constraint (11) limits the number of used vehicles. Constraint Set (12) indicates that each customer must be visited at least once, but it is guaranteed that in an optimal solution, each customer is visited exactly once. This is due to the assumption that the triangle inequality holds on graph G, which implies that a better solution can be obtained by removing repeated customers from routes while reducing the total distance. Therefore, Constraint Set (12) remains valid for the 2L-VRPTW. Finally, Constraint Set (13) defines the integrality of the variables.

Ω is a very huge collection of feasible routes. From the computational perspective, it is unpractical to enumerate all the routes in Ω to build Formulation (10)–(13). Therefore, a column generation method is used.

3.4. The Pricing Problem

The goal of solving the pricing problem is to obtain improving columns ("routes" and "columns" are interchangeable in the context) that have negative reduced cost. Let π_0 be the dual variable associated with constraint (11) and let π_j, $\forall j\in N$ be the dual variable associated with Constraint Set (12). We present the mathematical formulation of the pricing problem as follows.

3.4.1. Decision Variables.

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

$$[PP] \qquad \min_{(i,j)\in A} x_{ij}(c_{ij}-\pi_j)-\pi_0 \tag{14}$$

$$\text{s.t.} \quad \sum_{i\in N} x_{i0} = 1 \tag{15}$$

$$\sum_{j\in N} x_{0j} = 1 \tag{16}$$

$$\sum_{(i,j)\in A} x_{ij} = 1 \qquad \forall i\in N \tag{17}$$

$$\sum_{(j,i)\in A} x_{ji} = 1 \qquad \forall i\in N \tag{18}$$

$$s_i + u_i + c_{ij} - s_j + Mx_{ij} \le M \quad \forall (i,j)\in A \tag{19}$$

$$a_i \le s_i \le b_i \qquad \forall i\in N \tag{20}$$

$$\sum_{(i,j)\in A} x_{ij}q_j \le Q \tag{21}$$

$$\sum_{a\in A(\gamma)} x_a \le |A(\gamma)|-1, \quad \forall\gamma\in\bar{\Gamma}\backslash\Gamma \tag{22}$$

$$x_{ij}\in\{0,1\},\ \forall (i,j)\in A, \quad s_i\ge 0,\ \forall i\in N\cup\{0\} \tag{23}$$

The formulation resembles (1)–(9). It attempts to search for a valid route leaving from the depot and ending at the sink node in order to minimize the reduced cost (14). If Constraint Set (22) is relaxed, the problem is then reduced to a normal ESPPRC, which has been discussed in Desaulniers et al. (2006). We solve *[PP]* by partitioning it into two stages: (1) solving the ESPPRC resulting from relaxing Constraint Set (22) and (2) imposing feasibility check on new columns. The first stage is addressed by the traditional labeling algorithm with two new dominance rules, whereas the second stage is tackled by a packing algorithm from the literature.

3.4.2. The Labeling Algorithm. The labeling algorithm has been extensively used for the pricing problem (Desrochers et al. 1992, Feillet et al. 2004, Baldacci et al. 2011) owing to its practical performance. It is an extension of the Bellman-Ford-Fulkerson algorithm (Ahuja et al. 1995), storing and correcting partial routes from the source node in the form of labels. The main structure of our labeling algorithm comes from Feillet et al. (2004). A label L is defined as a 5-tuple $(\eta(L), t(L), q(L), c(L), \mu(L))$ with the following data: η—the node of the label, t—the arrival time, q—the consumed capacity, c—the accumulated cost, μ—the set of unreachable customers $[v_1, v_2, \ldots, v_n]$. Notation $P(L)$ represents the partial route indicated by label L. The label L, if "translated", means that the partial route $P(L)$ starts from the depot and arrives at the current node $\eta(L)$ at instant $t(L)$, having produced $c(L)$ cost, taking up $q(L)$ capacity, with no chance to extend itself to any node in $\mu(L)$. The labeling algorithm iteratively extends labels associated with each customer to reach the sink node. It performs in a breadth-first fashion. As a result, the efficiency of labeling algorithms highly relies on the ability to eliminate nonpromising labels. This gives rise to *dominance rules* based on which suboptimal paths can safely be removed during the search. Let $\varepsilon(L)$ denote the set of feasible paths extended from label L to the sink node. Let $\oplus$ denote concatenation between two paths. Let Ψ be a set of labels $\Psi(v_i) = \{L_1, L_2, \ldots, L_{|\Psi|}\}$ where all the labels arrive at the node v_i. The definition of dominance is as follows: A label L' is dominated and hence eliminated if $\forall e \in \varepsilon(L')$, there exists a label $L \in \Psi(\eta(L'))$ such that: (1) $P(L) \oplus e$ is feasible; and (2) $P(L) \oplus e$ has a lower or the same negative reduced cost. $L_i \prec L_j$ indicates that label L_i dominates label L_j. To check dominance relations by the definition is not efficient because all the possible extensions need to be enumerated. Instead, sufficient conditions that are computationally light are utilized as dominance rules. A classic dominance rule for the ESPPRC in VRPTW by Feillet et al. (2004) is as follows:

Lemma 1. *$L_i \prec L_j$ if the following conditions are satisfied:*

(1) $\eta(L_i) = \eta(L_j)$;
(2) $t(L_i) \le t(L_j)$;
(3) $q(L_i) \le q(L_j)$;
(4) $c(L_i) \le c(L_j)$;
(5) $\mu(L_i) \subseteq \mu(L_j)$.

If we consider the loading constraints, the difference made in the definition of a label stems from the representation of "consumed area." One scalar is no longer able to represent the consumption. Besides $q(L_i)$, we use $s(L_i)$ to indicate the set of dimensions of the items collected so far, which is amount to $[(w_{i1}, h_{i1}), (w_{i2}, h_{i2}), \ldots, (w_{i|s(L_i)|}, h_{i|s(L_i)|})]$. A label can thus be written as $(\eta(L), t(L), q(L), s(L), c(L), \mu(L))$. Before stating the new dominance rule, the following definitions are introduced.

Definition 1. Given two items, (w_i, h_i) and (w_j, h_j), if $w_i \le w_j$ and $h_i \le h_j$, then we denote $(w_i, h_i) \sqsubseteq (w_j, h_j)$.

Definition 2. Given two sets of items S_i and S_j, we say that S_i is covered by S_j if there exists a one-to-one mapping such that for any item $k \in S_i$, there exists an item $k' \in S_j$ such that $k \sqsubseteq k'$, and it is denoted as $S_i \sqsubseteq S_j$.

With the above definitions, the new dominance rule can be stated as follows:

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

Proof. Let $\varepsilon(L_j)$ be all the feasible extensions from L_j, which means $\forall e \in \varepsilon(L_j)$, $L_j \oplus e$ is feasible. Except for the packing restriction, it is trivial to see that $\forall e \in \varepsilon(L_j)$, $L_i \oplus e$ is also feasible in terms of the capacity and time windows constraints, because L_i has more resources. The only concern is whether the loading constraints are satisfied. We prove it by contradiction. Suppose there exists a feasible extension $e' \in \varepsilon(L_j)$ such that $L_i \oplus e'$ is infeasible in terms of the loading constraints. With the fourth condition holding, however, we can always find a feasible packing for $L_i \oplus e'$ via replacing items in $S(L_j)$ by items in $S(L_i)$ in the feasible packing of $L_j \oplus e'$. □

From Theorem 1, it is trivial to infer:

Corollary 1. *For two labels $L_i \prec L_j$, if there exists an extension e such that $L_i \oplus e$ is infeasible in terms of the loading constraints, then $L_j \oplus e$ is also infeasible.*

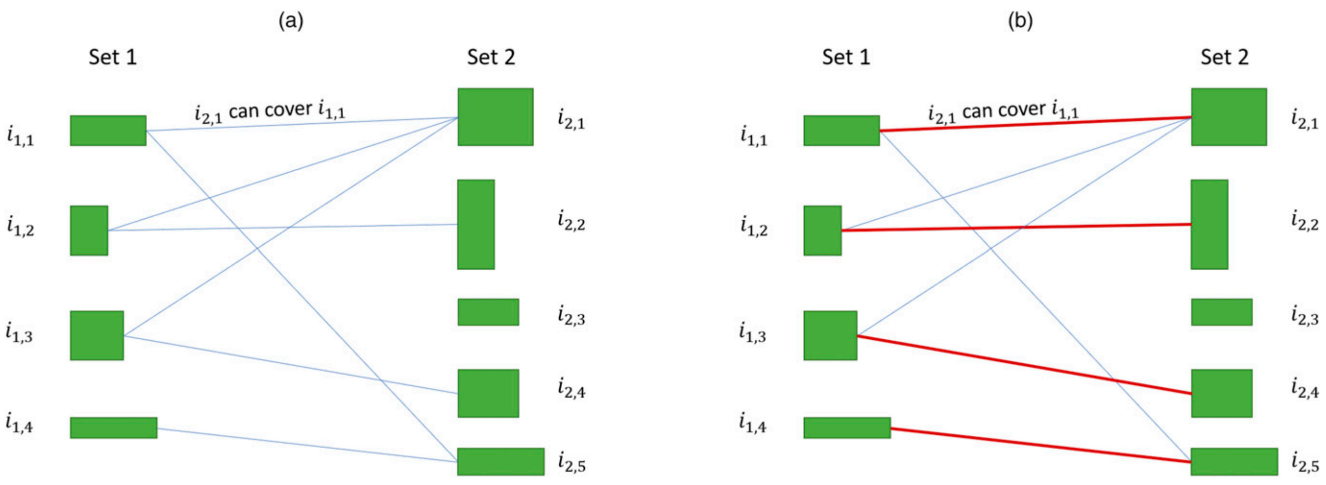
Dealing with the fourth condition in Theorem 1 can be treated as solving a maximum bipartite matching (MBM) problem, which can be described as follows. Given two sets of items $S_1 = \{i_{1,1}, i_{1,2}, \ldots, i_{1,m}\}$ $S_2 = \{i_{2,1}, i_{2,2}, \ldots, i_{2,n}\}$, where $m \le n$. If two items i, j respectively from S_1, S_2 have $(w_i, h_i) \sqsubseteq (w_j, h_j)$, then an edge will be drawn between them as shown by the bipartite graph in Figure 2(a). The goal is to maximize the number of edges selected while respecting the constraint that each item is linked to at most one selected edge. An optimal solution of the instance is presented in Figure 2(b). It is trivial to conclude that if the maximal number of edges is equal to the cardinality of S_1 then $S_1 \sqsubseteq S_2$. Fortunately, the MBM problem can be transformed into a maximal flow problem, which is polynomially solvable (Kleinberg and Tardos 2006).

However, the new dominance rule is predictably very weak, generating fewer dominated labels. Therefore, we also propose an approximate dominance rule by simply using the total area of items as their representation in a label. It is similar to the definition in Feillet et al. (2004) but adding "total area (denoted as $\nu(L)$)" as one of the label attributions, leading to the definition of a label $(\eta(L), t(L), q(L), \nu(L), c(L), \mu(L))$ as well as the following approximation dominance rule as Theorem 2. To distinguish from the exact dominance, an alternative notation $\ll$ is used to represent that L_j is approximately dominated by L_i.

Theorem 2. *$L_i \ll L_j$ if the following conditions are satisfied:*
(1) $\eta(L_i) = \eta(L_j)$;
(2) $t(L_i) \le t(L_j)$;
(3) $q(L_i)) \le q(L_j))$
(4) $\alpha\nu(L_i) \le \nu(L_j)$;
(5) $c(L_i) \le c(L_j)$;
(6) $\mu(L_i) \subseteq \mu(L_j)$.

Theorem 2 does not guarantee perfect dominance because a label with a smaller total area could possibly end up with no feasible extensions in terms of the loading constraints. On the contrary, a label with higher total area could be extended to a feasible complete path even though it has a larger negative reduced cost. Nevertheless, Theorem 2 has the merit of providing dominance rules that allow more labels to

Figure 2. (Color online) Demonstration for MBM



Note. (a) An instance of MBM; (b) a solution of MBM.

be dominated and that are computationally light. To make up possible *mis-dominance*, a control parameter α is introduced in condition 5 to manipulate the tightness of the dominance rule. The idea behind this is the following: in early stages of solving the pricing problem, with $\alpha = 1.0$, more labels can be dominated. When there is no improving route priced out with the current α, the parameter will be tuned up to weaken the dominance condition so that more labels can be released during the process.

3.4.3. Feasibility Checker. To ensure a priced-out route to satisfy the loading constraints, a recognition version of the SPP has to be solved. The SPP can be roughly described as follows: given a set of rectangular items and a strip with fixed width and height, determine if all the items can be packed in the strip (Martello et al. 2003). In our case, once a route r is constructed, we need to check whether the items collected along the route can be packed in the loading area. Alternatively, this can also be regarded as a bin packing problem in which we are looking for a solution with a single bin. To answer either question, checking the loading constraints is strongly-NP hard. Now that the process is a subroutine embedded in a column generation procedure, it only has a very limited budget for computation time. We implement the state-of-the-art exact algorithm for SPP named "BLUE" from Côté et al. (2014a) as the feasibility checker. Note that, a HashMap data structure is used to store checked routes that are infeasible to avoid repeated checks.

Finally, we obtain a column generation algorithm described in Algorithm 1, which consists of a labeling algorithm with the dominance rule of Theorem 2 and the exact packing algorithm BLUE. We refer to it as "pure column generation algorithm" PCGA throughout the paper. Regarding its exactness, we have the following statement based on Corollary 1.

Statement 1. *If Theorem 1 is applied in PCGA as the dominance rule, PCGA can solve "*[PP]*" optimally.*

Algorithm 1 (The Pure Column Generation Algorithm (PCGA))

```
1: Initialize $k = 0, \mathcal{C} = \varnothing, \mathcal{C}^k = \varnothing, \alpha = 1.0; \mathcal{H} = \varnothing$;
2: Generate the set of initial routes $\mathcal{C}^0, \mathcal{C} = \mathcal{C}^0, k \leftarrow k + 1$;
3: while True do
4:     Solve the master problem and let $x_k^*$ be an optimal solution;
5:     Build the pricing problem induced by $x_k^*$;
6:     Invoke the labeling algorithm to price out a set of columns $\mathcal{C}^k$;
7:     Filter out checked infeasible columns from $\mathcal{C}^k$ by querying $\mathcal{H}$;
8:     Set $\mathcal{C}_0^k = \varnothing$;
9:     for $r \in \mathcal{C}^k$ do
10:        if $r$ is found to be feasible by the feasibility checker then
11:            $\mathcal{C}_0^k = \mathcal{C}_0^k \cup \{r\}$;
12:        else
13:            Insert the key-value pair associated with $r$ into $\mathcal{H}$;
14:    if $\mathcal{C}_0^k$ is $\varnothing$ and $\alpha \le \bar{\alpha}$ then
15:        $\alpha \leftarrow \alpha + \Delta$;
16:        Continue;
17:    if $\mathcal{C}_0^k$ is $\varnothing$ and $\alpha > \bar{\alpha}$ then
18:        Break;
19:    $\mathcal{C} = \mathcal{C} \cup \mathcal{C}_0^k$;
20:    Update the master problem, $k \leftarrow k + 1$;
```

4. Acceleration Strategy

Running the feasibility checker on all columns is not necessary. One possible strategy is to let all columns enter the master without doing any feasibility check. Then we solve the master and only check columns selected into the optimal basis. If an infeasible column is detected, then a cut is added to the master and we solve the updated master problem until no infeasible column can be found in the optimal basis. This strategy would greatly decrease the number of feasibility checks. However, the master problem has to be solved many times due to hidden infeasible columns. As a result, once the master problem is large, the strategy becomes inefficient.

It can be envisaged that if there exists a predictor that blocks as many infeasible columns as possible from entering the master problem and permits as many feasible columns as possible to enter the master, then fewer infeasible columns enter the master and the above strategy can be accelerated. The following feasibility predictor is thus created to provide the prediction.

4.1. The Feasibility Predictor

In this subsection, the details of the feasibility predictor are demonstrated. First, the prediction task is formulated as a binary classification task. Second, candidate machine learning models to address the task are discussed. Third, the training algorithm is determined. Last but not the least, the evaluation criteria for the predictor is introduced.

4.1.1. The Binary Classification Task. It should be emphasized that what we are interested in is the recognition version of the SPP known as "two-dimensional orthogonal packing problem" (Clautiaux et al. 2007), asking whether a set of items can be packed in a given loading area. Therefore, the task can be regarded as binary classification. Classification is a type of machine learning task, where a computer program identifies to which of P classes a given input belongs. Binary classification is a special case when $p = 2$.

Let $(\mathcal{X}, \mathcal{Y})$ be a known data set where $\mathcal{X}$ stands for the set of inputs and $\mathcal{Y}$ represents the set of the corresponding desired outputs. A binary classifier is a model parameterized by θ that is required to approximate the distribution $\phi(y_i \mid \theta, x_i)$, $\forall x_i \in \mathcal{X}$, $\forall y_i \in \mathcal{Y}$ over the data set. Once the parameters θ are determined, a binary classifier is derived.

We apply the well-known maximum likelihood estimation (MLE) to estimate the parameters θ to make $\phi(y_i \mid \theta, x_i)$ match the observed set $(\mathcal{X}, \mathcal{Y})$.

$$\hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{|(\mathcal{X},\mathcal{Y})|} \phi(\theta, x_i)^{y_i}(1-\phi(\theta, x_i))^{1-y_i} \tag{24}$$

Because the right-hand-side of Equation (24) can easily cause numerical overflow, conventionally we take the natural log and obtain:

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{|(\mathcal{X},\mathcal{Y})|} (y_i \log \phi(\theta, x_i) + (1-y_i)\log(1-\phi(\theta, x_i)))$$

Because scaling a function does not alter its maximum point, the following equations can be obtained:

$$\hat{\theta} = \arg\max_{\theta} \frac{1}{|(\mathcal{X},\mathcal{Y})|} \sum_{i=1}^{|(\mathcal{X},\mathcal{Y})|} (y_i \log \phi(\theta, x_i) + (1-y_i)\log(1-\phi(\theta, x_i))) \tag{25}$$

$$\Longleftrightarrow$$

$$\hat{\theta} = \arg\min_{\theta} -\frac{1}{|(\mathcal{X},\mathcal{Y})|} \left[\sum_{i=1}^{|(\mathcal{X},\mathcal{Y})|} (y_i \log \phi(\theta, x_i) + (1-y_i)\log(1-\phi(\theta, x_i))) \right] \tag{26}$$

Hence, training a binary classifier parameterized by θ is equivalent to solving the following unconstrained optimization problem.

$$\min_{\theta \in \mathbb{R}} \mathcal{L}(\theta) = -\frac{1}{|(\mathcal{X},\mathcal{Y})|} \left[\sum_{i=1}^{|(\mathcal{X},\mathcal{Y})|} (y_i \log \phi(\theta, x_i) + (1-y_i)\log(1-\phi(\theta, x_i))) \right] \tag{27}$$

4.1.2. The Machine Learning Model. A machine learning model specifies the entity to make the inference $\hat{y}_i = \phi(\theta, x_i)$. There are a number of machine learning models capable of solving the binary classification task, including logistic regression (Kleinbaum et al. 2002), decision tree (Song and Ying 2015), k-nearest neighbors (Peterson 2009), support vector machine (Wang 2005) and artificial neural networks (Goodfellow et al. 2016), etc. According to the *no free lunch theorem* (Wolpert and Macready 1997), no machine learning model has *capacity* universally better than that of others, so model selection heavily depends on the characteristics of the task and the data.

Capacity of a machine learning model refers to the ability of the model to approximate a variety of functions or distributions. Generally a model with better capacity gives more precise predictions; however, the speed of running the model could be conflicted with the capacity. If a model performs inefficiently, its slowness would overshadow the benefits derived by introducing the model, thus slowing down the entire column generation algorithm. That is to say, we should select a model with sufficient capacity and acceptable speed.

In this study, we choose feedforward neural networks (FFNN) (Goodfellow et al. 2016) as the model for the following reasons.

1. High capacity. FFNN is the standard artificial neural network, which is composed of multiple layers of neurons. The neurons are activated by nonlinear functions that can facilitate the entire networks to learn very complex distributions. There are a number of hyperparameters, including the number of layers, the number of neurons at each layer, the type of activation function of the neurons, determined in advance. One can empirically tune the hyperparameters so that the FFNN learns the desired distribution. FFNN generally has good capacity according to an empirical study (Caruana and Niculescu-Mizil 2006).

2. Good flexibility. FFNN provides sufficient flexibility to balance the capacity and the speed by adjusting the hyperparameters. With fine tuning, one can find proper settings of the hyperparameters for the task at hand. In addition, FFNN forms the basis of many advanced network architectures including recurrent neural networks, convolution neural networks. Because the binary classification task has never been studied before, what architecture empirically fits the task the best is unknown. An extensible network could benefit future research.

3. Rich toolkits. There are many intelligible, robust and efficient software packages supporting FFNN, such as Pytorch (Paszke et al. 2019) developed by Facebook, TensorFlow (Abadi et al. 2016) developed by Google. The packages save a great amount of time for us from building the model, training the model, and testing the model.

4.1.3. The Training Algorithm.

A training algorithm specifies the method to solve Problem (27). In general, the problem could be solved by the classic gradient descent method (Lemaréchal 2012); however, calculating the mean gradient over all the entire set $(\mathcal{X},\mathcal{Y})$ is computationally expensive when the size of the training set is large. The method could also be easily trapped in *local minimum* (Bottou 1991). Therefore, the mini-batch gradient descent algorithm (MGDA) as shown in Algorithm 2, is chosen to solve Problem (27). Unlike the classic gradient descent, the MGDA samples a subset of training data and performs gradient descent accordingly. The MGDA converges much faster and it is also able to escape from local minimum. In practice, the MGDA is the most common optimization algorithm for training artificial neural networks (Bottou 1991, Goodfellow et al. 2016).

In Algorithm 2, an epoch means "a single pass over the entire training samples." If an epoch is finished, then the entire training samples $(\mathcal{X},\mathcal{Y})$ have been drawn. Because by passing the samples for multiple times, the loss in Problem (27) can be significantly reduced. Apparently, if the training samples are swept over for just once, that is, $E_{max} = 1$, then it is much likely that there is still much space to lessen the loss. That is to say, the resulting machine learning model does not fit the observed distribution $\phi(y_i \mid \theta, x_i)$, $\forall x_i \in \mathcal{X}$, $\forall y_i \in \mathcal{Y}$, well. Conversely, if the amount of the loss is too small, then it could lead to "overfitting," a very good approximation of the training samples but poor performance on unobserved samples. Therefore, in practice, the training samples $(\mathcal{X},\mathcal{Y})$ are split into two sets: a training set and a validation set. A machine learning model is trained to fit the training set while the model is also tested on the validation set to verify its generalization ability. Concretely, at the end of each epoch, the machine learning model parameterized by the newly updated θ, is run over the validation set, and thus the loss over the validation set (also called validation loss) is derived. Usually, E_{max} is determined based on preliminary results to make sure that after a sufficient number of epochs, the validation loss levels off.

Algorithm 2 (Mini-Batch Gradient Descent Algorithm)

```
1: Initialize Learning rate $\epsilon$, parameters $\theta$
2: for $e = 1, \ldots, E_{max}$ do
3:   while The $e^{th}$ epoch is not finished do
4:     Without replacement, randomly draw a set of $m$ samples from $(\mathcal{X},\mathcal{Y})$, $(x_1, x_2, \ldots x_i, \ldots, x_m)$ and the corresponding desired output $(y_1, y_2, \ldots, y_i, \ldots, y_m)$.
5:     Calculate the mean gradient over the mini-batch, $\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} \mathcal{L}(\theta, x_i, y_i)$
6:     Update the parameters, $\theta \leftarrow \theta - \epsilon\hat{g}$
```

4.1.4. The Evaluation Metric.

When a trained binary classifier is tested on a data set, a confusion matrix can be plotted as a basic metric for evaluation as shown by Table 1. There are in total four scenarios when an object comes into the classifier:

1. If the object is positive and the classifier identifies it as positive, then a *true positive* is obtained;

2. If the object is negative and the classifier identifies it as positive, then a *false positive* is obtained;

3. If the object is positive and the classifier identifies it as negative, then a *false negative* is obtained;

4. If the object is negative and the classifier identifies it as negative, then a *true negative* is obtained.

Table 1. Confusion Matrix

	Classified as positive	Classified as negative
Positive	True positive	False negative
Negative	False positive	True negative

Table 2. Classes Used for the Generation of Items

		Vertical		Homogeneous		Horizontal	
Packing class	$\lvert M_i \rvert$	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]

The following four ratios can be computed in row-wise out of the matrix:

$$r_{11} \approx \frac{\text{Total true positives}}{\text{Total positives}} \qquad r_{10} \approx \frac{\text{Total false negative}}{\text{Total positives}}$$

$$r_{01} \approx \frac{\text{Total false positive}}{\text{Total negatives}} \qquad r_{00} \approx \frac{\text{Total true negatives}}{\text{Total negatives}}$$

It is trivial to see that $r_{11} + r_{10} = 1.0, r_{01} + r_{00} = 1.0$. The ratios r_{11}, r_{00} of a good binary classifier should be as high as possible.

Another simpler metric is called classification accuracy or prediction accuracy, which is the ratio of the summation of the total true positives and the total true negatives and the total number of samples.

4.2. Feasibility Predictor in Column Generation

Figure 3 shows the diagram of integrating the feasibility predictor with column generation. A set of columns is generated by the labeling algorithm. These columns are then fed to the feasibility predictor to be predicted as either "feasible" or "infeasible."

As the feasibility predictor may produce wrong labels, two measures are taken. Columns being predicted as infeasible are passed to the feasibility checker rather than being discarded directly in order to rescue feasible columns that have been incorrectly predicted as infeasible. Conversely, columns predicted as feasible are added to the master without being checked by the feasibility checker. As a consequence, this could introduce some infeasible columns into the master. To address the issue, each time when an optimal solution of the master problem is obtained, all columns in the basis are checked by the feasibility checker. Once we find an infeasible column in the basis, the associated *false-positive cut* is added to the master problem to remove the column.

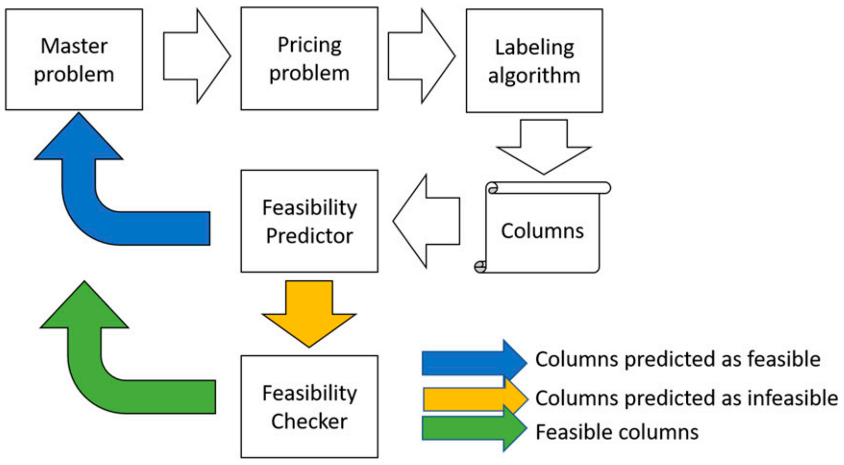
Definition 3. A column λ in the master problem detected as infeasible, the false-positive cut associated with the column is $x_\lambda \leq 0$, where x_λ is the corresponding variable.

Plugging the "Feasibility Predictor" as in Figure 3, we obtain the "integrated column generation algorithm" (ICGA) (see Algorithm 3 for the pseudocodes).

Statement 2. *If Theorem 1 is applied to ICGA, ICGA can solve (PP) optimally.*

Proof. Because we already have Statement 1, to prove its optimality, we only need to prove that introducing false-positive cuts does not lead to missing feasible improving columns. Suppose a cut associated with

Figure 3. (Color online) Diagram of ICGA



path r is added to the master problem, with Corollary 1, it is guaranteed that all the paths that are dominated by r are also infeasible. □

Algorithm 3 (The Integrated Column Generation Algorithm (ICGA))

```
1: Initialize $k = 0, \mathcal{C} = \emptyset, \mathcal{C}^k = \emptyset, \alpha = 1.0, \mathcal{H} = \emptyset$;
2: Generate the set of initial routes $\mathcal{C}^0, \mathcal{C} = \mathcal{C}^0, k \leftarrow k + 1$;
3: while True do
4:     while True do
5:         Solve the master problem and let $x_k^*$ be an optimal solution;
6:         if There exists an infeasible column $r$ in the basis then
7:             Add a cut to prohibit the column from being selected in the basis;
8:             Update the master problem;
9:             Insert the key-value pair associated with $r$ into $\mathcal{H}$;
10:        else
11:            Break;
12:    Build the pricing problem by $x_k^*$;
13:    Invoke the labeling algorithm to price out a set of columns $\mathcal{C}^k$;
14:    Filter out checked infeasible columns from $\mathcal{C}^k$ by looking for $\mathcal{H}$;
15:    Set $\mathcal{C}_0^k, \mathcal{C}_1^k = \emptyset$;
16:    for $r \in \mathcal{C}^k$ do
17:        if $r$ is found to be feasible by the feasibility predictor then
18:            $\mathcal{C}_0^k = \mathcal{C}_0^k \cup \{r\}$;
19:        else
20:            $\mathcal{C}_1^k = \mathcal{C}_1^k \cup \{r\}$;
21:    for $r \in \mathcal{C}_1^k$ do
22:        if $r$ is judged as feasible by the feasibility checker then
23:            $\mathcal{C}_0^k = \mathcal{C}_0^k \cup \{r\}$;
24:        else
25:            Insert the key-value pair associated with $r$ into $\mathcal{H}$;
26:    if $\mathcal{C}_0^k$ is $\emptyset$ and $\alpha \le \bar{\alpha}$ then
27:        $\alpha \leftarrow \alpha + \Delta$;
28:        Continue;
29:    if $\mathcal{C}_0^k$ is $\emptyset$ and $\alpha > \bar{\alpha}$ then
30:        Break;
31:    $\mathcal{C} = \mathcal{C} \cup \mathcal{C}_0^k$;
32:    Update the master problem, $k \leftarrow k + 1$.
```

4.3. The Branch-and-Price Algorithm

ICGA is nested in a B&B tree to achieve a B&P algorithm. The branching rule adopted in this study is the standard "branch on arcs" (Feillet 2010). We choose best-bound first as the node selection strategy because it is more robust to inefficient selection in the early stages of the tree search.

Infeasible columns and false-positive cuts should be cleared at each node of the search tree to keep the size of the master problem as small as possible.

Two versions of B&P algorithms are created, namely B&P with the approximate dominance rule (B&P-AD) and B&P with the exact dominance rule (B&P-ED). For the latter, the exact dominance rule is only invoked when the approximate dominance rule fails to find improving columns. Moreover, we have also adapted the Clarke and Wright's savings algorithm (Clarke and Wright 1964) for the loading constraints to generate good routes at the beginning of both B&P algorithms. Besides the weight capacity, a link between two customers has to respect the loading constraints. Otherwise, the link is rejected.

5. Computational Experiments

This section describes the computational experiments performed for ICGA and the B&P algorithms. The feasibility predictor (FP) was implemented in Python with dependency on Pytorch. The column generation and the B&P algorithms were implemented in C++ based on CPLEX 12.9 as the LP solver with the default settings. The FP, after being trained in Python, was transformed to a format that is loadable in C++.[1] A CPU time limit of 3,600 seconds is set for all the experiments if not specified (the training time for the FP is not included).

All the training was performed on Google Colab equipped with NVIDIA Tesla K80 as the GPU. The ICGA and the B&P algorithms were run on an Intel Gold 6148 Skylake (eight cores) at 2.4 GHz under CentOS Linux 7.8 with 8 GB as the memory limit.

5.1. Instances Generation

Instances are divided into two different sets: Type I and Type II.

As for Type I, the instances are used to generate training samples for the FP. Each instance is defined by its geographic feature, time windows feature and items feature.

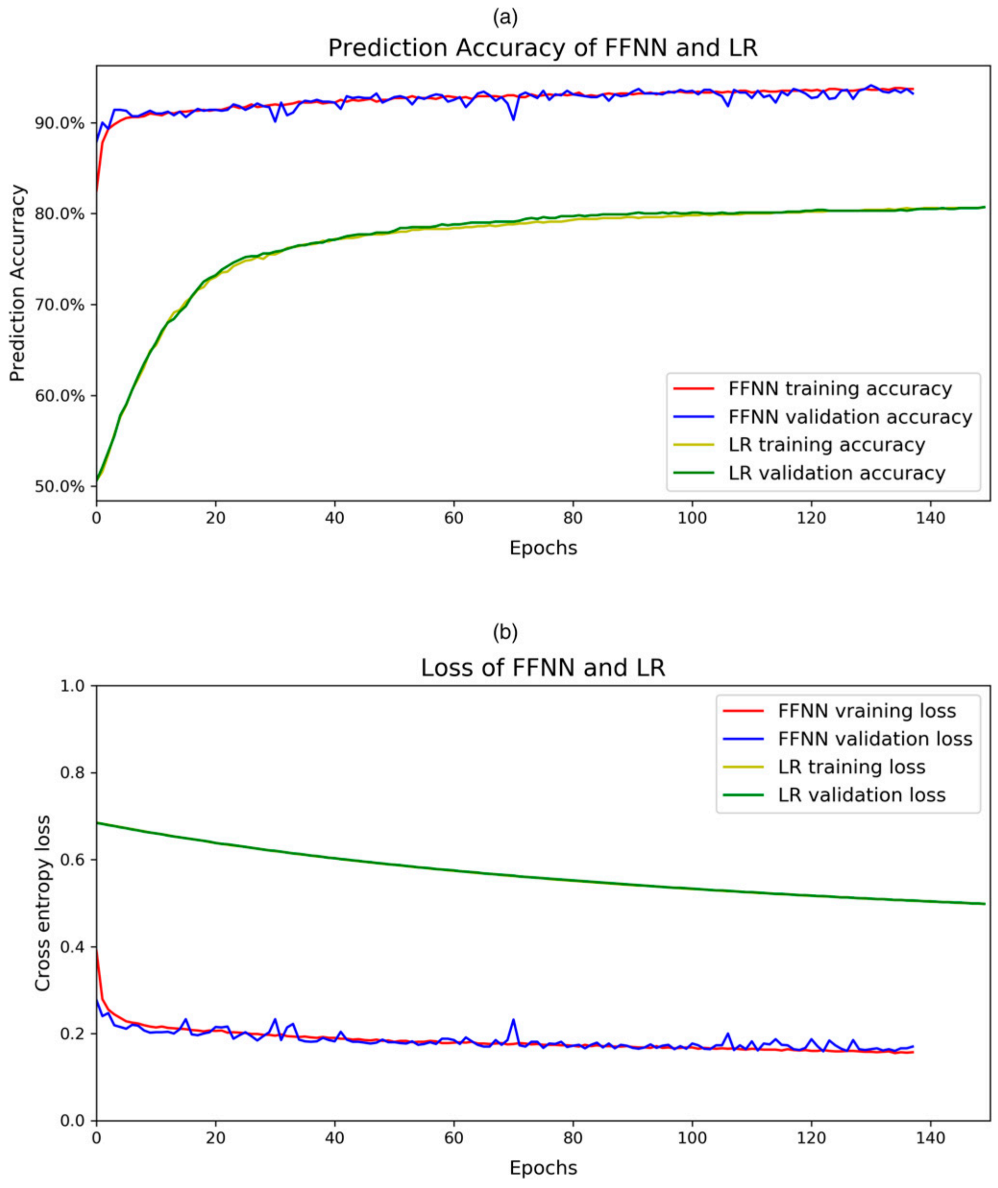
5.1.1. Geographic Feature. The geographic feature specifies the locations of the depot and customers. There are three cases: Random (denoted as R), where customers are randomly distributed in the 2-D plane; Clustered (denoted as C), where customers are grouped into a small number of clusters; Mixed (denoted as RC), where the locations are generated in a hybrid manner. For R instances, we set the depot at (35, 35) and generate coordinates by sampling from discrete uniform distribution $U \sim [0, 100]$. For C instances, the depot is set at (40, 50). Coordinates of the centers of each cluster are sampled from discrete uniform distribution $U \sim [10, 90]$. The number of customers in each cluster follows the discrete uniform distribution $U \sim [8, 9]$. For each cluster, coordinates of the customers are generated around its center with a given diameter sampled from discrete uniform distribution $U \sim [3, 5]$. For RC instances, half of customers are clustered and the others are randomly distributed.

5.1.2. Time Windows Feature. For each of the aforementioned three cases, there are two scenarios—narrow time windows and wide time windows determined by b_0. For generating the time windows of customers, there are two main steps. The first step is to fix the center of a time window, sampled from the uniform distribution $[a_0 + d_{0,j}, b_0 - d_{j,0} - u_j]$ for customer j. The second step is to generate the width of a time window, sampled from normal distribution $N \sim (\beta b_0/2, \delta)$.

5.1.3. Items Feature. Items are generated based on table 2 from Iori et al. (2007). Because packing class 1 (PC1) is equivalent to the one-dimensional case, it is excluded from our study. Other packing classes are characterized by the number of items and the dimensions (height and width).

Type II is composed of two groups of instances. The first group, namely Type II-Sol, is created to compare PCGA and ICGA. The instances are built on the well-known Solomon benchmark instances of Solomon (1987), in the same way as in Iori et al. (2007) for the

Figure 4. (Color online) Comparisons Between FFNN and LR



Note. (a) FFNN versus LR (by prediction accuracy); (b) FFNN versus LR (by loss).

generation of items. Although there is an existing group of 2L-VRPTW instances matching our problem, we think that the Solomon instances provide richer features regarding time window constraints.

The second group, namely Type II-Kh, is the instances from the literature created by Khebbache-Hadji et al. (2013), which are used to show the solution quality of the two B&P algorithms. All the instances, training samples as well as the FP can be found either in the online supplement or on the website.[2]

5.2. Training the FP

By running the PCGA on all the instances of Type I, we collect nearly 50,000 training samples. We process the raw input into 17 hand-crafted features as the input to the FFNN including: the ratio of total area of items to *WH*; the mean value, standard deviation, maximum and minimum of the following four indicators, respectively.

1. The ratio of items width to height;
2. The ratio of items width to *W*;
3. The ratio of items height to *H*;
4. The ratio of area of items to *WH*;

Because the binary classification task has never been investigated, we need to perceive whether it is a difficult task or not. To this end, logistic regression (LR) is applied as a base line. The LR model also works as a comparator for the FFNN. The FFNN model and the LR model were fed with the training samples that were generated by instances of packing class 2 (PC2). Training loss and validation loss are the loss on the training set and the validation set, respectively. Training accuracy and validation accuracy are the prediction accuracy on the training set and the validation set, respectively. Figure 4(c) illustrates the training loss, validation loss, training accuracy, and prediction accuracy of the FFNN and LR. The models were trained for 150 epochs in maximal with such early stopping criterion that if during the last 50 epochs the validation loss cannot be improved, the training is terminated. Stochastic gradient descent is used as the training algorithm. It took 6,009 seconds and 5,835 seconds to train the FFNN and the LR, respectively. The FFNN reaches a validation accuracy as high as 94.1% and mostly remains above 90%.

The trained FFNN model was tested on different packing classes, which were not "seen" by the model. Figure 5 shows a solid evidence that the model has good generalization capability to instances of other packing classes. For PC3, its prediction accuracy is quite satisfying, over 90%. For PC4, the model also achieves good performance, especially good at predicting feasible columns. PC5 is quite special in the sense that no infeasible columns are generated. Nonetheless, the model is able to capture the characteristics of the instances and has almost perfect performance (99.97% accuracy). Appendix C in the online supplement shows the detailed hyperparameters of the FFNN.

5.3. Experiments on the Column Generation Algorithms

Comparisons between ICGA and PCGA are conducted to demonstrate the contribution of the FP to the column generation algorithm on all Type II-Sol instances. If not specified, otherwise all the gaps regarding the comparisons are computed as $\%_{gap} = 100(z_{ICGA} - z_{PCGA})/z_{PCGA}$, where z is the objective value, CPU time, or number of calls of the feasibility checker associated with the corresponding algorithms. The results are reported in

Figure 5. (Color online) Prediction Accuracy on Other Packing Classes

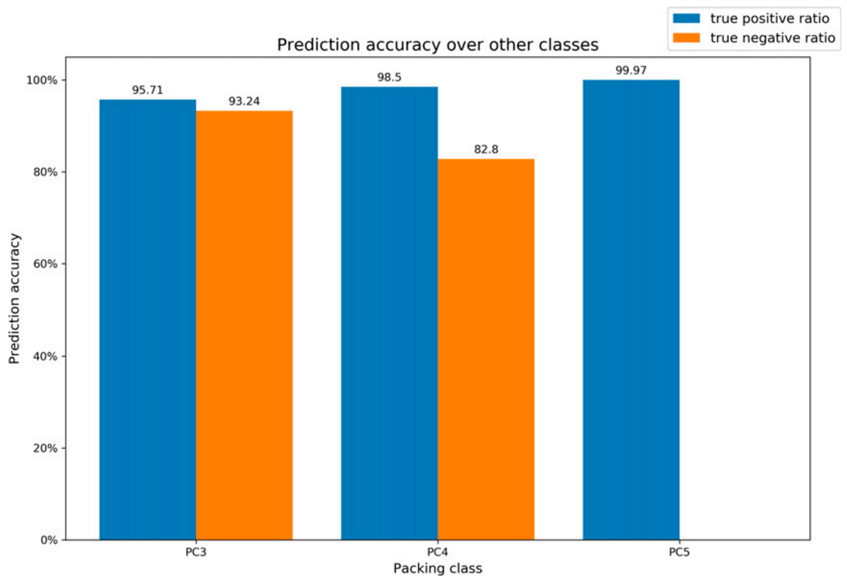


Table 3. ICGA vs. PCGA by Packing Class and Number of Customers

Packing class	No customers	*O-Gap*	*C-Gap*	*T-Gap*
PC2	25	0.08%	−22.20%	−69.04%
PC2	50	−0.33%	−30.54%	−66.56%
PC3	25	0.01%	−70.63%	−88.64%
PC3	50	−0.06%	−27.83%	−89.25%
PC4	25	−0.10%	−62.74%	−94.77%
PC4	50	−0.17%	−55.12%	−94.40%
PC5	25	−0.73%	−88.71%	−97.63%
PC5	50	−0.14%	−75.17%	−97.48%

Tables 3 and 4. For both tables, *O-Gap* indicates the percentage gap of best objective values derived by the two algorithms. *C-Gap* indicates the percentage gap of CPU time. *T-Gap* indicates the percentage gap of the number of calls for the feasibility checker.

From Table 3, it can be observed that: (1) across all kinds of settings, ICGA can obtain slightly better objective values on average; (2) ICGA needs 54% less CPU time on average; (3) for all types of instances, the feasibility checker is far less invoked by ICGA as shown by the column T-Gap in Table 3; overall, the number of calls is reduced by about 87%.

Among all packing classes, the least CPU time savings are observed for PC2 and the largest for PC5. This is mainly attributed to the fact that PC2 instances feature larger items. When the majority of items are large, more infeasible columns are priced out by the labeling algorithm due to a poor estimation on feasibility by total area of items. As there are more infeasible columns, the feasibility checker has to be invoked more frequently. At the opposite, PC5 has a lot of small items, in which case the total area of items provides a good estimate of feasibility. Based on our preliminary experiments, infeasible columns are rarely priced out. For this reason, the feasibility checker is invoked much less due to the nature of the proposed column generation mechanism (Figure 3). There is an interesting observation that ICGA can derive a slightly better objective value. This might be due to a slightly different dual space resulting from the introduction of infeasible columns into the master problem. Table 4 provides results related to the geographic feature. The O-Gap and C-Gap are both affected. We can see a significant O-gap in the case of "Mixed" instances. With respect to CPU time, ICGA shows better performance

Table 4. ICGA vs. PCGA by Instance Type

Geometric characteristics	*O-Gap*	*C-Gap*	*T-Gap*
Clustered	−0.06%	−44.50%	−87.85%
Random	−0.03%	−70.0%	−89.06%
Mixed	−0.26%	−59.2%	−86.21%

Table 5. FFNN vs. Logistic Regression

Packing class	NO customers	*O-Gap*	*C-Gap*	*T-Gap*
PC2	25	0.24%	−26.72%	−56.07%
PC2	50	−0.32%	−20.16%	−52.68%
PC3	25	0.03%	−38.55%	−84.0%
PC3	50	−0.09%	−9.25%	−85.69%
PC4	25	−0.04%	−63.83%	−92.93%
PC4	50	−0.15%	−73.01%	−92.69%
PC5	25	0.00%	−83.45%	−96.77%
PC5	50	−0.14%	−73.91%	−96.68%

in the case of "Random" instances. As customers become more clustered, the CPU time savings decrease.

Whether the prediction accuracy of the FP affects the ICGA is one of the major concerns. Therefore, another version of ICGA based on a logistic regression model was built for further comparison. To make the difference more distinguishable, we trained a logistic regression model with a significantly lower prediction accuracy (it turned out to be 64.5%). Then it is run on Type II-Sol instances comparing the results obtained by ICGA with the FFNN. In Table 5, columns O-Gap, C-Gap, and T-Gap are associated with ICGA with FFNN and ICGA with the logistic regression model. As shown by Table 5, with higher prediction accuracy, the ICGA with FFNN completely dominates the counterpart with the logistic regression on both CPU time and feasibility check times. With regards to objective values, ICGA with FFNN has a slight lead on average, but it seems that the prediction accuracy of a model does not have a large impact on solution quality.

5.4. Experiments on the Branch-and-Price Algorithm

This part of the experiments is conducted to test the ability to prove optimality of the exact branch-and-price algorithm (B&P-ED), as the first goal and meanwhile to illustrate the performance of the approximate B&P algorithm, namely B&P-AD, which has more value from the practical viewpoint. Both B&P algorithms are tested on Type II-Kh instances. There are three groups of Type II-Kh instances including class A, class B, and class C categorized by tightness of time windows (nondecreasing order of tightness). For Table 6, Table B.1, and Table B.2, *Obj* indicates the best objective value derived by an algorithm. A bold number indicates that the corresponding algorithm returns the best objective value among all the algorithms. A number with an asterisk means that the optimality of the objective value is proved. Column *Kh-MA* in Table 6 lists the solutions derived by the memetic heuristic proposed by Khebbache-Hadji et al. (2013). However, in Tables B.1 and B.2, because the results of the memetic heuristic on class B and class C instances were not reported in Khebbache-Hadji et al. (2013), we compare

Table 6. Results for Type II-Kh Instances of Class A

Instance	No customers	No items	B&P-AD Obj	CPU time	B&P-ED Obj	CPU time	Kh-MA[a] Obj
A102	15	24	**320.2**	0.3	320.2[b]	0.9	352.2
A103	15	24	**320.2**	0.2	320.2[b]	0.2	328.6
A202	20	29	**454.8**	1.6	454.8[b]	4.0	492.9
A203	20	46	**462.8**	27.6	462.8[b]	22.8	501.2
A302	21	31	**527.8**	2.8	527.8[b]	2.8	546.3
A303	21	37	**527.8**	0.4	527.8[b]	0.7	541.2
A402	22	32	**849.0**	2.8	849.0[b]	2.1	912.9
A403	22	41	**819.6**	16.4	819.6[b]	19.3	868.7
A502	25	40	**610.4**	2.9	610.4[b]	15.6	673.2
A503	25	61	**602.3**	24.8	602.3[b]	21.9	668.4
A602	29	43	**842.3**	14.8	842.3[b]	26.0	927.4
A603	29	49	**806.4**	4.8	806.4[b]	5.1	877.2
A702	30	56	**544.7**	11.1	544.7[b]	28.7	594.8
A703	30	82	**561.4**	28.8	561.4[b]	29.4	643.1
A802	50	82	**856.6**	161.9	858.7	7,235.1	950.6
A803	50	103	**842.5**	56.4	842.5[b]	178.2	970.5
A902	75	112	**1,252.2**	181.5	N/A	N/A	1,409.3
A903	75	155	**1,251.1**	3,664.0	N/A	N/A	1,387.1
A1002	100	157	**1,654.7**	3,887.0	N/A	N/A	1,821.2
A1003	100	212	**1,549.4**	3,702.7	N/A	N/A	1,838.2
Average value			782.8	589.6	N/A	N/A	865.2

Note. A bold number indicates that the corresponding algorithm returns the best objective value among all the algorithms.
[a]The average CPU time is 205.5 seconds.
[b]The optimality of the objective value is proved.

our algorithms with the two heuristics in Khebbache et al. (2009).

For B&P-ED, we prolonged the execution time of the feasibility checker and the entire time limit with the hope that it could solve as many instances to optimality as possible. Table 6 demonstrates that all instances of class A with 50 customers or less except "A802" are solved optimally. For class B and class C, all instances with 30 or fewer customers and 84 or fewer items are closed (see Appendix B in the online supplement). In comparison, B&P-AD cannot find the optimal solutions in only three instances out of 43 that are solved optimally by the B&P-ED, which provides a substantial evidence that the solutions returned by the B&P-AD algorithm are very close to the optimal ones. Additionally, B&P-AD algorithm manages to improve solutions of all the instances shown in Table 6 and the average improvement is as significant as 10.74%. The large amount of improvement also happens for class B and class C as shown in Tables B.1 and B.2 in Online Appendix.

6. Conclusions

In this study, we have addressed a vehicle routing problem with two-dimensional loading constraints. A new exact dominance rule and an approximate dominance rule were developed to solve the pricing problem. We proposed a new column generation mechanism by injecting a supervised learning model. Experimental results indicate that this new algorithm performs better than that without the supervised learning model. Two branch-and-price algorithms were built based on the new column generation algorithm. Several instances from literature were solved to optimality for first time. Most solutions were improved significantly as well.

We proposed a strategy to make better use of packing algorithms for loading constraints in the context of vehicle routing. To extend the idea, there could be two directions for further studies. One is developing machine learning models for more complex situations such as vehicle routing problems with unloading constraints. The other is exploring ways to plug a predictor in other algorithms such as branch-and-cut algorithm or meta-heuristics.

Acknowledgments

The authors thank Rahul Patel for insightful discussion on the project and two anonymous referees for valuable comments.

Endnotes

[1] See https://pytorch.org/tutorials/advanced/cpp_export.html.

[2] See https://github.com/arccos0/Feasibility-Predictor-for-two-dimensional-loading-constraints.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, et al (2016) TensorFlow: Large-scale machine learning on heterogeneous distributed systems. Preprint, submitted March 14, https://arxiv.org/abs/1603.04467.

Ahuja RK, Magnanti TL, Orlin JB, Weihe K (1995) Network flows: theory, algorithms and applications. *ZOR-Methods Models Oper. Res.* 41(3):252–254.

Alvarez AM, Louveaux Q, Wehenkel L (2017) A machine learning-based approximation of strong branching. *INFORMS J. Comput.* 29(1):185–195.

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

Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.* 290(2):405–421.

Bonami P, Lodi A, Zarpellon G (2018) Learning a classification of mixed-integer quadratic programming problems. van Hoeve W-J, ed. *Proc. Internat. Conf. Integration Constraint Programming, Artificial Intelligence, Oper. Res.* (Springer, Delft, Netherlands), 595–604.

Bortfeldt A, Homberger J (2013) Packing first, routing second-a heuristic for the vehicle routing and loading problem. *Comput. Oper. Res.* 40(3):873–885.

Bottou L (1991) Stochastic gradient learning in neural networks. *Proc. Neuro-N Times* 91(8):12.

Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. *Proc. 23rd Internat. Conf. Machine Learn.* (Association for Computing Machinery, New York), 161–168.

Clarke G, Wright JW (1964) Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res.* 12(4):568–581.

Clautiaux F, Carlier J, Moukrim A (2007) A new exact method for the two-dimensional orthogonal packing problem. *Eur. J. Oper. Res.* 183(3):1196–1211.

Cordeau JF, Iori M, Laporte G, Salazar González JJ (2010) A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with LIFO loading. *Networks* 55(1):46–59.

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

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

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

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

Côté JF, Guastaroba G, Speranza MG (2017) The value of integrating loading and routing. *Eur. J. Oper. Res.* 257(1):89–105.

Crainic TG, Perboli G, Tadei R (2008) Extreme point-based heuristics for three-dimensional bin packing. *INFORMS J. Comput.* 20(3): 368–384.

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

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

Desrosiers J, Dumas Y, Solomon MM, Soumis F (1995) Time constrained routing and scheduling. *Handb. Oper. Res. Management Sci.* 8:35–139.

Feillet D (2010) A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR* 8(4):407–424.

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* 44(3):216–229.

Fuellerer G, Doerner KF, Hartl RF, Iori M (2009) Ant colony optimization for the two-dimensional loading vehicle routing problem. *Comput. Oper. Res.* 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 Sci.* 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* 51(1):4–18.

Goodfellow I, Bengio Y, Courville A (2016) *Deep Learning* (MIT Press, Cambridge, MA).

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

Iori M, de Lima VL, Martello S, Miyazawa FK, Monaci M (2021) Exact solution techniques for two-dimensional cutting and packing. *Eur. J. Oper. Res.* 289(2):399–415.

Jia H, Shen S (2019) Benders cut classification via support vector machines for solving two-stage stochastic programs. Preprint, submitted June 14, https://arxiv.org/abs/1906.05994.

Khalil E, Dai H, Zhang Y, Dilkina B, Song L (2017) Learning combinatorial optimization algorithms over graphs. *Adv. Neural Inf. Process. Syst.* 30:6348–6358.

Khalil EB, Le Bodic P, Song L, Nemhauser G, Dilkina B (2016) Learning to branch in mixed integer programming. *30th AAAI Conf. Artificial Intelligence.* (AAAI Press, Phoenix), 724–731.

Khebbache S, Prins C, Yalaoui A, Reghioui M (2009) Heuristics for two-dimensional loading capacitated vehicle routing problem with time windows. *IFAC Proc. Volumes* 42(4):1544–1549.

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. *CEJOR Cent. Eur. J. Oper. Res.* 21(2):307–336.

Kleinbaum DG, Dietz K, Gail M, Klein M, Klein M (2002) *Logistic Regression* (Springer, New York).

Kleinberg J, Tardos E (2006) *Algorithm Design* (Pearson Education India).

Kool W, van Hoof H, Welling M (2018) Attention, learn to solve routing problems! Preprint, submitted March 22, https://arxiv.org/abs/1803.08475.

Kruber M, Lübbecke ME, Parmentier A (2017) Learning when to use a decomposition. *Internat. Conf. AI OR Techniques Constraint Programming Combin. Optim. Problems* (Springer), 202–210.

Lemaréchal C (2012) Cauchy and the gradient method. *Doc Math Extra* 251(254):10.

Mahvash B, Awasthi A, Chauhan S (2017) A column generation based heuristic for the capacitated vehicle routing problem with three-dimensional loading constraints. *Internat. J. Prod. Res.* 55(6):1730–1747.

Martello S, Monaci M, Vigo D (2003) An exact approach to the strip-packing problem. *INFORMS J. Comput.* 15(3):310–319.

Martínez L, Amaya CA (2013) A vehicle routing problem with multi-trips and time windows for circular items. *J. Oper. Res. Soc.* 64(11):1630–1643.

Nair V, Dvijotham D, Dunning I, Vinyals O (2018) Learning fast optimizers for contextual stochastic integer programs. *UAI*, 591–600.

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, et al. (2019) Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inform. Processing Systems* 32:8026–8037.

Peterson LE (2009) K-nearest neighbor. *Scholarpedia* 4(2):1883.

Pinto T, Alves C, de Carvalho JV (2013) Column generation based heuristic for a vehicle routing problem with 2-dimensional loading constraints: a prototype. *XI Congresso Galego Estatística Investigación Operacións*, A Coruña, Spain.

Pinto T, Alves C, de Carvalho JV (2015) Variable neighborhood search for the elementary shortest path problem with loading constraints. Gervasi O, Murgante B, Misra S, Gavrilova ML, Rocha AMAC, Torre C, Taniar D, Apduhan BO, eds. *Proc. Internat. Conf. Comput. Sci. Its Appl.* (Springer, Banff, Alberta, Canada), 474–489.

Pinto T, Alves C, de Carvalho JV (2016) A branch-and-price algorithm for the vehicle routing problem with 2-dimensional loading constraints. Paias A, Ruthmair M, Voß S, eds. *Internat. Conf. Comput. Logist.* (Springer International Publishing, Lisbon, Portugal), 321–336.

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* 51(3):155–170.

Smith KA (1999) Neural networks for combinatorial optimization: A review of more than a decade of research. *INFORMS J. Comput.* 11(1):15–34.

Solomon MM (1987) Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.* 35(2):254–265.

Song YY, Ying L (2015) Decision tree methods: Applications for classification and prediction. *Shanghai Jingshen Yixue* 27(2):130.

Song X, Jones D, Asgari N, Pigden T (2019) Multi-objective vehicle routing and loading with time window constraints: A real-life application. *Ann. Oper. Res.* 291:1–27.

Tang Y, Agrawal S, Faenza Y (2019) Reinforcement learning for integer programming: Learning to cut. Preprint, submitted June 11, https://arxiv.org/abs/1906.04859.

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

Toth P, Vigo D (2002) An overview of vehicle routing problems. *The Vehicle Routing Problem* (SIAM), 1–26.

Václavík R, Novák A, Šcha P, Hanzálek Z (2018) Accelerating the branch-and-price algorithm using machine learning. *Eur. J. Oper. Res.* 271(3):1055–1069.

Wang L (2005) *Support Vector Machines: Theory and Applications*, vol. 177 (Springer Science & Business Media).

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* 1(1):67–82.

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 Res. Part B: Methodological* 82:20–35.