This article was downloaded by: [222.205.46.89] On: 25 October 2021, At: 23:21
Publisher: Institute for Operations Research and the Management Sciences (INFORMS)
INFORMS is located in Maryland, USA

Transportation Science

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

Machine-Learning-Based Column Selection for Column Generation

Mouad Morabit, Guy Desaulniers, Andrea Lodi

To cite this article:
Mouad Morabit, Guy Desaulniers, Andrea Lodi (2021) Machine-Learning-Based Column Selection for Column Generation. Transportation Science 55(4):815-831. https://doi.org/10.1287/trsc.2021.1045

Full terms and conditions of use: https://pubsonline.informs.org/Publications/Librarians-Portal/PubsOnLine-Terms-and-Conditions

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval, unless otherwise noted. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article's accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2021, INFORMS

Please scroll down for article—it is on subsequent pages

With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org

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

TRANSPORTATION SCIENCE
Vol. 55, No. 4, July–August 2021, pp. 815–831
ISSN 0041-1655 (print), ISSN 1526-5447 (online)

Machine-Learning–Based Column Selection for Column Generation

Mouad Morabit,[a,b,c] **Guy Desaulniers,**[a,b] **Andrea Lodi**[a,b,c]

[a] Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Québec H3C 3A7, Canada; [b] Research Group in Decision Analysis (GERAD), Montréal, Québec H3T 2A7, Canada; [c] Canada Excellence Research Chair in Data Science for Real-Time Decision-Making, Polytechnique Montréal, Quebec H3C 3A7, Canada

Contact: mouad.morabit@gerad.ca (MM); guy.desaulniers@gerad.ca, https://orcid.org/0000-0003-4469-9813 (GD); andrea.lodi@polymtl.ca (AL)

Received: May 1, 2020
Revised: November 3, 2020
Accepted: December 22, 2020
Published Online in Articles in Advance: June 30, 2021

https://doi.org/10.1287/trsc.2021.1045

Copyright: © 2021 INFORMS

Abstract. Column generation (CG) is widely used for solving large-scale optimization problems. This article presents a new approach based on a machine learning (ML) technique to accelerate CG. This approach, called *column selection*, applies a learned model to select a subset of the variables (columns) generated at each iteration of CG. The goal is to reduce the computing time spent reoptimizing the restricted master problem at each iteration by selecting the most promising columns. The effectiveness of the approach is demonstrated on two problems: the vehicle and crew scheduling problem and the vehicle routing problem with time windows. The ML model was able to generalize to instances of different sizes, yielding a gain in computing time of up to 30%.

Funding: This work was supported by the Natural Sciences and Engineering Research Council of Canada and GIRO Inc. [Grant CRDPJ 520349-17].

Keywords: column generation • machine learning • column selection

1. Introduction

Column generation (CG) is an iterative method used to solve the linear relaxation of large-scale optimization models that involve a very large number of variables (columns) that cannot be considered at once. The method exploits the fact that the majority of these columns will not be part of an optimal solution. It starts with a subset of variables and generates new ones that have the potential to improve the current solution, that is, variables with a negative reduced cost (assuming a minimization problem), until an optimal solution is obtained and proved optimal. At each iteration, it solves a restricted master problem (RMP) and a pricing problem (PP). The RMP is the original linear program (LP) restricted to a subset of its variables and is solved by an LP solver, such as the primal simplex algorithm. The PP is application specific and aims at finding negative reduced cost columns with respect to the dual solution of the current RMP. When such columns are found, they are added to the RMP to start a new iteration. Otherwise, the CG process stops, certifying the optimality of the current RMP solution for the whole LP.

When solved by CG, several large-scale models, such as the set-partitioning model, are prone to high degeneracy, which results in slow convergence. For these problems, many columns can be generated at each iteration and added to the RMP. However, doing so just increases the difficulty of dealing with degeneracy and, thus, becomes counterproductive. In this context, we propose in this paper a new approach to accelerate CG that relies on machine learning (ML) techniques and focuses on "column selection." More precisely, at each CG iteration, a classification algorithm is used to select promising columns when a large set of columns is generated. The learning algorithm is trained on data collected from previous executions. Although the ideas presented here can be applied to various problems, the effectiveness of the approach will be demonstrated on two specific problems: the vehicle and crew scheduling problem (VCSP) and the vehicle routing problem with time windows (VRPTW).

1.1. Literature Review

CG has been used to solve many combinatorial optimization problems, a classic example being the cutting stock problem (Gilmore and Gomory 1961). The authors proposed to solve the linear relaxation of the original model by considering only a subset of the variables representing feasible cutting patterns and generating new columns as needed by solving a knapsack problem. The method has since been successfully applied to other problems, including vehicle routing problems (VRPs), crew scheduling problems (CSPs), and many other problems. To derive integer solutions, the method is embedded in a branch-and-bound framework, that is, CG is performed at every node of

the search tree, yielding a branch-and-price algorithm (see Barnhart et al. 1996; Desaulniers, Desrosiers, and Solomon 2005).

The CG algorithm is well known to have convergence issues due to high degeneracy and dual variable instability, especially on problems formulated as set partitioning problems. In the literature, several strategies have been developed to address these difficulties (Lübbecke and Desrosiers 2005, Vanderbeck 2005), such as removing redundant constraints from the problem or perturbing them by adding penalized slack and surplus variables (du Merle et al. 1999). Other stabilization techniques can be used to reduce the oscillation of the dual values from one solution to another. These techniques can, for example, force the dual variables to remain in relatively small intervals around what is called a stability center, which corresponds to the best estimate of the optimal dual values. A stabilization function (see, e.g., Amor, Desrosiers, and Frangioni 2009) is used to penalize dual solutions that are far from the stability center. The penalties and intervals can be adjusted dynamically throughout the execution of CG. Pessoa et al. (2018) propose, among others, a smoothing technique that considers a dual vector combining the current dual solution and those form previous iterations. The use of an interior point or primal-dual method (e.g., Rousseau, Gendreau, and Feillet 2007; Gondzio, González-Brevis, and Munari 2016) for solving the RMP produces central dual solutions that can also help to reduce the number of CG iterations and speed up the solution process.

Other methods more specific to problems involving set partitioning constraints and suffering from high degeneracy were explored. Elhallaoui et al. (2005) introduced a dynamic constraint aggregation method (DCA) that consists of reducing the number of set partitioning constraints in the RMP by aggregating some of them and revising this aggregation as needed to guarantee optimality. Further improvements of this method were introduced by Elhallaoui et al. (2008, 2010) with the aim of maintaining a higher level of aggregation and further reducing degeneracy. DCA has fostered the development of the improved primal simplex algorithm (IPS; Elhallaoui et al. 2011) that can tackle general linear programs without CG while reducing degeneracy. The method decomposes the problem in two subproblems: a reduced problem (RP) and a complementary problem (CP). The RP is a linear program that contains a subset of the constraints and of the variables which are called the compatible variables, that is, those whose coefficient columns can be written as linear combinations of the columns that are part of the current solution. The CP is also a linear program that is solved to prove the optimality of the RP solution for the original problem or, otherwise, to identify incompatible variables for building a new RP problem. Exploiting the IPS theoretical results, Zaghrouti, Soumis, and El Hallaoui (2014) and Zaghrouti, El Hallaoui, and Soumis (2018) developed variants of the integral simplex using decomposition algorithm (ISUD) for solving set-partitioning problems. This algorithm solves alternately a RP and a CP to find a sequence of improving integer solutions. More recently, to solve set partitioning problems, Tahir, Desaulniers, and El Hallaoui (2019) introduced a combination of ISUD and CG that is called integral column generation. At each CG iteration, ISUD solves the RMP to yield an integer solution as opposed to the standard CG that produces fractional solutions most of the times. Using a dual solution obtained by solving a linear system of equations involving the dual values of the last CP solved, the CG PP is solved to generate new columns to be handled by ISUD in the next RMP.

To the best of our knowledge, there has been no work dedicated to column selection in the CG literature. However, it has been shown many times that generating more than one column per iteration improves CG convergence. On the other hand, when many columns are generated at a given iteration, it is common to impose a maximum number of columns to add to the RMP and to select the columns to add in increasing order of their reduced cost.

In the last few years, researchers have become increasingly interested in the use of ML algorithms to solve combinatorial optimization problems, or to accelerate existing optimization methods. A good overview on ML techniques for combinatorial optimization is given by Bengio, Lodi, and Prouvost (2021). The techniques can fall into one of two categories. The first category comprises the *imitation learning* methods, where we know in advance what (not necessarily optimal) behavior is expected for the method, and we want to replace expensive calculations with a quick approximation. The second category includes methods based mainly on reinforcement learning, where current solutions are not satisfactory and we look for better ways to solve the problem.

Little work has been done so far in the CG or branch-and-price context. In Václavík et al. (2018), the authors proposed a regression model to estimate at each CG iteration an upper bound on the optimal value of the PP. This upper bound is then used to reduce the PP search space and accelerate its computational time. Other works have explored the use of ML in branch-and-bound algorithms. Khalil et al. (2016) and Alvarez, Louveaux, and Wehenkel (2017) studied variable selection for branching. Their goal is to imitate the *strong branching* strategy, a strategy that significantly reduces the number of nodes to explore, but is time consuming. Other branch-and-bound methods using learning algorithms are described in the survey of Lodi and Zarpellon (2017).

1.2. Paper Contribution and Structure

The main contribution of this paper is the design of a new CG algorithm that incorporates column selection by ML to reduce degeneracy and computational time. Column selection is applied at every CG iteration to select promising columns to add to the RMP. The selection is performed in a very short computational time by an ML model trained on data collected from previously solved instances. The algorithm is tested on two well-known problems from the literature (the VCSP and the VRPTW) and can also be applied to other problems that are solved using CG. Note that we focus on solving the initial linear relaxation only, that is, at the root node of a search tree; but the proposed column selection strategy is also applicable at other nodes.

The paper is organized as follows. In Section 2, we present a generic linear relaxation model and describe a standard CG algorithm to solve it. In Section 3, we focus on the CG/ML framework that we propose by describing in detail the different steps, from data generation to training and prediction. Sections 4 and 5 are devoted to our two case studies, namely, the VCSP and the VRPTW, respectively. The results of our computational experiments are discussed for each case. Finally, conclusions are drawn in Section 6.

2. Basic Model and Column Generation

Let us consider the following generic linear program, called the master problem (MP) in a CG context:

$$\min_{\theta} \sum_{p\in\mathcal{P}} c_p\theta_p \tag{1}$$

$$\text{s.t.} \sum_{p\in\mathcal{P}} \mathbf{a_p}\theta_p = \mathbf{b}, \tag{2}$$

$$\theta_p \geq 0, \ \forall p \in \mathcal{P}, \tag{3}$$

where $\mathcal{P}$ is the index set of the variables θ_p; $c_p \in \mathbb{R}$ and $\mathbf{a_p} \in \mathbb{R}^m$ are the cost coefficient and constraint coefficient vector of θ_p, respectively; and $\mathbf{b} \in \mathbb{R}^m$ is the right-hand-side vector of the Constraints (2). Note that some or all these constraints could also be inequalities. We assume that the number of variables θ_p is very large and the set of objects (e.g., vehicle schedules or cutting patterns) associated with these variables can be implicitly modeled as solutions of an optimization problem (e.g., a shortest path problem or a knapsack problem).

To solve this MP, CG proceeds as follows (see Figure 1). At each iteration, it solves an RMP that considers a subset $\Omega \subseteq \mathcal{P}$ of the columns. It yields a primal solution x (assuming that $\theta_p = 0, \ \forall p \in \mathcal{P} \setminus \Omega$) and a dual solution given by the dual values $\pi \in \mathbb{R}^m$ associated with Constraints (2). This dual solution is then used to find new negative reduced cost variables θ_p, by solving the following PP:

$$\bar{c} = \min_{p\in\mathcal{P}} \{c_p - \pi^T \mathbf{a_p}\}. \tag{4}$$

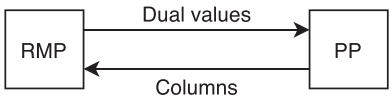
If negative reduced cost columns are found, they are added to the RMP, which is then reoptimized to obtain new dual values. Otherwise, the current RMP solution is optimal for the MP and CG stops. Note that, for some problems, the search for negative reduced cost columns can be distributed across several PPs.

Several strategies can be applied to accelerate column generation (see Desaulniers, Desrosiers, and Solomon 2002). We present here a few basic ideas that were used in our computational experiments.

– At each CG iteration, several negative reduced cost columns can be added to the RMP. Typically, this strategy reduces the total number of iterations and the overall time spent solving the PP. The possibility to generate several columns at once depends on the algorithm used to solve the PP. In our two case studies, the PP is formulated as a shortest path problem with resource constraints (Irnich and Desaulniers 2005) and solved by dynamic programming (more precisely, a labeling algorithm), which offers this possibility at no extra computational effort.

– Generating a large number of columns in a given iteration may not be effective if the columns added to the RMP are not sufficiently different from each other. To favor column diversity, we apply a simple preselection procedure that has proven to be effective for set-partitioning models (possibly with side constraints) such as the ones used for the VCSP and the VRPTW. In these models, the set-partitioning constraints enforce the covering of a task (a bus trip in the VCSP or customer service in the VRPTW) exactly once. Diversified columns should not cover the same tasks as much as possible. Two columns that do not cover the same tasks are said to be disjoint. The preselection procedure proceeds as follows. First, the columns are sorted in increasing order of their reduced cost. Second, following this order, the columns are distributed in blocks (numbered 1, 2, 3, …) of disjoint columns using the following rule: a column is put in the block numbered b if this column is not disjoint from at least one column in each block 1 to $b-1$, and disjoint from all columns already in block b. Consequently, the first column is put in block 1, the second in block 1 if it is disjoint from the first column or in block 2 otherwise, and so on. Then, only the columns in the first $n_{blks}^{\max}$ blocks of disjoint columns are kept, where $n_{blks}^{\max}$ is a predefined parameter value.

Figure 1. Standard CG Process



– Another strategy consists in removing columns from the RMP when it becomes large. The removed columns can still be generated again if necessary. In our tests, this strategy is implemented using two parameters: a minimum (n^-_{cols}) and a maximum (n^+_{cols}) number of columns to keep in the RMP. Once the maximum number is reached, the $n^+_{cols} - n^-_{cols}$ columns with the largest reduced costs are removed from the RMP.

– Heuristics can be used to speed-up the PP. For our case, we try to generate negative reduced cost columns using a heuristic labeling algorithm that does not consider all resource constraints in the dominance rule, nor all arcs in the PP network. When no negative reduced cost column can be found heuristically, an exact algorithm is invoked to ensure the exactness of the algorithm.

3. Methodology

At each CG iteration, after reoptimizing the RMP and solving the PP, a set $\mathcal{G}$ of columns is obtained. The goal is to select a subset of columns $\mathcal{G}^S \subseteq \mathcal{G}$ to be added to the RMP. This selection of columns can be considered as a binary classification problem (supervised learning problem), where we try to classify each generated column into one of two classes $y \in \{0,1\}$, that is, one if the column is to be selected and zero otherwise. The next sections describe the different steps required to build the data set, as well as details on the algorithm that will be used for the training.

3.1. Data Collection

As with all supervised learning problems, a data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ is required, where n is the size of the data set, $\boldsymbol{x}_i$ is the vector representing the features of column i, and $y_i \in \{0,1\}$ the column label (our target/output). Before talking about the features to be extracted from the generated columns, we start by describing how the labels are assigned, which is equivalent to answering the following question: which columns are considered as "promising" and should be selected?

3.1.1. Labels. The idea is to add a small subset of the generated columns to the RMP so that it remains fast to reoptimize. On the other hand, we also want a maximum reduction of the objective value at each CG iteration. For these reasons, a good option is to add only the generated columns that will take a positive value after the RMP reoptimization, that is, we are looking for a set of columns that will ensure a maximal decrease of the objective value and that is of minimal size. This column selection problem is, thus, a hierarchical biobjective problem. Using a weighted sum of the objective functions, it can be formulated as the following mixed integer linear program (MILP), which is a copy of the RMP with additional constraints and integer variables.

Let ℓ be the CG iteration number, Ω_ℓ the set of columns present in the RMP at the start of iteration ℓ, and $\mathcal{G}_\ell$ the generated columns at this iteration. For each column $p \in \mathcal{G}_\ell$, we define a decision variable y_p that takes value one if column p is selected and zero otherwise. To minimize the number of selected columns, a sufficiently small penalty ϵ is incurred for each selected column.

The proposed MILP is as follows:

$$\min_{\theta, y} \sum_{p \in \Omega_\ell \cup \mathcal{G}_\ell} c_p \theta_p + \sum_{p \in \mathcal{G}_\ell} \epsilon y_p, \tag{5}$$

$$\text{s.t.} \sum_{p \in \Omega_\ell \cup \mathcal{G}_\ell} \mathbf{a_p} \theta_p = \mathbf{b}, \tag{6}$$

$$\theta_p \geq 0, \ \forall p \in \Omega_\ell \cup \mathcal{G}_\ell \tag{7}$$

$$\theta_p \leq y_p, \ \forall p \in \mathcal{G}_\ell, \tag{8}$$

$$y_p \in \{0,1\}, \ \forall p \in \mathcal{G}_\ell. \tag{9}$$

Without the y_p variables and Constraints (8) and (9), model (5)–(9) would exactly be the RMP at iteration $\ell + 1$. Assuming that ϵ is sufficiently small, these variables and constraints are only considered to minimize the set of selected columns. These columns correspond to those for which $y_p = 1$, that is, the columns y_p associated with positive-valued θ_p variables (due to Constraints (8)). The subset of columns to be added in the RMP at iteration ℓ is therefore

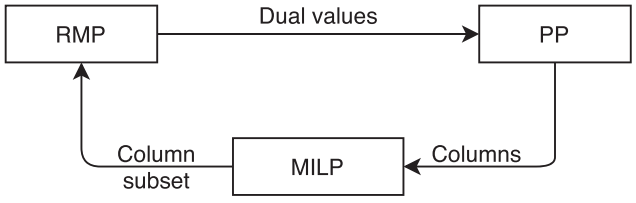
$$\mathcal{G}^S_\ell = \{p \in \mathcal{G}_\ell : y_p = 1\}. \tag{10}$$

The overall procedure is illustrated in Figure 2.

In other words, we are using the above MILP as the "expert" that we want to imitate and learn from.

3.1.2. Column Features. In CG, each column represents something different depending on the PP generating the columns. For example, if the PP is a shortest path problem with resource constraints and each column is associated with a path representing, for example, a route or a schedule, features such as resource values, number of nodes in the path, or distances between nodes can be considered. In addition, general features that can be used for any problem are also considered, such as the cost, reduced cost, and dual values of the constraints that the column contributes to.

Figure 2. CG Algorithm with MILP Column Selection



A complete list of the features is given in Sections 4 and 5 for our two case studies.

3.2. Algorithmic Requirements

In ML, many classification algorithms exist. In our case, the choice of the appropriate algorithm should take into consideration the following points:

– The selection of a column depends on the presence of other columns in the problem, that is, $\Omega_\ell \cup \mathcal{G}_\ell$. It is preferable that each column has information about the other columns.

– Based on the previous point, if we consider all the columns at once (taking $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ all at once as input to the ML model), it should be noted that the number of columns n will be different from one iteration to another and that the result of the prediction should be the same regardless of the columns order (*order invariance*).

– Dual values contain rich information that should be used to determine whether a column should be selected. However, the number of dual values depends on the number of constraints in the problem and the number of constraints to which a column contributes is different from one column to another.

– The ML model should be able to generalize to new instances of a given problem, keeping in mind that the instances will not be necessarily of the same size.

– The purpose of the ML model is to replace the MILP presented above, which is computationally expensive. A fast prediction is desirable.

3.3. Graph Neural Networks for Column Classification

Graph neural networks (GNNs) form an effective framework for learning with graph structured data. They can be used for tasks like node classification, link prediction, graph prediction, document classification, and even combinatorial optimization. They were initially introduced by Gori, Monfardini, and Scarselli (2005) and further developed by Scarselli et al. (2009). A GNN aims at learning a node representation by aggregating information from the neighbor nodes in an iterative manner. This is also known as a message-passing method. The node representations can then be used to produce an output label for node classification tasks.

Because of the effectiveness of these methods in capturing dependencies between nodes, various studies tried to extend the learning approaches on graphs by adopting ideas from deep learning paradigms and gave birth to several methods based on convolutional neural networks, known as graph convolutional networks (GCN), where the convolution operation has been redefined for graph structured data. GCN methods are divided into two groups: the first group contains the spectral-based methods, which are based on spectral graph theory and graph signal processing, and the second one includes the spatial-based methods that aggregate information directly from the neighbor nodes. These approaches are summarized in a recent survey by Wu et al. (2019).

3.3.1. GNN Overview. Given a graph $G = (V, E)$, where V is the set of nodes and E the set of edges, each node $v \in V$ is characterized by its feature vector $\boldsymbol{x}_v$. The goal is to iteratively update the representation of a node (also called *state*) by aggregating information from the neighbor nodes. Let $\boldsymbol{h}_v^{(k)}$ be the representation vector of node $v \in V$ at iteration $k = 0, 1, \ldots, K$. Let $\mathcal{N}(v)$ be the set of neighbor (adjacent) nodes of $v \in V$. As shown in Figure 3, the representation vectors are iteratively updated by aggregating the representations of the neighbor nodes as follows. At an iteration $k > 0$, an aggregated function, denoted *aggr*, is first applied to compute an aggregated information vector $\boldsymbol{a}_v^{(k)}$ for each node $v \in V$:

$$\boldsymbol{a}_v^{(k)} = aggr\left(\left\{\phi^{(k)}\left(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{h}_v^{(k-1)}\right) : u \in \mathcal{N}(v)\right\}\right), \qquad \forall v \in V,$$

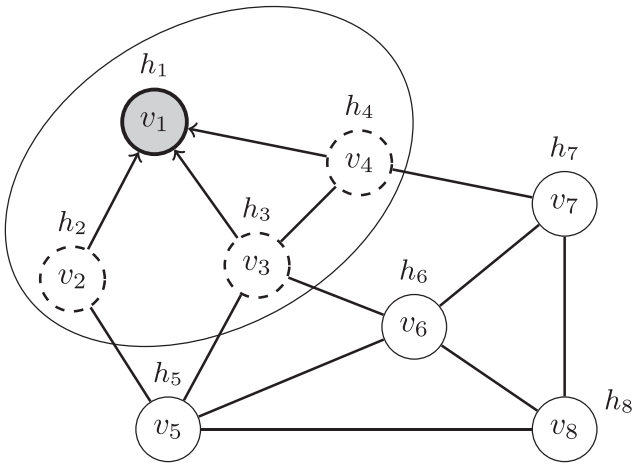
with $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v$ and $\phi^{(k)}$ is a learned function. Function *aggr* should be invariant to permutations of the node order, such as the sum, mean, and min/max functions. Then, using a function denoted *comb* (which is often a concatenation function), we combine the aggregated information with the current state of the given nodes to obtain the updated node representation vectors:

$$\boldsymbol{h}_v^{(k)} = \psi^{(k)}\left(comb\left(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}\right)\right), \qquad \forall v \in V,$$

where $\psi^{(k)}$ is another learned function.

As the iterations progress, the nodes collect information from farther away neighbor nodes. At the final iteration K, the representation $\boldsymbol{h}_v^{(K)}$ of a node $v \in V$ can

Figure 3. Representation Vector h_1 of Node v_1 Is Updated by Aggregating the Representations h_2, h_3, h_4 of Its Neighbor Nodes



be used to predict its label, denoted l_v, by applying a final learned transformation, denoted *out*, that is,

$$l_v = out\left(\boldsymbol{h}_v^{(K)}\right), \qquad \forall v \in V.$$

The learned functions $\phi^{(k)}$, $\psi_k^{(k)}$, and *out* are implemented by using multilayer perceptron feed forward neural networks (Goodfellow, Bengio, and Courville 2016).

A variety of architectures can be implemented within a GNN. For example, we can have edge features, a directed graph instead of an undirected one, an edge or graph classification problem, etc. A good summary of the architectures that have been proposed in the literature can be found in Battaglia et al. (2018), where a configurable framework for building complex architectures is also proposed.

3.3.2. Our Model. The model presented in the previous section can be used for our column classification task. One obvious architecture consists in representing each column by a node and linking two nodes with an edge if their corresponding columns contribute to at least one constraint in common. However, in this case, the number of edges would be very large and the dual value information would be difficult to include in the model. A different architecture uses a bipartite graph with two node types: column nodes V and constraint nodes C. An edge (v,c) exists between a node $v \in V$ and a node $c \in C$ if column v contributes to constraint c (see Figure 4(a)). The advantage of this representation is that we can have feature vectors attached to the constraints nodes, which will allow us to include dual information easily.

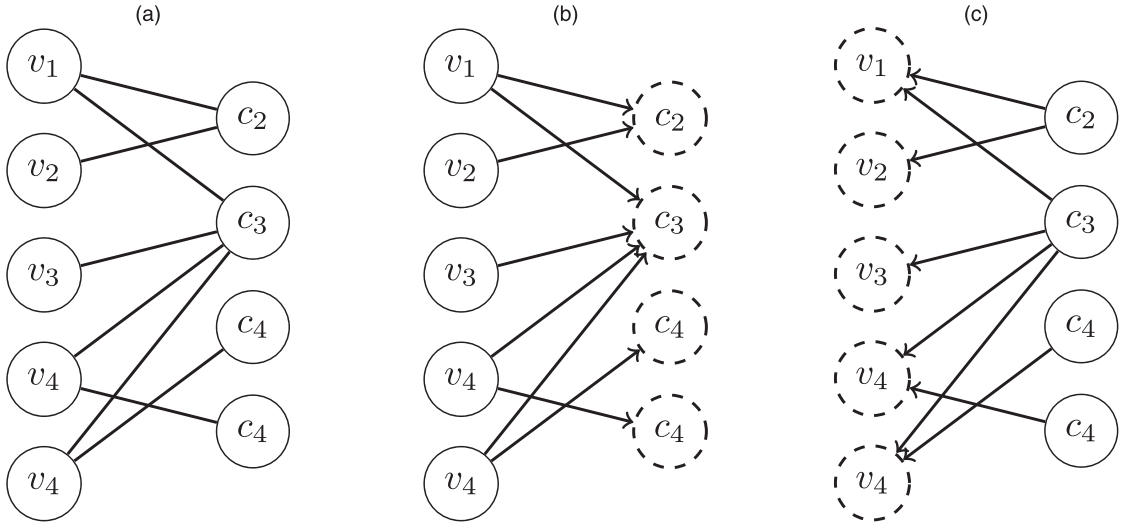
Because there are two node types, each iteration proceeds in two phases: a first phase is performed to update the constraint representations (Figure 4(b)), followed by a second phase that updates the column representations (Figure 4(c)). Note that, after a single iteration, every column node contains some information about the other columns that contribute to the same constraints. However, better information may be transmitted by performing additional iterations. The column representations $\boldsymbol{h}_v^{(K)}$, $v \in V$, are then used for predicting labels $y_v \in \{0,1\}$. The steps are described in Algorithm 1.

Algorithm 1 (GNN Applied to the Bipartite Graph Model)

1: $\boldsymbol{h}_v^{(0)} = \boldsymbol{x}_v, \quad \forall v \in V$
2: $\boldsymbol{h}_c^{(0)} = \boldsymbol{x}_c, \quad \forall c \in C$
3: for $k = 1, \ldots, K$ do
4: $\quad \boldsymbol{a}_c^{(k)} = \sum_{u \in \mathcal{N}(c)} \phi_C^{(k)}(\boldsymbol{h}_c^{(k-1)}, \boldsymbol{h}_u^{(k-1)}), \quad \forall c \in C$
5: $\quad \boldsymbol{h}_c^{(k)} = \psi_C^{(k)}([\boldsymbol{h}_c^{(k-1)}, \boldsymbol{a}_c^{(k)}]), \quad \forall c \in C$ } Phase 1
6: $\quad \boldsymbol{a}_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \phi_V^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{h}_u^{(k-1)}), \quad \forall v \in V$
7: $\quad \boldsymbol{h}_v^{(k)} = \psi_V^{(k)}([\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}]), \quad \forall v \in V$ } Phase 2
8: end for
9: $y_v = out(\boldsymbol{h}_v^{(K)}), \quad \forall v \in V$

As described in the previous section, we start by initializing the representation vectors of both the column and constraint nodes by the feature vectors $\boldsymbol{x}_v$ and $\boldsymbol{x}_c$, respectively (steps 1 and 2). For each iteration k, we perform the two phases: updating the constraint representations (steps 4 and 5), then the column ones (steps 6 and 7). The sum function is used for the *aggr* function and the vector concatenation for the *comb*

Figure 4. Our Model Represented by a Bipartite Graph for Column Classification/Selection



Note. (a) Bipartite graph with two node types V and C; (b) Phase 1: constraint representations update; (c) Phase 2: column representation update.

function. The functions $\phi_C^{(k)}$, $\psi_C^{(k)}$, $\phi_V^{(k)}$, and $\psi_V^{(k)}$ are two-layer feed forward neural networks with rectified linear unit (ReLU) activation functions and *out* is a three-layer feed forward neural network with a sigmoid function for producing the final probabilities (step 9). A weighted binary cross entropy loss is used to evaluate the performance of the model, where the weights are used to deal with the imbalance between the two classes. Indeed, about 90% of the columns belong to the unselected class, that is, their label $y_v = 0$.

Column set V includes the newly generated columns (i.e., those in $\mathcal{G}_\ell$) and also the columns in the current basis of the RMP. The latter columns can give an indication of which columns to select, and they have proven to be valuable. Taking all the columns of the RMP (i.e., the nonbasic columns as well) can result in a slightly better accuracy of the model. However, the graphs would be much larger each time that columns are added to the RMP, which would require more computational effort. This would slow down the training and the prediction, and also increase memory usage.

The data are collected by solving multiple problem instances using the MILP (10)–(15) to select the columns and assign the labels. At each CG iteration, a bipartite graph is built and the following data are stored:

- The sets of column and constraint nodes;
- A sparse matrix $E \in \mathbb{R}^{n \times m}$ storing the edges;
- A column feature matrix $X^V \in \mathbb{R}^{n \times d}$, where n is the number of columns and d the number of column features;
- A constraint feature matrix $X^C \in \mathbb{R}^{m \times p}$, where m is the number of constraints and p the number of constraint features;
- The label vector $\boldsymbol{y}$ of the newly generated columns in $\mathcal{G}_\ell$.

Note that, in a CG algorithm, the number of generated columns n is not constant throughout the iterations. Furthermore, if CG is used to solve various instances of different sizes, the number of constraints m may vary from one instance to another. Nonetheless, the same GNN model can handle graphs with different numbers of nodes and arcs, as long as the numbers of column and constraint features (d and p) are identical.

4. Case Study I: Vehicle and Crew Scheduling Problem

In this first case study, we focus on a problem arising in urban transit systems called the VCSP. Traditionally, this problem is split into two problems that are solved in a sequential fashion, where the vehicle schedules are first determined (vehicle scheduling problem) before the crew schedules (crew scheduling problem). This approach has been strongly criticized (Ball, Bodin, and Dial 1983) because in most cases driver costs dominate the vehicle costs. In our case, we consider the VCSP, which is a complete integration of the CSP and the VSP. The objective is to determine vehicle schedules and driver schedules simultaneously while minimizing the total cost. The first formulation that integrates both problems is that of Freling, Wagelmans, and Paixão (1999), where two approaches for solving the problem, the first using CG and the second using Lagrangian relaxation, are proposed.

In our work, we consider the model of Haase, Desaulniers, and Desrosiers (2001), which is a set partitioning model with additional constraints involving only the crew scheduling variables. The additional constraints ensure that vehicle schedules are obtained in polynomial time. Moreover, taking into account the vehicle costs in the objective function ensures that an optimal solution is obtained. The authors propose a branch-and-price method to solve the proposed formulation. Other formulations based on CG and set partitioning models were then proposed by Freling, Huisman, and Wagelmans (2003). An extension for the case of multiple depots (MD-VCSP) using Lagrangian relaxation in combination with CG is proposed by the same authors in Huisman, Freling, and Wagelmans (2005). A similar formulation but with less decision variables and fewer constraints is proposed by Mesquita and Paias (2008). More work was done in this direction by de Groot and Huisman (2008), where different splitting strategies for tackling large real-world MD-VCSP instances are devised.

4.1. Problem Description and Basic Solution Approach

We focus on the case of a bus company with a single depot and a homogeneous fleet of vehicles as described in Haase, Desaulniers, and Desrosiers (2001). Each bus line is defined by a departure point, a destination point, several intermediate stops where passengers can get on and off the bus. Some of these stops are called relief points where driver exchanges can occur. For each line, there is a predefined timetable defining a set of trips to be covered during the day. The objective is to assign the buses to the different trips, as well as assigning the drivers to the buses, while minimizing total costs, which include driver and bus operational costs, and may also include fixed costs for vehicles and drivers.

As mentioned above, our basic solution approach is that of Haase, Desaulniers, and Desrosiers (2001) who developed a branch-and-price algorithm, where the master problem is a set partitioning model with additional constraints, involving driver schedule variables

only. The PP is a shortest path problem with resource constraints solved with a labeling algorithm (Irnich and Desaulniers 2005) that allows the generation of driver schedules, also called duties. Note that the number of columns generated by this algorithm can be adjusted by modifying the dominance rule or by solving the same PP multiple times after removing some nodes/arcs from the network after each resolution. The detailed network structure and formulation are described in Haase, Desaulniers, and Desrosiers (2001).

4.2. Features Considered

As mentioned in Section 3.1.2, the features collected depend on the problem at hand. For the VCSP case, each column represents a path with resource constraints and the following features are used:

– *Columns:* cost, reduced cost, total number of constraints that the column contributes to, number of constraints per constraint group that the column contributes to (the set partitioning model of Haase, Desaulniers, and Desrosiers (2001) has four groups of constraints), resource values (i.e., work time, duty duration), duty type (there are two possible duty types), columnIsNew (Boolean value indicating if the column is newly generated or in the basis), column value (if the column is in the basis, zero otherwise), column incompatibility degree (as defined in Elhallaoui et al. 2010);

– *Constraints:* dual value, number of columns (newly generated or in the RMP basis) contributing to the constraint (equal to the degree of the node in the bipartite graph).

4.3. VCSP Instances

Instances of the VCSP were randomly generated as in Haase, Desaulniers, and Desrosiers (2001). The trips to cover during a day are uniformly distributed over the time horizon, with more trips during peak hours. A total of 100 instances of 400 trips were generated for the training phase. These instances were solved using the MILP for the column selection at each iteration, and the labels were assigned accordingly. The data of the 100 instances were collected resulting in more than 7,000 iteration data sets (i.e., 7,000 bipartite graphs). Once the training was completed, new instances of different sizes varying between 300 and 800 trips were solved using the learned model for selecting the columns. The experimental results of both ML and optimization phases are reported in the next section.

4.4. Computational Results

We start our computational experiments by evaluating the performance of the MILP proposed in Section 3.1.1 because we are trying to imitate it. Once the performance of the MILP is confirmed, we can proceed to the evaluation of the training phase and, finally, to the integration of the learned model in the CG algorithm.

4.4.1. MILP Performance. Column selection by the MILP was performed on several VCSP test instances. For a typical instance, Figure 5 compares the normal execution, where all generated columns are added (curve with triangles), and the execution with the MILP selection (curve with squares). By adding all the columns, the normal execution reaches the optimal solution in 56 iterations for this instance, unlike the execution with the MILP selection, which exhibits a major convergence issue. This is mainly due to the rejected columns that remain with a negative reduced cost after the RMP reoptimization and keep on being generated in subsequent iterations, even though they do not improve the objective value (degeneracy). Somehow, the columns selected by the MILP are not sufficient and do not allow the generation of significantly better columns in the subsequent iterations. A possible solution would be to add more columns, and the best candidates are the columns that remain with a negative reduced cost after the reoptimization of the RMP (see Figure 6). Given the large number of those columns, it is not necessary to add them all. One option consists of sorting them by increasing reduced cost and adding a predetermined percentage of them.

For five VCSP instances with 400 trips, Table 1 shows the results obtained by the CG algorithm without column selection and the one with column selection by the MILP and additional columns. For each instance, we report the total computational time (in seconds) required to solve the linear relaxation, the time taken to solve the RMPs only (in seconds), the number of CG iterations, the number of columns in the RMP at the end of the optimization, and the percentage of the computational time saved by using column selection. Preliminary tests were run to

Figure 5. Comparison Between the Execution Without Selection and with MILP Selection

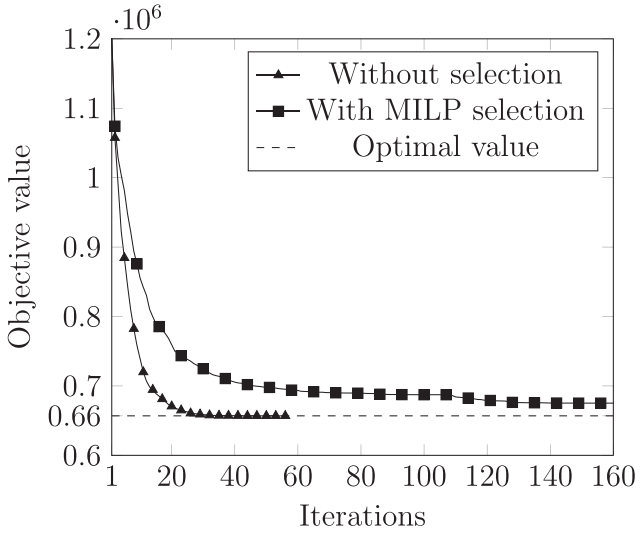
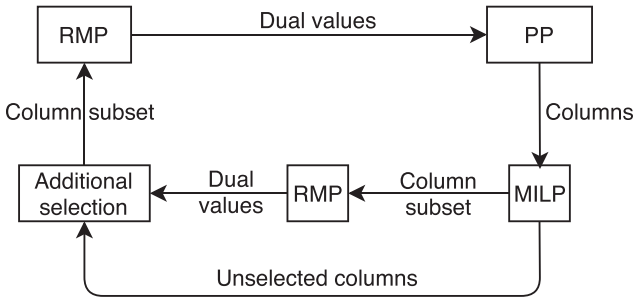


Figure 6. CG Algorithm with MILP Column Selection and Additional Columns



configure both algorithms. In particular, the CG algorithm without column selection turns out to be better when the number of columns generated is approximately three times less than the number of columns generated by the algorithm with column selection. For the latter algorithm, in addition to the columns selected by the MILP, we add 50% of the unselected columns that have a negative reduced cost after the RMP reoptimization. Note that, for the results presented in this section, no columns are removed from the RMP (i.e., $n^{+}_{cols} = \infty$) and the computational time of the algorithm with column selection does not include the time spent solving the MILP at every iteration because we are interested only in assessing the effectiveness of the selection. The MILP will be replaced afterward with a fast learned model.

From these results, we observe that column selection allows a reduction of the average computational time of 34%, showing the potential of column selection. Given that the average number of iterations is quite similar for the two algorithms (except for the first instance, which highly suffered from degeneracy without column selection), we can deduce that this gain is due to a reduction in the time spent solving the RMPs. Indeed, the saving on the average RMP time is around 38%, which results from a smaller average number of columns to handle (around 20% less columns).

4.4.2. Machine Learning Phase. As always in ML, several hyperparameters have to be adjusted. The best model that we obtained uses the values presented in Table 2. The accuracy can be slightly improved in favor of a longer prediction or training time, which is not worth it based on the experiments we conducted.

The data set is split into two sets: training and test sets (75% for training and 25% for test). The training is performed in 1,000 epochs, where each epoch includes 32 batches and each batch includes data from 16 iterations. The number of iterations corresponds to the number of updates of the constraint and column nodes in the graph (parameter K in Algorithm 1). This parameter is set to one as a higher value of K resulted in a slight improvement of 1%; but the training and the prediction took at least twice as much time, which is not desirable. The intermediate NN architecture corresponds to a two-layer neural network with 32 neurons on each layer and ReLU activations. This architecture implements the functions $\phi_C^{(k)}$, $\psi_C^{(k)}$, $\phi_V^{(k)}$, and $\psi_V^{(k)}$ in Algorithm 1, whereas the output NN architecture implements the output function *out*. To deal with the imbalanced data, a weighted sigmoid function is used, assigning a weight of 10 and 1 to the columns with label $y_v = 1$ and $y_v = 0$, respectively (i.e., misclassifying one column with label $y_v = 1$ is equivalent to misclassifying 10 columns with label $y_v = 0$). The weights allow also to give more importance to the columns to be selected.

To evaluate the performance of the ML model obtained, we consider different metrics that are listed in Table 3 and computed as follows. Let tp, tn, fp, and fn be the number of true positives (columns with label $y_v = 1$ that are predicted correctly), true negatives (columns with label $y_v = 0$ that are predicted correctly), false positives (selected columns that should have been rejected), and false negatives (rejected columns that should have been selected), respectively. Then,

$$\text{Recall} = \frac{tp}{tp + fn}$$

$$TNR = \frac{tn}{tn + fp}$$

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Balanced Accuracy} = \frac{TPR + TNR}{2},$$

Table 1. Results with the MILP Selection

		Without selection				With selection				
Instance	# Constraints	Time	Time RMP	# Itr	# Col	Time	Time RMP	# Itr	# Col	Time reduction (%)
VCS_400_1	1,740	310	244	104	20,195	174	132	44	15,624	44
VCS_400_2	1,757	311	262	46	17,942	178	140	41	14,754	53
VCS_400_3	1,752	464	405	53	20,140	298	243	57	17,101	36
VCS_400_4	1,757	363	308	50	19,453	284	233	51	16,683	22
VCS_400_5	1,759	319	270	46	18,162	218	175	43	14,352	32
Average	1,753	353	297	59	19,178	230	184	47	15,703	34

Note. Itr, iterations; Col, columns.

Table 2. Parameters Used for Training the VCSP GNN Model

Parameters	Value
Number of iterations	1
Learning rate	10^{-3}
Maximum number of epochs	1,000
Epoch size	32
Batch size	16
Intermediate NN architecture	32×32
Output NN architecture	$32 \times 32 \times 1$
Activation function	ReLU
Optimizer	Adam
Sigmoid class weights	10:1

where Recall is also called *TPR* for true positive rate and *TNR* stands for true negative rate.

The results reported in Table 3 were obtained on a test set consisting of 400-trip instances. From these results, one can see that the recall value is high, indicating that the columns to select are predicted correctly with an 86.2% accuracy. However the *TNR* value is only 66.6%, which means that among all the columns that should not be selected, only 66.6% were predicted correctly and the remaining ones are added when they should not. In our case, we are more interested in selecting accurately the columns with label $y_v = 1$ because selecting additional columns is not an issue and may be desirable to avoid convergence issues as encountered with the initial MILP selection. Because of the label imbalance, the 33.4% columns that are misclassified and added by the model largely exceed in numbers the 86.2% correctly predicted ones, which is reflected by the precision value of 23.7%. In other words, out of all the columns selected by the model, only 23.7% should actually be selected and the remaining columns are supplementary. Therefore, no additional columns are required when using this model in the CG algorithm, unlike the MILP where additional columns were needed. Next, note that the use of the normal accuracy $(accuracy = (tp + tn)/(tp + tn + fp + fn)$ can be ambiguous when dealing with imbalanced data. Consequently, it is more interesting to consider the balanced accuracy that normalizes both classes. A balanced accuracy of 50% is obtained when all the columns are given the same label 0 or 1, so our 76.5% balanced accuracy is a satisfactory result. Finally, to conclude this section, we highlight that

Table 3. Metrics of the Best GNN Model Obtained for the VCSP

Metric	Value
Recall	86.2%
TNR	66.6%
Precision	23.7%
Balanced accuracy	76.5%

Note. TNR, true negative rate.

the training phase can require a few hours of computing time or even more depending on the performance of the computer used and the usage of GPUs or not.

4.4.3. Optimization Phase. Once the training is done, the learned model is integrated into the CG algorithm for selecting columns at each iteration (it replaces the MILP in Figure 2). To test the efficiency of the learned model, we compare the following five algorithms that all use the acceleration strategies described in Section 2: (i) disjoint blocks to diversify the columns to be added or selected, (ii) column removal when the RMP becomes too large, and (iii) heuristic labeling for solving the PP before calling the exact labeling algorithm if needed.

No selection (NO-S): This is the standard CG algorithm with no selection involved, with the use of the acceleration strategies described in Section 2.

MILP selection (MILP-S): The MILP is used to select the columns at each iteration, with 50% additional columns to avoid convergence issues. Because the MILP is considered to be the expert we want to learn from and we are looking to replace it with a fast approximation, the total computational time does not include the time spent solving the MILP.

GNN selection (GNN-S): The learned model is used to select the columns. At every CG iteration, the column features are extracted, the predictions are obtained, and the selected columns are added to the RMP.

Sorting selection (Sort-S): The generated columns are sorted by reduced cost in ascending order, and a subset of the columns with the lowest reduced cost are selected. The number of columns selected is on average the same as with the GNN selection.

Random selection (Rand-S): A subset of the columns is selected randomly. The number of columns selected is on average the same as with the GNN selection.

Although the reduced cost of a column is not a reliable indicator of whether a column is interesting or not, comparing the model with the last two selection strategies will help us to determine if the model was really able to identify some hidden patterns and to learn effectively from the data.

The parameter values used by the five algorithms are summarized in Table 4. As defined in Section 2, the parameters n^-_{cols} and n^+_{cols} correspond to the minimum and maximum number of columns to keep in the RMP, respectively, where n_{cons} is the number of constraints in the RMP. To ensure column diversity, a number $n^{\max}_{blks}$ of disjoint blocks is used. For algorithms including a selection strategy, a higher value is used to select from a larger pool of columns. For the MILP, a low penalty $\epsilon = 0.1$ is used for every selected column because the selection with the MILP alone suffers from convergence issues and a higher penalty implies less selected columns. In addition to the low penalty,

Table 4. Parameter Values Used by the Five Algorithms

Algorithm	n^-_{cols}	n^+_{cols}	$n^{\max}_{blks}$	min_{sel}	ϵ (penalty)	Additional columns
NO-S	$3 \times n_{cons}$	$6 \times n_{cons}$	4	100	–	–
MILP-S			14		0.1	50%
GNN-S					–	0%
Sort-S						
Rand-S						

50% of the remaining columns with the most negative reduced cost (after the RMP reoptimization) are added to the RMP. The parameter min_{sel} is equal to the minimum number of columns that need to be generated to activate column selection at a CG iteration. In fact, in preliminary tests, we observed that it is not worthwhile selecting columns from a small set of generated columns. As shown in the ML results, the GNN model selects more columns than the MILP (recall of 23.7%); therefore, no additional columns are needed. Note that some parameters were tuned by a group of algorithms to obtain the best results for each algorithm and to provide a fair comparison between them.

The results obtained for 15 VCSP instances involving 300–500 trips are reported in Table 5. For each algorithm, the total time in seconds, the time spent solving the RMP and the PPs, as well as the number of CG iterations are reported. The time reduction column compares the GNN-S to the NO-S algorithm. First, let us mention that the time taken by the GNN model to make the prediction in GNN-S is negligible: it varies between 40 *ms* and 70 *ms* per iteration, depending on the size of the instance and the number of generated columns. Therefore, given the reduced number of iterations performed, the total prediction time sums up to a few seconds only. By excluding the time taken to solve the MILPs in the algorithm MILP-S, we are assuming that the MILP requires a negligible time just like the GNN prediction time, which allows us to fairly compare the two selection techniques and to check how close GNN is to the MILP selection. Obviously, solving the MILP is too time consuming to be used in practice. In fact, its solution time is larger than the time required to solve the RMP.

The computational times reported for both GNN-S and MILP-S (the latter excluding the MILP solution time) are very close to each other for most instances, showing that the GNN does a good job at imitating the MILP selection. Still, there is a difference between the two, namely, the GNN model tends to select more columns than the MILP. Therefore, slightly different results are obtained, sometimes better and sometimes worse but in average very close. The times of the last two algorithms based on the other two selection strategies (Sort-S and Rand-S) are either close or worse than those obtained by NO-S; they are far from the results given by GNN-S or MILP-S. Their relatively bad performance is also reflected by the larger number of iterations performed. The selection with the GNN model gives good results on all the instances even if

Table 5. Results for VCSP Instances with 300–500 Trips

	NO-S				MILP-S				GNN-S				Sort-S				Rand-S				
	Time (s)				Time (s)				Time (s)				Time (s)				Time (s)				Time
Instance	Total	RMP	PP	# Itr	Total	RMP	PP	# Itr	Total	RMP	PP	# Itr	Total	RMP	PP	# Itr	Total	RMP	PP	# Itr	reduction (%)
VCS_300_1	105	66	39	63	73	41	32	62	69	41	28	54	93	52	41	74	110	63	47	81	34
VCS_300_2	72	48	24	41	49	31	18	40	49	33	16	37	71	44	27	55	73	46	27	52	32
VCS_300_3	131	80	51	83	115	59	56	94	95	57	38	76	156	78	78	151	150	83	67	118	27
VCS_300_4	158	104	54	73	107	55	52	76	110	57	53	75	151	84	67	118	175	97	78	123	30
VCS_300_5	76	47	29	58	54	31	23	49	55	34	21	45	79	48	31	61	86	54	32	60	28
Average	108	69	39	63	79	43	36	64	75	44	31	57	110	61	49	91	118	68	50	86	30
VCS_400_1	265	178	87	74	170	113	57	56	180	96	84	75	287	174	113	108	270	170	100	95	32
VCS_400_2	259	184	75	72	187	129	58	58	158	106	52	55	305	192	113	112	227	154	73	71	39
VCS_400_3	365	260	105	93	277	191	86	89	278	171	107	96	435	256	179	186	427	274	153	140	24
VCS_400_4	315	222	93	77	254	173	81	73	221	138	83	70	391	246	145	137	370	243	127	109	30
VCS_400_5	234	168	66	59	199	137	62	60	207	138	69	61	319	207	112	110	277	188	89	79	12
Average	287	202	85	75	217	148	68	67	208	129	79	71	347	215	132	130	314	205	108	98	25
VCS_500_1	721	551	170	79	449	281	168	75	464	295	169	78	781	508	273	150	781	561	220	108	36
VCS_500_2	769	596	173	79	490	321	169	79	455	298	157	72	766	522	244	144	779	574	205	108	41
VCS_500_3	910	679	231	113	510	332	178	88	490	326	164	83	1,105	707	398	229	881	622	259	131	46
VCS_500_4	572	416	156	77	485	327	158	87	496	328	168	90	616	430	186	102	583	390	193	105	13
VCS_500_5	818	641	177	92	645	474	171	94	703	512	191	97	794	609	185	107	926	660	266	150	14
Average	758	576	181	88	515	347	168	84	521	351	170	84	812	555	257	146	790	561	228	120	26

Note. Itr, iterations.

the training has only been done on instances with 400 trips, which shows the ability of the model to generalize to instances of different size. Additionally, the total number of columns added to the RMP before reaching the optimal solution is generally lower than the case without selection. Compared with NO-S, GNN-S gives on average a reduction ranging from 25%–30% as shown in the last column where most of the reductions are obtained on the RMP side.

Additional experiments have been conducted on larger instances with 600–800 trips, comparing only the first three algorithms. The detailed results are shown in Table 6. They indicate that column selection using the GNN model is still practical and gives good results, namely, average time reductions of 21%–23%. Note that, for these tests, we used the same prediction model as for the smaller instances, which was trained on instances with 400 trips.

As mentioned in Section 1.1, the CG algorithm is well known to have convergence issues because of the degeneracy and dual variable instability. To overcome these difficulties, various stabilization techniques have been devised and proven to be very effective on different problems (Pessoa et al. 2018). To test whether the improvements obtained by our column selection strategy are preserved for a stabilized CG algorithm, we implemented a variant of the stabilized algorithm of Pessoa et al. (2018) that is based on the Wentges rule,

$$\tilde{\pi}^t = \alpha\hat{\pi} + (1-\alpha)\pi^t, \tag{11}$$

where α is a parameter in $(0,1]$ (set to 0.6 for our tests); $\tilde{\pi}^t$ is the vector of dual variables used by the PPs at iteration t; π^t is the dual solution returned by the RMP at this iteration; and $\hat{\pi}$ is the dual solution that allowed to find, in the last five iterations, the largest least reduced cost obtained in an iteration. This algorithm differs from that of Pessoa et al. (2018) only by how $\hat{\pi}$ is computed. In their algorithm, $\hat{\pi}$ corresponds to the dual solution yielding, over all previous iterations, the largest least reduced cost in an iteration, that is, the best Lagrangian bound found so far. Because columns are mostly generated by heuristic labeling, we do not have access to a Lagrangian bound in most iterations, making it difficult to apply the exact same technique.

We integrated this stabilization procedure to the NO-S and GNN-S algorithms, to yield two stabilized algorithms denoted Stab-NO-S and Stab-GNN-S. The results obtained on the same 300- to 500-trip VCSP instances as above are reported in Table 7, where, for both algorithms, we indicate the same information as in the previous tables. The results show that stabilization can improve the performance of the nonstabilized algorithms by approximately 37% for NO-S and 35% for GNN-S. Therefore, integrating column selection into a stabilized CG algorithm still yields average computational time reductions ranging between 24% and 27%, whereas these reductions range between 25% and 30% without stabilization (see Table 5). As above, most of the reductions come from the RMP side.

5. Case Study II: Vehicle Routing Problem with Time Windows

The vehicle route problem has been the subject of intensive research for over 50 years because of its

Table 6. Results for VCSP Instances with 600–800 Trips

	NO-S				MILP-S				GNN-S				
	Time (s)				Time (s)				Time (s)				
Instance	Total	RMP	PP	# Itr	Total	RMP	PP	# Itr	Total	RMP	PP	# Itr	Time reduction (%)
VCS_600_1	1,319	1,012	307	108	1,049	751	298	100	998	713	285	96	24
VCS_600_2	1,445	1,098	347	112	1,005	699	306	92	1,031	709	322	99	29
VCS_600_3	983	740	243	89	659	466	193	75	805	555	250	93	18
VCS_600_4	1,930	1,434	496	159	1,659	1,172	487	156	1,570	1,054	516	165	19
VCS_600_5	1,379	977	402	137	1,028	708	320	108	1,025	674	351	115	26
Average	1,411	1,052	359	121	1,080	759	320	106	1,085	741	344	115	23
VCS_700_1	2,396	1,944	452	101	1,977	1,513	464	107	1,882	1,444	438	97	21
VCS_700_2	1,613	1,285	328	80	1,118	836	282	73	1,212	901	311	78	25
VCS_700_3	2,074	1,665	409	106	1,470	1,090	380	89	1,546	1,151	395	97	25
VCS_700_4	2,465	1,982	483	102	1,854	1,377	477	104	2,061	1,563	498	109	16
VCS_700_5	2,101	1,730	371	85	1,487	1,118	369	87	1,672	1,278	394	101	20
Average	2,129	1,721	408	94	1,581	1,186	395	92	1,674	1,267	407	96	21
VCS_800_1	4,282	3,525	757	126	3,107	2,523	584	109	3,614	2,848	766	135	16
VCS_800_2	4,655	3,755	900	145	3,205	2,550	655	114	3,461	2,685	776	133	26
VCS_800_3	4,998	4,036	962	163	3,065	2,484	581	106	3,929	3,046	883	155	21
VCS_800_4	8,561	6,262	2,299	382	7,333	5,809	1,524	274	6,983	4,994	1,989	343	18
VCS_800_5	10,156	6,944	3,212	549	5,480	4,329	1,151	195	6,676	4,731	1,945	330	34
Average	6,530	4,904	1,626	273	4,438	3,539	800	159	4,932	3,660	1,272	219	23

Note. Itr, iterations.

Table 7. Stabilized Algorithms' Results for VCSP Instances with 300–500 Trips

	Stab-NO-S				Stab-GNN-S				
	Time (s)				Time (s)				Time
Instance	Total	RMP	PP	# Itr	Total	RMP	PP	# Itr	reduction (%)
VCS_300_1	65	37	28	47	43	26	17	35	33
VCS_300_2	50	29	21	38	37	22	15	32	26
VCS_300_3	70	40	30	52	58	33	25	48	17
VCS_300_4	94	56	38	56	74	45	29	50	21
VCS_300_5	55	33	22	40	38	22	16	34	30
Average	67	39	28	46	50	30	20	40	25
VCS_400_1	156	100	56	51	131	84	47	49	16
VCS_400_2	161	106	55	50	107	69	38	41	33
VCS_400_3	204	140	64	54	144	95	49	50	29
VCS_400_4	211	142	69	54	167	111	56	52	20
VCS_400_5	189	132	57	49	142	96	46	42	24
Average	184	124	60	51	138	91	47	47	24
VCS_500_1	381	274	107	56	330	224	106	57	13
VCS_500_2	471	354	117	56	320	212	108	53	32
VCS_500_3	493	364	129	66	317	221	96	53	35
VCS_500_4	477	337	140	63	333	208	125	59	30
VCS_500_5	571	453	118	58	379	275	104	56	33
Average	478	356	122	60	335	228	107	55	27

importance in transportation planning and its great difficulty to solve. The VRP consists in planning least-cost delivery routes in order to serve a geographically dispersed set of customers, while respecting the capacity of the vehicles used. In this paper, we focus on the VRPTW, which is a generalization of the VRP where, for each customer, a time window in which the delivery must be made is imposed. To date, the best known algorithms for solving the problem are based on CG where the PP generates vehicle routes. This PP is modeled as an elementary shortest path problem with resource constraints (ESPPRC), where the resource constraints enforce the respect of the time windows, the vehicle capacity, and elementarity requirements. The ESPPRC is usually solved by a labeling algorithm (see, e.g., Irnich and Desaulniers 2005).

The first CG algorithm for the VRPTW was introduced by Desrochers, Desrosiers, and Solomon (1992). They obtained their best computational results by considering a relaxation of the PP that allows multiple visits to a customer in a route but avoids two cycles. This relaxation, however, yields weak lower bounds. Later, Feillet et al. (2004) developed a labeling algorithm for solving the ESPPRC and a CG algorithm where only elementary routes are generated, yielding strong lower bounds (see Contardo, Desaulniers, and Lessard 2015). Because the ESSPRC is NP-hard, it is highly difficult to solve it as a PP for instances where a large number of customers can be visited in a route. Therefore, other PP relaxations have been investigated. In particular, Baldacci, Mingozzi, and Roberti (2011) proposed the ng-route relaxation, which allows the generation of routes with cycles but prohibits certain cycles by defining a neighborhood for each customer. The neighborhood size can be adjusted to define a compromise between the complexity of solving the PP and the quality of the lower bound achieved. In a different research stream, several valid inequalities specific to the VRPTW have been proposed to tighten the lower bound like the two-path cuts (Kohl et al. 1999) and the subset-row cuts (Jepsen et al. 2008). Further enhancements were presented in Pecin et al. (2017). For a broader view of the various techniques devised for the VRP and its variants, the readers can consult the surveys of Desaulniers, Madsen, and Ropke (2014) and Costa, Contardo, and Desaulniers (2019).

5.1. Problem Description and Basic Solution Approach

Given a fleet of vehicles assigned to a single depot, the VRPTW consists of determining a set of possible routes (one route per vehicle used) to deliver goods to a geographically dispersed set of customers while minimizing the total cost, which is most often proportional to the distance traveled. Each customer must be visited exactly once by a vehicle during a specific time window. If the vehicle visits the customer earlier, the vehicle has to wait until the customer is available in order to make the delivery. A route starts from the depot and visits a sequence of customers before returning to the depot, and it is feasible if the total amount of goods delivered does not exceed the capacity of the vehicle and if the time windows of the visited customers are respected.

In this work, we model the VRPTW as a set-partitioning problem like in most recent papers (see, e.g., Pecin et al. 2017) and solve its linear relaxation by CG where the PP is a shortest path problem with resource constraints and two-cycle elimination as in Desrochers, Desrosiers, and Solomon (1992). The PP is solved by a labeling algorithm that can generate multiple columns as for the VCSP.

5.2. Features Considered

The features considered for the VRPTW are quite similar to those of the VCSP because both problems are formulated as set-partitioning problems where every column represents a path with resource constraints. The features are

– *Columns:* cost, reduced cost, number of constraints that the column contributes to (i.e., number of customers visited by the route), resource values (i.e., time and load), boolean value indicating if the column is newly generated, column value (if the column is in the basis or zero otherwise);

– *Constraints:* dual value, number of columns (newly generated or in the RMP basis) contributing to the

constraint (i.e., number of routes that can visit the customer).

5.3. VRPTW Instances

The column selection strategy proposed in this paper aims at reducing degeneracy in the MP and, thus, reducing the time spent solving the RMPs. Thus, it can have a significant impact on the overall computational time if the RMPs take a relatively large proportion of it. Because the known VRPTW benchmark instances typically require more time for solving the PP than the RMP, we have decided to modify some of the Gehring and Homberger instances (Homberger and Gehring 2005) with the aim of creating instances where the PP becomes relatively easy to solve. To do so, the widths of the largest time windows, namely, those with a width exceeding a given threshold, were reduced so that they became shorter than this threshold. The chosen threshold varied from one instance to another between 90 and 120. For our tests, we retained only the instances that spent between 70% and 90% of the total time solving the RMPs. These instances are all derived from the R2 class (the customers are randomly located in a square) and involve 400, 600, or 800 customers.

To perform training, data were collected from only 20 VRPTW instances with 600 customers. This was sufficient given that much more CG iterations (a total of around 10,000 for these 20 instances) were needed to find an optimal solution than for the VCSP. The MILP was again used for column selection and label assignment. To avoid convergence issues, all the remaining columns with negative reduced cost were added to the RMP because their number is not as high as in the VCSP where only 50% of them were added. Additionally, the iterations with less than 150 generated columns were ignored. Once the data generation was completed, the training was performed by splitting the data sets into training and test sets.

5.4. Computational Results

As for the VCSP, this section is divided in two: a ML subsection to present the training results, followed by the results of the integration of the learned model in the CG algorithm.

5.4.1. Machine Learning Phase. The hyperparameter values are presented in Table 8. The values are quite similar to the ones used for the VCSP. The only differences are (i) a larger batch size because the bipartite graphs are smaller because of the reduced number of constraints and (ii) a larger weight assigned to the columns to select (15 versus 10) because the columns with label $y_v = 1$ represent only 7% of all the columns.

We use the same metrics as in Section 4.4.2 to evaluate the performance of the GNN model. From the results reported in Table 9, we can see that the columns with label $y_v = 1$ are correctly predicted with 79.3% accuracy. However, the accuracy of the columns with label 0 is only 68.2%; given their large number due to the imbalance between the two classes, the columns that actually should be selected ($y_v = 1$) represent only 21.8% of the total columns selected by the model. Compared with the results obtained for the VCSP, these results are only slightly different. Thus, with a minimum of tuning, it was possible to obtain a learned model with a very close performance, even though the VRPTW and the VCSP are two completely different problems from a combinatorial optimization perspective.

Table 8. Parameters Used for Training the VRPTW GNN Model

Parameters	Value
Number of iterations	1
Learning rate	10^{-3}
Maximum number of epochs	1,000
Epoch size	32
Batch size	24
Intermediate NN architecture	32×32
Output NN architecture	$32 \times 32 \times 1$
Activation function	ReLU
Optimizer	Adam
Sigmoid class weights	15:1

5.4.2. Optimization Phase. To evaluate the performance of the GNN model, VRPTW instances with 400, 600, and 800 customers were generated. The results on instances with 400 and 800 customers allow us to test the model's ability to generalize to instances of different sizes. To do so, we compare the following three algorithms that apply the acceleration strategies described in Section 2:

NO-S: This is the standard CG algorithm with no column selection involved. All the generated columns are added to the RMP.

GNN-S: The learned model is used to select the columns to add to the RMP. At each iteration, we generate more columns compared with NO-S from which we select the columns to add to the RMP. The number of columns considered is controlled by the number of disjoint blocks.

Table 9. Metrics of the Best GNN Model Obtained for the VRPTW

Metric	Value (%)
Recall	79.3
TNR	68.2
Precision	21.8
Balanced accuracy	73.8

Table 10. Parameter Values Used by the Three Algorithms

Algorithm	n^{-}_{vols}	n^{+}_{cols}	$n^{\max}_{blks}$	min_{sel}
NO-S	$20 \times n_{cons}$	$40 \times n_{cons}$	35	–
GNN-S Rand-S	$10 \times n_{cons}$	$20 \times n_{cons}$	50	150

Rand-S: A subset of the generated columns is randomly selected. The number of columns selected is on average the same as with GNN-S.

The values of the main parameters used by these algorithms are provided in Table 10. Recall that n^{-}_{cols} and n^{+}_{cols} are the minimum and maximum numbers of columns to keep in the RMP, respectively; n_{cons} is the number of constraints in the RMP; $n^{\max}_{blks}$ is the number of disjoint blocks used to ensure column diversity; and min_{sel} is the minimum number of generated columns to activate column selection.

For each tested instance, Table 11 reports the total time in seconds, the time spent solving the RMP and the PP, as well as the number of iterations required by the three algorithms for solving the linear relaxation. The last column corresponds to the time reduction when comparing GNN-S with NO-S. One can see that the column selection with the GNN model gives positive results, yielding average reductions ranging from 20%–29%. These reductions could have been larger if the number of CG iterations performed had not increased. Note also that for the VRPTW, the total time needed to make the predictions is larger than for the VCSP because the total number of iterations is significantly larger. The total prediction time can vary from 50 seconds to 300 seconds depending on the number of iterations and the instance size, which can explain the limited effectiveness of column selection for the largest tested instances. However, despite this, the selection with the GNN model is still more effective than a completely random selection, even if the latter does not require much computational effort.

Table 11. Results for the VRPTW Instances

	NO-S				GNN-S				Rand-S				
	Time (s)				Time (s)				Time (s)				
Instance	Total	RMP	PP	# Itr	Total	RMP	PP	# Itr	Total	RMP	PP	# Itr	Time reduction (%)
400_1	349	245	103	668	244	141	103	790	342	198	144	829	30
400_2	234	177	56	352	162	101	60	384	204	132	72	436	30
400_3	310	225	85	572	219	146	73	612	276	175	101	632	29
400_4	169	121	48	292	116	71	45	333	189	120	69	395	31
400_5	260	195	64	393	201	132	68	509	273	175	97	560	23
400_6	390	278	111	817	308	177	130	762	360	213	145	912	21
400_7	283	207	75	467	181	90	90	557	250	143	106	653	36
400_8	265	227	38	602	177	139	38	678	237	187	50	679	33
400_9	190	155	35	366	94	68	21	305	157	116	40	321	50
400_10	270	168	102	754	211	118	93	665	287	145	142	792	22
Average	272	200	72	528	191	118	73	559	267	160	97	620	29
600_1	1,066	898	168	910	712	455	257	1,178	1,093	821	272	1,226	33
600_2	630	530	100	625	461	323	137	758	541	408	132	765	27
600_3	1,164	1,003	161	929	935	678	256	1,321	1,433	1,103	330	1,385	20
600_4	1,187	739	448	378	803	532	270	820	1,097	792	305	792	32
600_5	1,800	1,614	185	1,130	1,138	864	273	1,475	1,289	1,030	259	1,498	37
600_6	1,105	986	119	768	725	552	173	791	884	678	206	772	34
600_7	1,439	1,244	195	980	1,268	920	348	1,194	1,585	1,230	355	1,250	12
600_8	2,116	1,812	304	1,007	1,867	1,270	597	2,211	1,965	1,500	465	1,277	12
600_9	1,158	920	238	960	1,029	620	409	1,054	1,117	626	491	1,310	11
600_10	1,690	1,547	143	1,191	1,374	1,182	192	1,536	1,617	1,366	251	1,698	19
600_11	754	637	117	746	597	443	154	680	726	570	156	827	21
600_12	1,503	1,189	314	1,062	1,265	821	444	1,300	1,627	1,151	476	1,335	16
600_13	1,957	1,770	187	1,469	1,338	1,119	219	1,476	1,873	1,579	294	1,982	32
600_14	821	687	134	739	600	442	158	842	825	594	231	1,077	27
Average	1,314	1,113	201	921	1,008	730	278	1,188	1,262	960	302	1,228	23
800_1	4,195	3,726	469	1,258	3,307	2,534	773	1,630	4,210	3,531	679	1,748	21
800_2	4,002	3,318	684	1,404	3,349	2,522	827	1,833	3,528	2,732	796	1,873	20
800_3	4,228	3,550	678	1,563	3,221	2,322	899	1,843	3,481	2,368	1,113	2,204	24
800_4	3,182	2,658	524	1,572	2,461	1,750	711	1,788	2,968	2,125	843	2,008	23
800_5	4,594	4,097	497	1,347	3,925	3,229	696	2,410	4,625	3,765	860	2,570	15
Average	4,040	3,470	570	1,159	3,253	2,471	782	1,900	3,762	2,904	858	2,080	20

Note. Itr, iterations.

6. Conclusion

In this paper, we have presented a new approach for accelerating CG. The approach, based on ML, is more suitable to problems that take more time to solve the RMP than the PP as it focuses on reducing the RMP computational time. The objective is to select the most promising columns at each iteration using a supervised learning algorithm trained on data collected from previous executions. We started by defining an effective but expensive column selection strategy with an MILP. Then we replaced such expensive computation with a fast prediction. The learning problem was brought down to a node classification problem on a graph using a GNN. The advantage of using this approach is that, to determine whether a column should be selected or not, the information from the other considered columns is used. Computational results on two well-known problems from the literature, the VCSP and the VRPTW, have indicated average reductions of 20%–30% in the total computational time required to solve the linear relaxation. These results also showed the ability of the GNN model to generalize to instances of sizes different than those used in training. Still, improvements to the combined ML-CG algorithm can be made, either in the choice of features or the learning algorithm used or the method applied to identify the promising columns (maybe by combining information from more than one expert instead of the MILP only).

References

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

Amor HMB, Desrosiers J, Frangioni A (2009) On the choice of explicit stabilizing terms in column generation. *Discrete Appl. Math.* 157(6):1167–1184.

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

Ball M, Bodin L, Dial R (1983) A matching based heuristic for scheduling mass transit crews and vehicles. *Transportation Sci.* 17(1):4–31.

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1996) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.

Battaglia PJ, Hamrick B, Bapst V, Sanchez-Gonzalez A, Zambaldi V, Malinowski M, Tacchetti A, et al. (2018) Relational inductive biases, deep learning, and graph networks. Preprint, submitted June 4, https://arxiv.org/abs/1806.01261.

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.

Contardo C, Desaulniers G, Lessard F (2015) Reaching the elementary lower bound in the vehicle routing problem with time windows. *Networks* 65(1):88–99.

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

de Groot SW, Huisman D (2008) Vehicle and crew scheduling: Solving large real-world instances with an integrated approach. Hickman M, Mirchandani P, Voß S, eds. *Computer-Aided Systems in Public Transport* (Springer, Berlin), 43–56.

Desaulniers G, Desrosiers J, Solomon MM (2002) Accelerating strategies for column generation methods in vehicle routing and crew scheduling problems. Ribeiro CC, Hansen P, eds. *Essays and Surveys in Metaheuristics* (Kluwer, Norwell, MA), 309–324.

Desaulniers G, Desrosiers J, Solomon MM (2005) *Column Generation* (Springer, New York).

Desaulniers G, Madsen OBG, Ropke S (2014) The vehicle routing problem with time windows. Toth P, Vigo D, eds. *Vehicle Routing: Problems, Methods and Applications*, 2nd ed. (Society for Industrial and Applied Mathematics, Philadelphia), 119–159.

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.

du Merle O, Villeneuve D, Desrosiers J, Hansen P (1999) Stabilized column generation. *Discrete Math.* 194(1):229–237.

Elhallaoui I, Desaulniers G, Metrane A, Soumis F (2008) Bi-dynamic constraint aggregation and subproblem reduction. *Comput. Oper. Res.* 35(5):1713–1724.

Elhallaoui I, Metrane A, Desaulniers G, Soumis F (2011) An improved primal simplex algorithm for degenerate linear programs. *INFORMS J. Comput.* 23(4):569–577.

Elhallaoui I, Metrane A, Soumis F, Desaulniers G (2010) Multi-phase dynamic constraint aggregation for set partitioning type problems. *Math. Programming* 123(2):345–370.

Elhallaoui I, Villeneuve D, Soumis F, Desaulniers G (2005) Dynamic aggregation of set-partitioning constraints in column generation. *Oper. Res.* 53(4):632–645.

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.

Freling R, Huisman D, Wagelmans APM (2003) Models and algorithms for integration of vehicle and crew scheduling. *J. Scheduling* 6(1):63–85.

Freling R, Wagelmans APM, Paixão JMP (1999) An overview of models and techniques for integrating vehicle and crew scheduling. Wilson NHM, ed. *Computer-Aided Transit Scheduling* (Springer, Berlin), 441–460.

Gilmore P, Gomory RE (1961) A linear programming approach to the cutting stock problem. *Oper. Res.* 9(6):849–859.

Gondzio J, González-Brevis P, Munari P (2016) Large-scale optimization with the primal-dual column generation method. *Math. Programming Comput.* 8(1):47–82.

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

Gori M, Monfardini G, Scarselli F (2005) A new model for learning in graph domains. *Proc. 2005 IEEE Internat. Joint Conf. Neural Networks*, vol. 2 (IEEE, Piscataway, NJ), 729–734.

Haase K, Desaulniers G, Desrosiers J (2001) Simultaneous vehicle and crew scheduling in urban mass transit systems. *Transportation Sci.* 35(3):286–303.

Homberger J, Gehring H (2005) A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *Eur. J. Oper. Res.* 162(1):220–238.

Huisman D, Freling R, Wagelmans APM (2005) Multiple-depot integrated vehicle and crew scheduling. *Transportation Sci.* 39(4): 491–502.

Irnich S, Desaulniers G (2005) Shortest path problems with resource constraints. Desaulniers G, Desrosiers J, Solomon MM, eds. *Column Generation* (Springer US, Boston), 33–65.

Jepsen M, Petersen B, Spoorendonk S, Pisinger D (2008) Subset-row inequalities applied to the vehicle-routing problem with time windows. *Oper. Res.* 56(2):497–511.

Khalil EB, Bodic PL, Song L, Nemhauser G, Dilkina B (2016) Learning to branch in mixed integer programming. *Proc. Thirtieth*

AAAI Conf. Artificial Intelligence (AAAI Press, Palo Alto, CA), 724–731.

Kohl N, Desrosiers J, Madsen OBG, Solomon MM, Soumis F (1999) 2-Path cuts for the vehicle routing problem with time windows. *Transportation Sci.* 33(1):101–116.

Lübbecke ME, Desrosiers J (2005) Selected topics in column generation. *Oper. Res.* 53(6):1007–1023.

Lodi A, Zarpellon G (2017) On learning and branching: A survey. *TOP* 25(2):207–236.

Mesquita M, Paias A (2008) Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. *Comput. Oper. Res.* 35(5):1562–1575.

Pecin D, Contardo C, Desaulniers G, Uchoa E (2017) New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS J. Comput.* 29(3):489–502.

Pessoa A, Sadykov R, Uchoa E, Vanderbeck F (2018) Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS J. Comput.* 30(2): 339–360.

Rousseau LM, Gendreau M, Feillet D (2007) Interior point stabilization for column generation. *Oper. Res. Lett.* 35(5):660–668.

Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G (2009) The graph neural network model. *IEEE Trans. Neural Networks* 20(1):61–80.

Tahir A, Desaulniers G, El Hallaoui I (2019) Integral column generation for the set partitioning problem. *Eur. J. Transportation Logist.* 8(5):713–744.

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

Vanderbeck F (2005) Implementing mixed integer column generation. Desaulniers G, Desrosiers J, Solomon MM, eds. *Column Generation* (Springer, Boston), 331–358.

Wu Z, Pan S, Chen F, Long G, Zhang C, Yu PS (2019) A comprehensive survey on graph neural networks. Preprint, submitted January 3, https://arxiv.org/abs/1901.00596.

Zaghrouti A, El Hallaoui I, Soumis F (2018) Improved integral simplex using decomposition for the set partitioning problem. *Eur. J. Comput. Optim.* 6(2):185–206.

Zaghrouti A, Soumis F, El Hallaoui I (2014) Integral simplex using decomposition for the set partitioning problem. *Oper. Res.* 62(2): 435–449.