Les Cahiers du GERAD ISSN: 0711–2440

Machine-learning-based column selection for column generation

M. Morabit,
G. Desaulniers, A. Lodi

G–2020–29

May 2020

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2020
– Bibliothèque et Archives Canada, 2020

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2020
– Library and Archives Canada, 2020

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Machine-learning-based column selection for column generation

Mouad Morabit [a,b,c]

Guy Desaulniers [a,b]

Andrea Lodi [a,b,c]

[a] *GERAD, Montréal (Québec), Canada, H3T 2A7*

[b] *Department of Mathematics and Industrial Engineering, Polytechnique Montréal (Québec) Canada, H3C 3A7*

[c] *Canada Excellence Research Chair in Data Science for Real-Time Decision-Making, Polytechnique Montréal, Montréal (Québec), Canada, H3C 3A7*

mouad.morabit@gerad.ca
guy.desaulniers@gerad.ca
andrea.lodi@polymtl.ca

May 2020
Les Cahiers du GERAD
G–2020–29

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%.

Keywords: Column generation, machine learning, column selection

Acknowledgments: We thank Giro Inc. and the Natural Sciences and Engineering Research Council of Canada for their financial support through the grant CRDPJ 520349–17.

1 Introduction

Column generation (CG) is an iterative method used to solve the linear relaxation of large-scale optimization models which 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 generate new ones that have the potential to improve the current solution, i.e., 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, crew scheduling problems, and many other problems. To derive integer solutions, the method is embedded in a branch-and-bound framework, i.e., CG is performed at every node of the search tree, yielding a branch-and-price algorithm (see Barnhart et al., 1996; Desaulniers et al., 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 et al., 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 et al., 2007; Gondzio et al., 2016) for solving the RMP produces central dual solutions which can also help to reduced 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 in 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 in Elhallaoui et al. (2010, 2008) with the aim of maintaining a higher level of aggregation and further reducing degeneracy. DCA has fostered the development of the improved primal implex 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, i.e., 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 et al. (2014, 2018) developed variants of the integral simplex using decomposition algorithm 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 et al. (2019) introduced a combination of ISUD and CG that is called integral column generation (ICG). 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 et al. (2018). 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 et al. (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 a 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, i.e., 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 details the different steps, from data generation, to training and prediction. Sections 4 and 5 will be devoted to our two case studies, namely, the VCSP and the VRPTW, respectively. In section 6, we report and discuss the results of our computational experiments. Finally, conclusions are drawn in Section 7.

2 Basic model and column generation

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

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

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

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

where $\mathcal{P}$ is the index set of the variables θ_p; $c_p \in \mathbb{R}$ and $a_p \in \mathbb{R}^m$ are the cost coefficient and constraint coefficient vector of θ_p, respectively; and $b \in \mathbb{R}^m$ 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 a 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 :

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

If negative reduced cost columns are found, they are added to the RMP, which is then re-optimized 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.

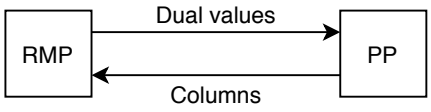


Figure 1: Standard CG process

Several strategies can be applied to accelerate column generation (see Desaulniers et al., 2002). We present here three 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 (SPPRC, Irnich and Desaulniers, 2005) and solved by dynamic programming (more precisely, a labeling algorithm) which offers this possibility at no extra computational effort.
- 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 solve the PP as long as an exact algorithm is invoked in the last iteration to prove optimality. 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.

3 Methodology

At each CG iteration, after reoptimizing the RMP and solving the PP, a set $\mathcal{G}$ of columns is generated. 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 $y \in \{0, 1\}$, i.e., 1 if the column is to be selected and 0 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 dataset, $\boldsymbol{x}_i$ 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 minimum number of columns in the RMP so that it remains fast to reoptimize. We also want a maximum reduction of the objective value at each CG iteration. For these reasons, a good option is to add the columns that will take a positive value after the RMP reoptimization, i.e., we are looking for a minimal set of columns that will ensure a maximal decrease of the objective value. This column selection problem 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 1 if column p is selected and 0 otherwise. To minimize the number of selected columns, a small penalty ϵ is incurred for each selected column.

The proposed MILP is as follows:

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

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

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

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

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

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 $y_p = 1$, i.e.,

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

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

The overall procedure is illustrated in Figure 2.

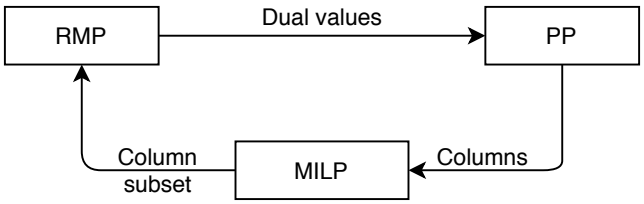


Figure 2: CG algorithm with MILP column selection

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, e.g., a route or a schedule, features such as resource values, number of nodes in the path, or distances between nodes can be considered. This is in addition to general features that can be used for any problem such as the cost, reduced cost, dual values of the constraints that the column contributes to. 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, i.e., $\Omega_\ell \cup \mathcal{G}_\ell$. 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 $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n$ 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 or not 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 combinatiorial optimization. They were initially introduced by Gori et al. (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.

Due to 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 agreggate 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 a vector of aggregated information vector $\boldsymbol{a}_v^{(k)}$ for each node $v \in V$:

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

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 obtained the updated node representation vectors:

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

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

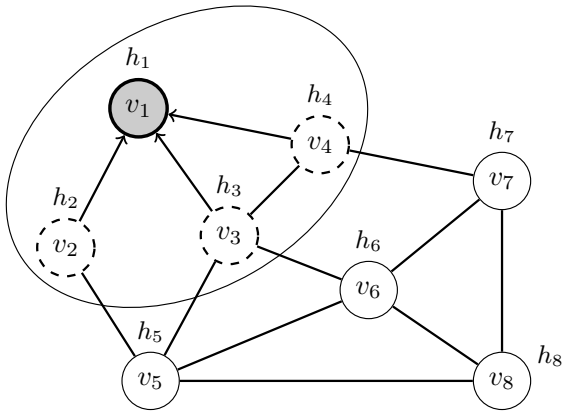


Figure 3: Representation vector h_1 of node v_1 is updated by aggregating the representations h_2, h_3, h_4 of its neighbor nodes

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 be used to predict its label, denoted l_v,

by applying a final learned transformation, denoted *out*, i.e.,

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

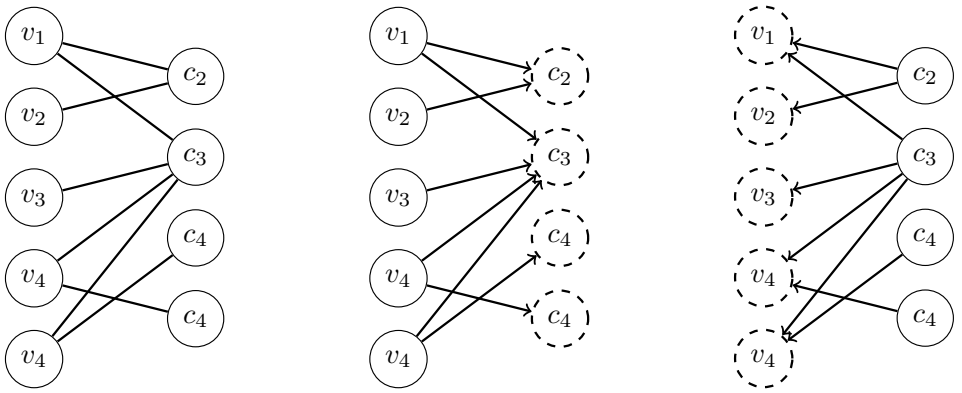
The learned functions $\phi^{(k)}$, $\psi_k^{(k)}$ and *out* are implemented by using multilayer perceptron feedforward neural networks (Goodfellow et al., 2016).

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 4a). 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.

Since there are two node types, each iteration proceeds in two phases: a first phase is performed to update the constraint representations (Figure 4b), followed by a second phase that updates the column representations (Figure 4c). 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.



(a) Bipartite graph with two node types V and C

(b) Phase 1 : constraint representations update

(c) Phase 2 : column representations update

Figure 4: Our model represented by a bipartite graph for column classification/selection

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* function. The functions $\phi_C^{(k)}$, $\psi_C^{(k)}$, $\phi_V^{(k)}$, and $\psi_V^{(k)}$ are 2-layer feedforward

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** $k = 1, ..., K$ **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: $y_v = out(\boldsymbol{h}_v^{(K)}), \quad \forall v \in V$

neural networks with ReLU activation functions and *out* is a 3-layer feedforward 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, i.e., 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 non-basic 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 is 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 is 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 $\mathcal{G}_\ell$.

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 - VSP) before the crew schedules (crew scheduling problem - CSP). This approach has been strongly criticized (Ball et al., 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 et al. (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 et al. (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 et al. (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 et al. (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 et al. (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 et al. (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 et al. (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 may be used:

Columns: cost, reduced cost, number of constraints that the column contributes to (the set partitioning model of Haase et al. (2001) have four groups of constraints and it is, thus, possible to have a distinct feature for each group), resource values (i.e., work time, duty duration), duty type (if applicable), columnIsNew (boolean value indicating if the column is newly generated or in the basis), column value (if the column is in the basis, 0 otherwise), column incompatibility degree (as defined in Elhallaoui et al., 2010).

Constraints: dual value, number of columns 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 et al. (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 7000 iteration datasets (i.e., 7000 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 (blue curve), and the execution with the MILP selection (orange curve). 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.

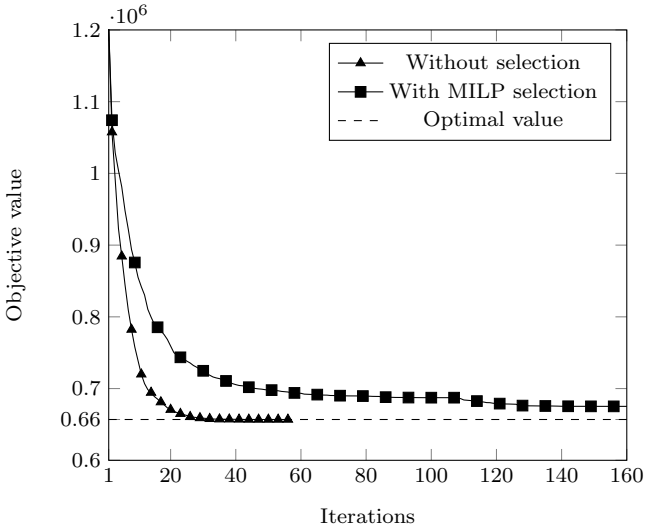


Figure 5: Comparison between the execution without selection and with MILP selection

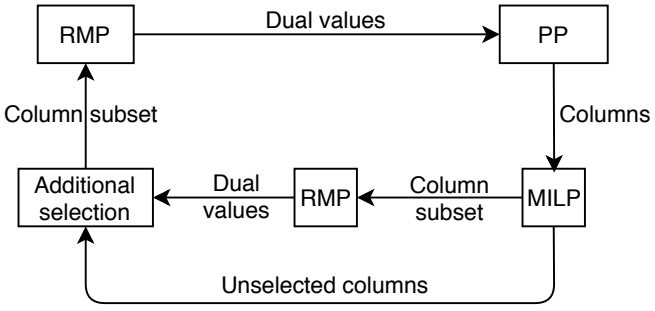


Figure 6: CG algorithm with MILP column selection and additional columns

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

Table 1: Results with the MILP selection

Instance	# Constraints	Without Selection				With Selection				Time
		Time	Time RMP	# Itr	# Col	Time	Time RMP	# Itr	# Col	Reduction
VCS_400_1	1740	310	244	104	20,195	174	132	44	15,624	44%
VCS_400_2	1757	311	262	46	17,942	178	140	41	14,754	53%
VCS_400_3	1752	464	405	53	20,140	298	243	57	17,101	36%
VCS_400_4	1757	363	308	50	19,453	284	233	51	16,683	22%
VCS_400_5	1759	319	270	46	18,162	218	175	43	14,352	32%
Average	**1753**	**353**	**297**	**59**	**19,178**	**230**	**184**	**47**	**15,703**	**34%**

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 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, we select in addition of the column selected by the MILP 50% of the unselected columns that have a negative reduced cost of 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 since we are interested only on assessing the effectiveness of the selection. The MILP will be replaced afterwards 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.

Table 2: Parameters used for training the VCSP GNN model

Parameters	Value
Number of iterations	1
Learning rate	10^{-3}
Maximum number of epochs	1000
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

The data set is split into two sets: training and test sets (75% for training and 25% for test). The training is performed in 1000 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 1 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 2-layer neural network with 32 neurons on each layer and ReLU activations. This architecture implements the functions $\phi_C^{(k)}$, $\psi_C^{(k)}$, $\phi_V^{(k)}$, $\psi_V^{(k)}$ 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,

$$\begin{aligned} \text{Recall} &= \frac{tp}{tp + fn} \\ TNR &= \frac{tn}{tn + fp} \\ \text{Precision} &= \frac{tp}{tp + fp} \\ \text{Balanced Accuracy} &= \frac{TPR + TNR}{2}, \end{aligned}$$

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 by 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. Finally, note that the use of the normal accuracy ($accuracy = \frac{tp+tn}{tp+tn+fp+fn}$) 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.

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%

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 different algorithms:

No selection (NO-S): This is the CG algorithm with no selection involved: all the generated columns are added to the RMP

MILP selection (MILP-S): The MILP is used to select the columns, with 50% additional columns to avoid convergence issues. The total computing time does not include the time to solve the MILP because we are looking to replace it with a fast approximation.

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.

Table 4: Parameter values used by the five algorithms

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

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. Different values for n^-_{cols} and n^+_{cols} were tested and the overall best ones were retained. A low penalty $\epsilon = 0.1$ is used for every selected column in the MILP since the selection with the MILP alone suffers from convergence issues and a higher penalty implies less selected columns. In addition to the low penalty, 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%) and therefore no additional columns are needed.

The results obtained for 15 VCSP instances involving 300 to 500 trips are reported in Table 5. For each algorithm, the time in seconds and the number of CG iterations is reported. The time reduction column compares the GNN-S to the NO-S algorithm. The first observation is that the computational time taken by GNN-S and MILP-S are very close to each other. The difference is that 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 selection strategies 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 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 to NO-S, GNN-S gives on average a gain ranging from 25% to 30% as shown in the last column. The time taken by the model to make the prediction 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.

Table 5: Results for VCSP instances with 300 to 500 trips (times in seconds)

Instance	NO-S		MILP-S		GNN-S		Sort-S		Rand-S		Time
	Time	# Itr	Time	# Itr	Time	# Itr	Time	# Itr	Time	# Itr	reduction
VCS_300_1	105	63	73	62	69	54	93	74	110	81	**34%**
VCS_300_2	72	41	49	40	49	37	71	55	73	52	**32%**
VCS_300_3	131	83	115	94	95	76	156	151	150	118	**27%**
VCS_300_4	158	73	107	76	110	75	151	118	175	123	**30%**
VCS_300_5	76	58	54	49	55	45	79	61	86	60	**28%**
Average	**108**	**63**	**79**	**64**	**75**	**57**	**110**	**91**	**118**	**86**	**30%**
VCS_400_1	265	74	170	56	180	75	287	108	270	95	**32%**
VCS_400_2	259	72	187	58	158	55	305	112	227	71	**39%**
VCS_400_3	365	93	277	89	278	96	435	186	427	140	**24%**
VCS_400_4	315	77	254	73	221	70	391	137	370	109	**30%**
VCS_400_5	234	59	199	60	207	61	319	110	277	79	**12%**
Average	**287**	**75**	**217**	**67**	**208**	**71**	**347**	**130**	**314**	**98**	**25%**
VCS_500_1	721	79	449	75	464	78	781	150	781	108	**36%**
VCS_500_2	769	79	490	79	455	72	766	144	779	108	**41%**
VCS_500_3	910	113	510	88	490	83	1105	229	881	131	**46%**
VCS_500_4	572	77	485	87	496	90	616	102	583	105	**13%**
VCS_500_5	818	92	645	94	703	97	794	107	926	150	**14%**
Average	**758**	**88**	**515**	**84**	**521**	**84**	**812**	**146**	**790**	**120**	**26%**

Additional experiments have been conducted on larger instances with 600 to 800 trips, comparing only the first three strategies. 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 to 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.

Table 6: Results of VCSP instances with 600 to 800 trips (times in seconds)

Instance	NO-S		MILP-S		GNN-S		Time
	Time	# Itr	Time	# Itr	Time	# Itr	reduction
VCS_600_1	1319	108	1049	100	998	96	**24%**
VCS_600_2	1445	112	1005	92	1031	99	**29%**
VCS_600_3	983	89	659	75	805	93	**18%**
VCS_600_4	1930	159	1659	156	1570	165	**19%**
VCS_600_5	1379	137	1028	108	1025	115	**26%**
Average	**1411**	**121**	**1080**	**106**	**1085**	**115**	**23%**
VCS_700_1	2396	101	1977	107	1882	97	**21%**
VCS_700_2	1613	80	1118	73	1212	78	**25%**
VCS_700_3	2074	106	1470	89	1546	97	**25%**
VCS_700_4	2465	102	1854	104	2061	109	**16%**
VCS_700_5	2101	85	1487	87	1672	101	**20%**
Average	**2129**	**94**	**1581**	**92**	**1674**	**96**	**21%**
VCS_800_1	4282	126	3107	109	3614	135	**16%**
VCS_800_2	4655	145	3205	114	3461	133	**26%**
VCS_800_3	4998	163	3065	106	3929	155	**21%**
VCS_800_4	8561	382	7333	274	6983	343	**18%**
VCS_800_5	10156	549	5480	195	6676	330	**34%**
Average	**6530**	**273**	**4438**	**159**	**4932**	**219**	**23%**

5 Case study II : Vehicle routing problem with time windows

The vehicle route problem (VRP) has been the subject of intensive research for over fifty years because of its 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 et al. (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 2-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 et al., 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 et al. (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 2-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 et al. (2014) and Costa et al. (2019).

5.1 Problem description and basic solution approach

Given a fleet of vehicles assigned to a single depot, the VRPTW consists in 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 2-cycle elimination as in Desrochers et al. (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 since 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 clients visited by the route), resource values (i.e., time and load), columnIsNew (boolean value indicating if the column is newly generated or in the basis), column value (if the column is in the basis or 0 otherwise).

Constraints: dual value, number of columns contributing to the constraint (i.e, number of routes that can visit the client).

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 & 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 was 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 since 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 datasets 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 7. 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 due to the reduced number of constraints, 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.

Table 7: Parameters used for training the VRPTW GNN model

Parameters	Value
Number of iterations	1
Learning rate	10^{-3}
Maximum number of epochs	1000
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

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 8, 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%, and given their large number due to the imbalance between the two classes, the columns that actually should

Table 8: 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%

be selected ($y_v = 1$) represent only 21.8% of the total columns selected by the model. Compared to 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.

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:

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

GNN selection (GNN-S): The learned model is used to select the columns to add to the RMP. At each iteration, the PP generates around 50% more columns compared to NO-S.

Random selection (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 9. Recall that n^-_{cols} and n^+_{cols} correspond to the minimum and maximum number of columns to keep in the RMP, respectively, n_{cons} is the number of constraints in the RMP, and min_{sel} is the minimum number of generated columns to activate column selection.

Table 9: Parameter values used by the three algorithms

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

For each tested instance, Table 10 reports the total time in seconds and the number if CG 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 gains ranging from 20% to 29%. These gains 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 s to 300 s 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 10: Results for the VRPTW instances

Instance	NO-S		GNN-S		Rand-S		Time
	Time	# Itr	Time	#Itr	Time	# Itr	reduction
400_1	349	668	244	790	342	829	**30%**
400_2	234	352	162	384	204	436	**30%**
400_3	310	572	219	612	276	632	**29%**
400_4	169	292	116	333	189	395	**31%**
400_5	260	393	201	509	273	560	**23%**
400_6	390	817	308	762	360	912	**21%**
400_7	283	467	181	557	250	653	**36%**
400_8	265	602	177	678	237	679	**33%**
400_9	190	366	94	305	157	321	**50%**
400_10	270	754	211	665	287	792	**22%**
Average	**272**	**528**	**191**	**559**	**267**	**620**	**29%**
600_1	1066	910	712	1178	1093	1226	**33%**
600_2	630	625	461	758	541	765	**27%**
600_3	1164	929	935	1321	1433	1385	**20%**
600_4	1187	378	803	820	1097	792	**32%**
600_5	1800	1130	1138	1475	1289	1498	**37%**
600_6	1105	768	725	791	884	772	**34%**
600_7	1439	980	1268	1194	1585	1250	**12%**
600_8	2116	1007	1867	2211	1965	1277	**12%**
600_9	1158	960	1029	1054	1117	1310	**11%**
600_10	1690	1191	1374	1536	1617	1698	**19%**
600_11	754	746	597	680	726	827	**21%**
600_12	1503	1062	1265	1300	1627	1335	**16%**
600_13	1957	1469	1338	1476	1873	1982	**32%**
600_14	821	739	600	842	825	1077	**27%**
Average	**1314**	**921**	**1008**	**1188**	**1262**	**1228**	**23%**
800_1	4195	1258	3307	1630	4210	1748	**21%**
800_2	4002	1404	3349	1833	3528	1873	**20%**
800_3	4228	1563	3221	1843	3481	2204	**24%**
800_4	3182	1572	2461	1788	2968	2008	**23%**
800_5	4594	1347	3925	2410	4625	2570	**15%**
Average	**4040**	**1159**	**3253**	**1900**	**3762**	**2080**	**20%**

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 a 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 to 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

Alejandro Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. INFORMS Journal on Computing, 29:185–195, 01 2017. doi: 10.1287/ijoc.2016.0723.

Hatem M.T. Ben Amor, Jacques Desrosiers, and Antonio Frangioni. On the choice of explicit stabilizing terms in column generation. Discrete Applied Mathematics, 157(6):1167–1184, 2009. doi: https://doi.org/10.1016/j.dam.2008.06.021.

Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. Operations Research, 59(5):1269–1283, 2011. doi: 10.1287/opre.1110.0975.

Michael Ball, Lawrence Bodin, and Robert Dial. A matching based heuristic for scheduling mass transit crews and vehicles. Transportation Science, 17(1):4–31, 1983. doi: 10.1287/trsc.17.1.4.

Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. Operations Research, 46:316–329, 1996.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. ArXiv e-prints, 2018.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. ArXiv e-prints, 2018.

Claudio Contardo, Guy Desaulniers, and François Lessard. Reaching the elementary lower bound in the vehicle routing problem with time windows. Networks, 65(1):88–99, 2015.

Luciano Costa, Claudio Contardo, and Guy Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. Transportation Science, 53(4):946–985, 2019. doi: 10.1287/trsc.2018.0878.

Sebastiaan W. de Groot and Dennis Huisman. Vehicle and crew scheduling: Solving large real-world instances with an integrated approach. In Mark Hickman, Pitu Mirchandani, and Stefan Voß, editors, Computer-aided Systems in Public Transport, pages 43–56, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. Accelerating strategies for column generation methods in vehciel routing and crew scheduling problems. In Celso C. Ribeiro and Pierre Hansen, editors, Essays and Surveys in Metaheuristics, pages 309–324. Kluwer, Norwell, MA, 2002.

Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon. Column generation. Springer, New York, January 2005.

Guy Desaulniers, Oli B G Madsen, and Stefan Ropke. The vehicle routing problem with time windows. In P. Toth and D. Vigo, editors, Vehicle Routing: Problems, Methods and Applications, chapter 5, pages 119–159. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2nd edition, 2014.

Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. Operations Research, 40(2):342–354, 1992. doi: 10.1287/opre.40.2.342.

Olivier du Merle, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen. Stabilized column generation. Discrete Mathematics, 194(1):229–237, 1999. doi: https://doi.org/10.1016/S0012-365X(98)00213-1.

Issmail Elhallaoui, Daniel Villeneuve, François Soumis, and Guy Desaulniers. Dynamic aggregation of set-partitioning constraints in column generation. Operations Research, 53(4):632–645, 2005. doi: 10.1287/opre.1050.0222.

Issmail Elhallaoui, Guy Desaulniers, Abdelmoutalib Metrane, and François Soumis. Bi-dynamic constraint aggregation and subproblem reduction. Computers & Operations Research, 35(5):1713–1724, 2008.

Issmail Elhallaoui, Abdelmoutalib Metrane, François Soumis, and Guy Desaulniers. Multi-phase dynamic constraint aggregation for set partitioning type problems. Mathematical Programming, 123(2):345–370, 2010. doi: 10.1007/s10107-008-0254-5.

Issmail Elhallaoui, Abdelmoutalib Metrane, Guy Desaulniers, and François Soumis. An improved primal simplex algorithm for degenerate linear programs. INFORMS Journal on Computing, 23(4):569–577, 2011. doi: 10.1287/ijoc.1100.0425.

Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. Networks, 44(3):216–229, 2004.

Richard Freling, Albert P. M. Wagelmans, and José M. Pinto Paixão. An overview of models and techniques for integrating vehicle and crew scheduling. In Nigel H. M. Wilson, editor, Computer-Aided Transit Scheduling, pages 441–460, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-642-85970-0.

Richard Freling, Dennis Huisman, and Albert P. M. Wagelmans. Models and algorithms for integration of vehicle and crew scheduling. Journal of Scheduling, 6(1):63–85, 2003. doi: 10.1023/A:1022287504028.

P.C. Gilmore and Ralph E. Gomory. A linear programming approach to the cutting stock problem. Operations Research, 9:849–859, 1961.

Jacek Gondzio, Pablo González-Brevis, and Pedro Munari. Large-scale optimization with the primal-dual column generation method. Mathematical Programming Computation, 8(1):47–82, 2016.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., volume 2, pages 729–734 vol. 2, July 2005.

Knut Haase, Guy Desaulniers, and Jacques Desrosiers. Simultaneous vehicle and crew scheduling in urban mass transit systems. Transportation Science, 35(3):286–303, 2001. doi: 10.1287/trsc.35.3.286.10153.

Jörg Homberger and Hermann Gehring. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. European Journal of Operational Research, 162(1):220–238, 2005. doi: https://doi.org/10.1016/j.ejor.2004.01.027.

Dennis Huisman, Richard Freling, and Albert P. M. Wagelmans. Multiple-depot integrated vehicle and crew scheduling. Transportation Science, 39(4):491–502, 2005. doi: 10.1287/trsc.1040.0104.

Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, Column Generation, pages 33–65. Springer US, Boston, MA, 2005.

Mads Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research, 56(2):497–511, 2008. doi: 10.1287/opre.1070.0449.

Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16, pages 724–731. AAAI Press, 2016.

Niklas Kohl, Jacques Desrosiers, Oli B. G. Madsen, Marius M. Solomon, and François Soumis. 2-path cuts for the vehicle routing problem with time windows. Transportation Science, 33(1):101–116, 1999. doi: 10.1287/trsc.33.1.101.

Andrea Lodi and Giulia Zarpellon. On learning and branching: a survey. TOP, 25(2):207–236, Jul 2017. doi: 10.1007/s11750-017-0451-6.

Marco E. Lübbecke and Jacques Desrosiers. Selected topics in column generation. Operations Research, 53 (6):1007–1023, 2005. doi: 10.1287/opre.1050.0234.

Marta Mesquita and Ana Paias. Set partitioning/covering-based approaches for the integrated vehicle and crew scheduling problem. Computers & Operations Research, 35(5):1562–1575, 2008. doi: https://doi.org/10.1016/j.cor.2006.09.001.

Diego Pecin, Claudio Contardo, Guy Desaulniers, and Eduardo Uchoa. New enhancements for the exact solution of the vehicle routing problem with time windows. INFORMS Journal on Computing, 29(3):489–502, 2017. doi: 10.1287/ijoc.2016.0744.

A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. INFORMS Journal on Computing, 30(2):339–360, 2018. doi: 10.1287/ijoc.2017.0784.

Louis Martin Rousseau, Michel Gendreau, and Dominique Feillet. Interior point stabilization for column generation. Operations Research Letters, 35(5):660–668, 2007.

F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. IEEE Transactions on Neural Networks, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Adil Tahir, Guy Desaulniers, and Issmail El Hallaoui. Integral column generation for the set partitioning problem. EURO Journal on Transportation and Logistics, 8(5):713–744, 2019. doi: 10.1007/s13676-019-00145-6.

François Vanderbeck. Implementing mixed integer column generation. In Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors, Column Generation, pages 331–358. Springer US, Boston, MA, 2005.

Roman Václavík, Antonín Novák, Přemysl Šůcha, and Zdeněk Hanzálek. Accelerating the branch-and-price algorithm using machine learning. European Journal of Operational Research, 271(3):1055–1069, 2018. doi: https://doi.org/10.1016/j.ejor.2018.05.046.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. ArXiv e-prints, 2019.

Abdelouahab Zaghrouti, François Soumis, and Issmail El Hallaoui. Integral simplex using decomposition for the set partitioning problem. Operations Research, 62(2):435–449, 2014. doi: 10.1287/opre.2013.1247.

Abdelouahab Zaghrouti, Issmail El Hallaoui, and François Soumis. Improved integral simplex using decomposition for the set partitioning problem. EURO Journal on Computational Optimization, 6(2):185–206, Jun 2018. doi: 10.1007/s13675-018-0098-6.