CIRRELT

CIRRELT-2022-25

COIL: A Deep Architecture for Column Generation

Behrouz Babaki
Laurent Charlin
Sanjay Dominik Jena

September 2022

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

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

COIL: A Deep Architecture for Column Generation

Behrouz Babaki[1,*], Laurent Charlin[1,2], Sanjay Dominik Jena[3]

1. HEC Montréal
2. Mila – Quebec Institute
3. Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) and Analytics, Operations, and Information Technologies Department, School of Management, Université du Québec à Montréal

Abstract. Column generation is a popular method to solve large-scale linear programs with an exponential number of variables. Several important applications, such as the vehicle routing problem, rely on this technique in order to be solved. However, in practice, column generation methods suffer from slow convergence (i.e. they require too many iterations). Stabilization techniques, which carefully select the column to add at each iteration, are commonly used to improve convergence. In this work, we frame the problem of selecting which columns to add as one of sequential decision-making. We propose a neural column generation architecture that iteratively selects columns to be added to the problem. Our architecture is inspired by stabilization techniques and predicts the optimal duals, which are then used to select the columns to add. We proposed architecture, trained using imitation learning. Exemplified on the Vehicle Routing Problem, we show that several machine learning models yield good performance in predicting the optimal duals and that our architecture outperforms them as well as a popular state-of-the-art stabilization technique. Further, the architecture approach can generalize to instances larger than those observed during training.

Keywords: Column Generation, Stabilization, Imitation Learning, Graph Neural Networks

Acknowledgements. We are thankful to Louis-Martin Rousseau, Maxime Gasse, Prateek Gupta, Seyed Mehran Kazemi, Thibaut Vidal and Giulia Zarpellon for enlightening discussions. This research was partially supported by IVADO, FRQNT, NSERC, the CIFAR AI Chairs program, Calcul Québec and Compute Canada.

---

* Corresponding author: behrouz.babaki@gmail.com

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2022

1 Introduction

Column generation is widely regarded as an efficient technique for solving linear programming problems that have an exponential number of variables. This includes, most notably, combinatorial optimization problems that are hard to solve, such as cutting stock, vehicle routing and crew scheduling. While the number of variables in these problems is typically large, a key observation to solve them is that only a small subset of the variables may be part of the optimal solution (i.e., have non-zero values). Therefore, in principle, it is possible to find the optimal solution using a small subset of columns. Column generation is an iterative procedure: starting with a subset of columns, at each iteration, it decides which column(s) to add to the problem. The procedure stops once the optimal solution is obtained.

Despite the theoretical guarantees of this procedure, in practice, it is often difficult to efficiently converge to the optimal solution. In particular, unless special care is taken, its vanilla version is likely to add many columns which do not serve the purpose of reaching the optimal solution. Several classes of methods have been proposed for alleviating this issue, including interior-point stabilization (see, e.g. Rousseau et al., 2007), which serves as an inspiration and a point of comparison for our work. Despite those improvement, convergence and thus solving hard combinatorial optimization problems remains a challenge.

Recently, machine learning technique have been increasingly used to improve the performance of combinatorial optimization algorithms (see, e.g. Bengio et al., 2021, for a general overview). Examples include learning to branch (Khalil et al., 2016; Gasse et al., 2019), learning to select cuts (Tang et al., 2020), and general learning heuristics (Khalil et al., 2017; Nazari et al., 2018). While there has been a growing interest in applying machine learning methods to improve classical optimization algorithms, few learning-based techniques exist for column generation. Morabit et al. (2021) learn a classification model to identify promising columns when the pricing algorithm returns several columns. Shen et al. (2021) learn a heuristic that solves the pricing subproblem of the graph coloring problem.

In this paper, we take a learning-based approach, which we call ***Co**lumn Generation by **I**mitation **L**earning (**COIL**)*. We formulate column generation as sequential decision making and propose a neural architecture to predict which column to add at each iteration. As such, our approach explicitly acknowledges the iterative and sequential nature of Column Generation, and is therefore radically different from existing approaches that consider iterations separately. The model first encodes the problem (i.e., its variables and constraints) using a graphical neural network (see, e.g. Gasse et al., 2019). The learned representations (of the rows) are combined with instance-specific features and a differentiable optimization layer then predicts the optimal duals. This exact optimization layer guarantees the validity of the duals. In a final step, the predicted duals are decoded as a distribution over all candidate variables. The model is trained to imitate an expert that picks high-quality columns to add.

We here use as a case study the *Capacitated Vehicle Routing Problem (CVRP)*, which is is strongly useful in practice (e.g. among logistics companies) and is one of the most popular combinatorial optimization problems in the scientific literature (Toth and Vigo, 2014), particularly in the context of Column Generation (Desrochers et al., 1992). Trained on instances of the CVRP, we demonstrate that our approach outperforms a strong stabilization method (Rousseau et al., 2007) in terms of number of columns added until convergence. Further, we validate our design choices using an ablation study and by comparing it to other neural architectures. The main contributions of our paper can be summarized as follows:

- We formulate column selection as a sequential decision making problem and propose a neural architecture for it.
- Included in the architecture, we propose a method for dealing with an exponential number of actions and ensure the correctness of the learning-based method by plugging exact optimization layers into the deep network.
- We validate the effectiveness of the method on the case of the CVRP and its corresponding problem instances.

The remainder of this paper is organized as follows. Section 2 reviews the Capacitated Vehicle Routing Problem and its corresponding Column Generation formulation. Section 3 introduces the Column Generation framework based on Imitation Learning, providing details on the used expert policy and its architecture.

Computational experiments to validate and compare the performance of our method to existing approaches are presented in Section 4. Finally, we conclude and lay out future research directions in Section 5.

2 Column Generation

We now discuss the application of Column Generation for the Capacitated Vehicle Routing Problem. Given the locations of N customers and one depot, the customer demands $d_i, i \in \{1, \dots, N\}$, and vehicle capacity c, find a set of routes starting and ending at the depot such that every customer is visited on some route, the sum of demands of customers on each route does not exceed the vehicle capacity, and the total travelled distance (i.e. the sum of lengths of the routes) is minimized.

This problem can be formulated as an integer linear program with an exponential number of variables (Desrochers et al., 1992). Each variable in this formulation corresponds to a subset of customers $r \in \{1, \dots, N\}$ that can form a valid route (i.e. $\sum_{i \in r} d_i \leq c$). This formulation only considers the shortest route formed by each such subset. Hence we simply call each subset a route, and denote the set of all routes by R. The cost of route r, denoted by c_r, is the length of the shortest tour that visits all customers in r and the depot. Let binary variable x_r be 1 if route r is selected, and 0 otherwise. Further, for every route $r \in R$ and customer $i \in \{1, \dots, N\}$, let constant $\boldsymbol{e}_{ir}$ be 1 if $i \in r$ and 0 otherwise. The CVRP can be formulated as:

$$(I): \quad \min \quad \sum_{r \in R} c_r x_r \tag{1}$$

$$\text{s.t.} \quad \sum_{r \in R} \boldsymbol{e}_{ir} x_r \geq 1 \quad \forall i \in \{1, \dots, N\} \tag{2}$$

$$\boldsymbol{x} \in \{0, 1\}^{|R|}. \tag{3}$$

Objective function (1) minimizes the total costs of the selected routes. Constraints (2) ensure that every customer is visited at least once. Even though in the optimal solution every customer is visited exactly once, this constraint is formulated as an inequality for practical reasons. In the rest of this paper, we will deal with the problem of solving the linear relaxation of this formulation (denoted by problem P) where Constraint (3) is replaced by $\boldsymbol{x} \geq 0$. The integrality constraints can then be enforced through a branch-and-bound procedure.

The exponential size of R makes it impossible to explicitly represent the formulation I or P beyond a certain number of routes. Column generation is a procedure that iteratively adds promising columns (which, in the case of the CVRP, correspond to routes) until the problem is solved to optimality. For ease of exposition, we will describe the working principle of Column Generation using the dual formulation of problem P:

$$(D): \quad \max \quad \sum_{i=1}^{N} \boldsymbol{\lambda}_i \tag{4}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} \boldsymbol{e}_{ir} \boldsymbol{\lambda}_i \leq c_r \quad \forall r \in R \tag{5}$$

$$\boldsymbol{\lambda} \geq 0. \tag{6}$$

The dual formulation maximizes the sum over variables $\boldsymbol{\lambda}_i$, which represent the marginal cost of serving customer i. Constraints (5) ensure that for every route r, the sum of marginal costs of the customers in that route does not exceed the cost of that route. The Column Generation process is equivalent to iteratively adding constraints to the dual formulation. At each iteration, only a subset of the Constraints (5) (i.e., those corresponding to the routes $R' \in R$) is included in the dual formulation. After solving this restricted problem, which we will call problem D', we verify if any of the excluded constrains are violated by the obtained solution. If this is the case, (some of) the violated constraints are added to the formulation and the process is repeated. Otherwise, the problem has been solved to optimality.

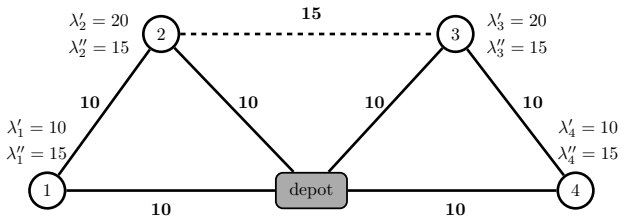

Figure 1: The route ($depot \to 2 \to 3 \to depot$) has reduced costs -5 with $\boldsymbol{\lambda'}$ and 5 with $\boldsymbol{\lambda''}$.

Finding the most violated constraint in the dual formulation requires exploring the exponential space of all possible routes. This can be formulated as a combinatorial optimization problem, called the *pricing* problem:

$$\hat{r} = \arg\min_{r \in R} (c_r - \sum_{i \in r} \hat{\lambda}_i). \tag{7}$$

The degree of violation of the constraint corresponding to route r (i.e. $c_r - \sum_{i \in r} \hat{\lambda}_i$) in the dual formulation is equal to the reduced cost of the column corresponding to route r in the primal formulation. In other words, the pricing problem finds the column with the most negative reduced cost at each iteration of the Column Generation algorithm.

Problem D' can have more than one optimal solution, and the choice of this solution can affect the route obtained by the pricing problem. This, in turn, can significantly influence the speed at which the Column Generation algorithm converges. Consider the example presented in Figure 1, adapted from Rousseau et al. (2007), where $R' = \big\{\{1,2\},\{3,4\}\big\}$. Given the optimal solution $\boldsymbol{\lambda'} = (10, 20, 20, 10)$, the route $\{2,3\}$ has a reduced cost of $\hat{r}' = -5$ and is a candidate to be added to D', even though it is not likely to be selected in the optimal solution of P. However, the alternative optimal solution $\boldsymbol{\lambda''} = (15, 15, 15, 15)$ yields a positive reduced cost of $\hat{r}'' = 5$ for this route and is therefore not added to D'.

2.1 Stabilization

The importance of choosing the right dual solution and its effect on the convergence of the Column Generation algorithm has been widely acknowledged, and different classes of techniques have been developed to address this problem. For a recent discussion on this topic, see Pessoa et al. (2018) and references therein.

Our work is particularly inspired by the stabilization method proposed by Rousseau et al. (2007). The authors directly tackle the issue that the restricted dual problem tends to have several optimal solutions, each of which may distribute differently the marginal costs among the customers. Given that these extreme solutions are sparse by definition, some customers may have large dual values, while others may have zero dual values. This, in turn, may result in a bad estimation of the marginal costs of the customers. Instead of using a set of dual values proposed by one of the optimal solutions to the restricted dual problem, this stabilization method uses an interior point of the convex hull of the extreme solutions to the restricted dual by averaging over a few of them.

In this work, we follow the above intuition. However, instead of averaging over a set of randomly chosen extreme solutions, we aim at learning how to identify the extreme point (represented by its simplex tableau) that provides optimal convergence within the Column Generation procedure.

3 Column Generation by Imitation Learning

In this section, we discuss how to frame the Column Generation procedure as a sequential decision making problem (Puterman, 1994). At each step (i.e., at each iteration), the task is to choose which column to include in the restricted problem. Using a collection of problem instances, we aim at learning a *policy* that dictates which column to pick. This problem is a *Markov Decision Process* (MDP).

An MDP can be defined as a tuple $\langle \mathcal{S}, \mathcal{A}, p(s' \mid s, a), r(s, a), p_0 \rangle$ with state space $\mathcal{S}$, the action space $\mathcal{A}$, the transition probability $p(s' \mid s, a)$, the reward function $(s', s \in \mathcal{S}, a \in \mathcal{A})$, $r(s, a)$, and the initial state distribution p_0.

We represent the state at iteration t of the Column Generation procedure by the pair (z, s_t), where z represents the properties of the CVRP instance (vehicle capacity, demands and locations of customers, etc.) and s_t encodes the restricted problem, i.e., all the information presented in the final simplex tableau in that iteration. The most important components of s_t are the columns (routes), available within the restricted problem, and the dual values (i.e., the optimal solution of the restricted dual problem). The action a_t represents the next column to add to the restricted problem. The transition function $p(s' \mid s, a)$ is deterministic and is evaluated by adding the column corresponding to action a to the restricted problem encoded in s and solving it. Since our goal is to improve the convergence rate, the reward is set to a constant (-1) after each action, therefore encouraging a fast convergence. The process terminates when no more column with a negative reduced cost is found.

A policy $\pi_\theta(a_t \mid s_t, z)$ is a probability distribution over actions, conditioned on the state, and parameterized by a vector θ. Our goal is to learn this distribution, that is, find the value for θ that maximizes the expected reward. We learn this policy by *imitating* an *expert*. This expert is assumed to know how to select columns in a way that leads to fast convergence of the Column Generation algorithm, but is potentially too expensive to be used directly. We record the actions of the expert by solving a collection of N problem instances using the expert policy (discussed below). For every instance i, we record the properties $z^{(i)}$. Moreover, at every iteration t, we record ${s_t}^{(i)}$ and the action $a_t^{*(i)}$ advised by the expert. We then learn the policy by minimizing the cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \sum_{t=1}^{T_i} \log \pi_\theta(a_t^{*(i)} \mid {s_t}^{(i)}, z^{(i)}). \tag{8}$$

Minimizing this loss function is equivalent to maximizing the likelihood of the expert actions.

3.1 Expert Policy

Our proposed method requires an expert that can choose the right columns to make the Column Generation procedure converge quickly. We will now present such an expert policy. Recall that at each iteration, the vector of dual values $\boldsymbol{\lambda'}$ (the optimal solution of D' obtained by the simplex algorithm) is an *estimate* of $\boldsymbol{\lambda}^*$, the optimal solution of D. Motivated by this observation, from the space of optimal solutions of D', we pick the $\boldsymbol{\lambda'}$ that is closest to $\boldsymbol{\lambda}^*$, i.e., the one with the smallest $\| \boldsymbol{\lambda'} - \boldsymbol{\lambda}^* \|_2^2$. The problem of finding $\boldsymbol{\lambda'}$ can be formulated as a quadratic optimization problem:

$$\min_{\boldsymbol{\lambda'}} \quad \| \boldsymbol{\lambda'} - \boldsymbol{\lambda}^* \|_2^2 \tag{9}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} \boldsymbol{\lambda}'_i = \Lambda \tag{10}$$

$$\sum_{i=1}^{N} \boldsymbol{\lambda}'_i \boldsymbol{e}_{ir} \leq c_i \quad \forall r \in R' \tag{11}$$

$$\boldsymbol{\lambda'} \geq 0 \tag{12}$$

where Λ is the optimal objective value of D', and constraint (10) enforces the optimality of the solution. Note that we do not use $\boldsymbol{\lambda}^*$ since it is not necessarily an optimal solution of D' at every iteration.

After solving this problem, we provide these *adjusted* duals to the pricer, and obtain a route a^* which is used as the expert action at this state. Note that the expert requires access to $\boldsymbol{\lambda}^*$, which means that we first need to solve the instance to optimality. After solving a collection of instances offline and collecting the expert actions, we can train the policy function $\pi_\theta(a_t \mid s_t, z)$ which is parametrized as a deep neural network.

3.2 Policy Network

The policy function takes the state (i.e., a CVRP instance and the restricted problem) as input, and returns a distribution over all columns (i.e., all feasible routes). Expressing this function as a deep network raises a number of challenges. First, this distribution is defined over an exponential number of columns which can be enumerated only for small instances. We should be able to use this distribution without the need to explicitly represent it. Second, the size of the state varies by instance and iteration. Hence this function should accept variable-sized inputs. Finally, this function should be aware of certain symmetries in the input. For example, permuting the columns of the simplex tableau should not change the output, and after swapping two constraints in the tableau the corresponding output probabilities should also swap.

3.2.1 A Distribution over Actions

We address the first challenge by learning a mapping from s_t and z to a vector of adjusted duals that is optimal w.r.t. the restricted dual corresponding to s_t. After learning this mapping, we use these adjusted duals in the Column Generation procedure. At each iteration, we provide s_t and z to this mapping and obtain the adjusted duals. The pricer then takes the adjusted duals as input and returns a column. The procedure terminates when the reduced cost of the returned column is non-negative. Note that the optimality of adjusted duals is essential for the soundness of this approach.

Let us assume for now that there is a function f_θ that maps s_t and z to the optimal duals $\boldsymbol{\lambda}$. We can turn this function into a distribution over all columns (and the termination action) and train it using the loss function of Equation (8). Since the number of routes grow exponentially, we train the model using small instances where the columns can be enumerated. Once we have trained the model, we use a pricer to find the best column, enabling us to apply it to larger instances.

For an instance with n columns, let us represent column i and its cost by vector $\boldsymbol{e}_i$ and scalar c_i, respectively. Given the adjusted duals $\boldsymbol{\lambda}$, the reduced cost of column i is $c_i - \boldsymbol{\lambda}^T \boldsymbol{e}_i$. We denote the action of choosing column i by $a_i\,(i = 1, \ldots, n)$, and choosing no column (i.e., terminating the procedure) by a_0. Our goal is to define a probability distribution over $a_0, a_1, \ldots, a_n$ that depends on $\boldsymbol{\lambda}$ and gives higher probability to columns with smaller reduced cost. More formally, we require that for every pair of columns i and j, $P(a_i) \geq P(a_j)$ iff $c_i - \boldsymbol{\lambda}^T \boldsymbol{e}_i \leq c_j - \boldsymbol{\lambda}^T \boldsymbol{e}_j$. Moreover, the action a_0 should have the highest probability only if there is no column with a negative reduced cost. We enforce this by requiring that for every column i, $P(a_0) \geq P(a_i)$ iff $c_i - \boldsymbol{\lambda}^T \boldsymbol{e}_i \geq 0$. These requirements are met by the following distribution:

$$P(a_0) = \frac{1}{1 + \sum_{j=1}^{n} \exp(\boldsymbol{\lambda}^T \boldsymbol{e}_j - c_j)}, \tag{13}$$

$$P(a_i) = \frac{\exp(\boldsymbol{\lambda}^T \boldsymbol{e}_i - c_i)}{1 + \sum_{j=1}^{n} \exp(\boldsymbol{\lambda}^T \boldsymbol{e}_j - c_j)}. \tag{14}$$

This distribution is obtained by applying the softmax function to the vector $(0, \boldsymbol{\lambda}^T \boldsymbol{e}_1 - c_1, \ldots, \boldsymbol{\lambda}^T \boldsymbol{e}_n - c_n)$. Figure 2 depicts the mapping from s_t and z to this distribution.

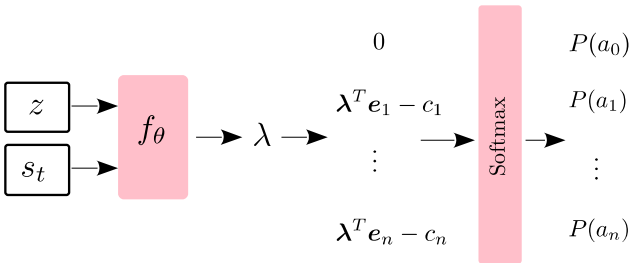

Figure 2: The deep network f_θ maps state (z, s_t) to a vector of optimal duals $\boldsymbol{\lambda}$ which is then used to define policy $\pi_\theta(a_t \mid s_t, z)$.

Next, we will describe the architecture of f_θ, the deep network that maps s_t and z to the optimal duals $\boldsymbol{\lambda}$.

3.2.2 Representing the Restricted Problem

We represent state s_t as a bipartite graph $G = (\mathcal{C}, \mathcal{V}, \mathcal{E})$, where $\mathcal{C}$ corresponds to constraints (customers), $\mathcal{V}$ corresponds to variables/columns (routes), and $(c, v) \in E$ iff variable v has a nonzero coefficient in constraint c. Other state information correspond either to variables (e.g. costs, reduced costs) or constraints (e.g. slacks, dual values) and can be represented as features of the nodes in $\mathcal{V}$ and $\mathcal{C}$.

After encoding s_t as a bipartite graph $G = (\mathcal{C}, \mathcal{V}, \mathcal{E})$ with node features $c_i \in \mathbb{R}^{dc}, i \in \{1, \dots, |\mathcal{C}|\}$ and $v_i \in \mathbb{R}^{d_\mathcal{V}}, i \in \{1, \dots, |\mathcal{V}|\}$, we can apply the graph convolutional layers on this graph and obtain the node embeddings $c'_i \in \mathbb{R}^{d'_\mathcal{C}}, i \in \{1, \dots, |\mathcal{C}|\}$ and $v_i \in \mathbb{R}^{d'_\mathcal{V}}, i \in \{1, \dots, |\mathcal{V}|\}$. We use a graph convolutional layer similar to the one introduced by Gasse et al. (2019) to process this bipartite graph:

$$c'_i = f_\mathcal{C}\big(c_i \parallel \sum_{j:(i,j)\in E} g_\mathcal{C}(c_i \parallel v_j)\big), \tag{15}$$

$$v'_i = f_\mathcal{V}\big(v_j \parallel \sum_{i:(i,j)\in E} g_\mathcal{V}(c'_i \parallel v_j)\big), \tag{16}$$

where $\parallel$ is the concatenation operator and $f_\mathcal{C}$, $f_\mathcal{V}$, $g_\mathcal{C}$ and $g_\mathcal{V}$ are 2-layer perceptrons with Rectified Linear Unit (ReLU) activation functions.

Representing this part of the state as a graph enables us to process states with different sizes by the same model. Moreover, the model maintains the desired permutation invariance and equivariance properties.

3.2.3 Representing the Instance Properties

We include the instance properties z in the model by adding extra information to the constraint embeddings c'_i. Let us represent the properties of customer i (e.g. location and demand) by the vector $\hat{c}_i$. After concatenating c'_i and $\hat{c}_i$, we obtain the feature vector c''_i, which may also include global information (e.g. the vehicle capacity). Given that this information is represented as a set (i.e., the order of customers is irrelevant), it must be encoded using a representation that allows to preserve this property. We here apply set encoder layers (Vaswani et al., 2017) to the set of constraint features and obtain a vector $\boldsymbol{q}$ with size $|\mathcal{C}|$. Self-Attention is a mechanism that maintains permutation equivariance, and has been used in models that operate on sets (Lee et al., 2019) or graphs (Velickovic et al., 2018). An attention layer takes n elements $x_i \in \mathbb{R}^{d_x}, i \in \{1, \dots, n\}$ and outputs $z_i \in \mathbb{R}^{d_z}, i \in \{1, \dots, n\}$, where each output is a weighted sum of linear transformations of the input elements:

$$z_i = \sum_{j=1}^{n} \alpha_{ij}(x_j W^V) \tag{17}$$

where the weights α_{ij} are computed in terms of compatibility scores e_{ij}:

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}} \tag{18}$$

and the compatibility scores compare two input elements using a scaled dot product function:

$$e_{ij} = \frac{(x_i W^Q)(x_j W^K)^T}{\sqrt{d_z}}. \tag{19}$$

Parameters $W^Q, W^K, W^V \in \mathbb{R}^{d_x \times d_z}$ are learnable. Similar to Transformers, we create encoder layers which consist of a self-attention and a position-wise feedforward layer, where each of these sublayers is accompanied by residual connections and is followed by layer normalization.

3.2.4 Generating the Optimal Duals

Finally, we need to map the vector $\boldsymbol{q}$ to a vector of dual values $\boldsymbol{\lambda}$ that is optimal w.r.t. the restricted dual. The space of optimal duals can be represented by a set of linear constraints consisting of the restricted dual and an optimality constraint, similar to constraints (10)-(12).

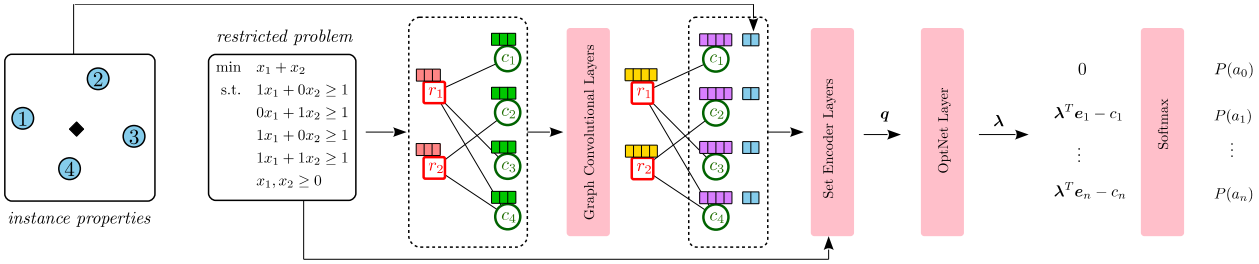

Figure 3: The architecture of policy network in neural Column Generation. The nodes r_i and c_j correspond to the routes and customers (i.e. the columns and rows in the restricted problem), respectively.

To this end, we define $\frac{1}{2}\boldsymbol{\lambda}^T \boldsymbol{Q}\boldsymbol{\lambda} + \boldsymbol{q}^T\boldsymbol{\lambda}$ as the objective function over the space of optimal duals, where $\boldsymbol{Q} = 0.001\boldsymbol{I}$. By solving the resulting optimization problem, we obtain a vector of duals which is a function of vector $\boldsymbol{q}$, and also optimal w.r.t. the restricted dual problem.

This optimization problem is differentiable w.r.t. vector $\boldsymbol{q}$ and can be included as a layer in a deep network. *Differentiable optimization* layers are computational units that not only solve an optimization problem, but also calculate the gradient of its solution with respect to the problem parameters. Such layers make it possible to include an optimization step as part of a deep learning pipeline. We use a modified version of the OptNet architecture (Amos and Kolter, 2017). OptNet solves Quadratic Programming (QP) problems of the form:

$$\min_z \quad \frac{1}{2} z^T Q z + q^T z \tag{20}$$

$$\text{s.t.} \quad Az = b \tag{21}$$

$$Gz \leq h \tag{22}$$

Assume that in the backward pass of the backpropagation algorithm, we receive the vector $\frac{\partial \ell}{\partial z^*} \in \mathbb{R}^n$, and z^*, ν^* and λ^* are the optimal primal and dual variables. Moreover, let us denote by $D(x)$ the diagonal matrix created from the vector x. OptNet first calculates:

$$\begin{bmatrix} d_z \\ d_\lambda \\ d_\nu \end{bmatrix} = - \begin{bmatrix} Q & G^T D(\lambda^*) & A^T \\ G & D(Gz^* - h) & 0 \\ A & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} (\frac{\partial \ell}{\partial z^*})^T \\ 0 \\ 0 \end{bmatrix}, \tag{23}$$

using which then the gradients with respect to all problem parameters are calculated. In this work, we will use the gradients with respect to the coefficients of the objective function (i.e., Q and q):

$$\nabla_Q \ell = \frac{1}{2}(d_z z^T + z d_z^T) \qquad \nabla_q \ell = d_z \tag{24}$$

The coefficient matrices G in our examples are not always full-rank, and this makes it impossible to perform the matrix inversion in Equation (23). Hence, we replace the inversion operation with pseudo-inversion:

$$\begin{bmatrix} d_z \\ d_\lambda \\ d_\nu \end{bmatrix} = - \begin{bmatrix} Q & G^T D(\lambda^*) & A^T \\ G & D(Gz^* - h) & 0 \\ A & 0 & 0 \end{bmatrix}^{+} \begin{bmatrix} (\frac{\partial \ell}{\partial z^*})^T \\ 0 \\ 0 \end{bmatrix} \tag{25}$$

Figure 3 summarizes the entire architecture of the policy network.

4 Computational Experiments

We now explore the effectiveness of COIL by performing an empirical evaluation. Specifically, we investigate four research questions: **Q1** To what extent do the learned models improve the convergence of the column generation algorithm?, **Q2** How do different architectures compare in terms of loss and accuracy?, **Q3** To

what extent does the removal of the differentiable optimization layer affect the performance of the learned models?, and **Q4** Do the learned models generalize to instances larger than those seen during training?

We train and evaluate our approach using randomly generated CVRP instances. We generate the instances using the method from Uchoa et al. (2017) with 21 customers, a *central* depot, *clustered* customer positioning, and a value of 6 for the r parameter (the desired average number of customers in a route). To train our algorithm, we require all valid routes for an instance. We therefore generate all valid routes for a given problem instance based on a dynamic programming algorithm for the Traveling Salesman Problem. The pseudocode of this procedure is presented in Algorithm 1.

Algorithm 1: Generating all routes for CVRP

function GenerateRoutes(n, *distances*, *demands*, *capacity*)
  for $i \leftarrow 1, \ldots, n$ **do**
    $path\_lenght[\{i\}, i] \leftarrow distances[0, i]$
    $Q.push[\{i\}]$
    $routes \leftarrow routes \cup \{i\}$
  while $!Q.empty()$ **do**
    $path \leftarrow Q.pop()$
    $d \leftarrow \sum_{i \in path} demands[i]$
    for $i \leftarrow \max(path) + 1 \ldots n$ **do**
      if $demands[i] + d \leq capacity$ **then**
        $next\_path \leftarrow path \setminus \{i\}$
        $Q.push(next\_path)$
        $routes \leftarrow routes \cup \{next\_path\}$
        for $u \in next\_path$ **do**
          $prefix \leftarrow next\_path \setminus \{u\}$
          $path\_length[next\_path, u] \leftarrow \min_{v \in prefix} (path\_length[prefix, v] + distances[v, u])$
  for $path \in routes$ **do**
    $route\_lengths[path] \leftarrow \min_{v \in path} path\_lengths[path, v] + distances[v, 0])$
  return *route_lenghts*

We solved 300 training and 300 validation CVRP problem instances using the expert policy described in Section 3.1. Each training/validation instance corresponds to one iteration executed according to the expert policy. For testing, we solve 1,652 CVRP instances using different policies and record the number of iterations. Since the stabilization method of Rousseau et al. (2007) is randomized and the number of iterations can vary significantly, for this method, we compute the average over 20 runs.

We train all models using Adam optimizer (Kingma and Ba, 2015) with an initial learning rate of 3e-4 and minibatches of size 32. After 10 epochs with no validation error improvement, the rate is divided by 5, and after 20 such epochs the training stops. The runtime of the overall training processes is limited to 10 hours. We train the models using a single thread of CPU using machines with Intel 6148 2.4 GHz processors with a memory limit of 32 GB. The code and data is publicly available online[1].

To answer **Q1**, we compare the convergence of several models with the Interior-point Stabilization (**IPS**) method Rousseau et al. (2007). We use the following methods for learning the linear coefficients q:

- **Baseline**: learning a fixed q for all training instances, discarding the state and context information.
- **MLP**: a model with a row-wise two-layer feed-forward network, applied independently to each customer i with q_i as the output.
- **COIL-S**: a set encoder with customer information as input and q as output.

[1] https://github.com/Behrouz-Babaki/COIL

- **COIL-G**: a graph neural network with state information as input, and q as output.
- **COIL-GS**: using both graph convolutional and set encoder layers, as depicted in Figure 3.

For the route nodes (used in **COIL-G** and **COIL-GS**), we used a single feature, namely the cost of the route. Initially, the customer nodes have a single feature in the **COIL-GS** model, which is the original dual value given by the simplex tableau. After receiving the embeddings of customer nodes from the graph convolutional layers, they are concatenated with the demands and locations of customers, and the vehicle capacity (appended to the feature vectors of all customers). In models **Baseline**, **MLP**, **COIL-S** and **COIL-G**, all customer features are provided in one feature vector.

	IPS	*Baseline*	*MLP*	*COIL-S*	*COIL-G*	*COIL-GS*
#Wins	1	278	296	248	436	**695**
Ratio	1.581	1.239	1.211	1.232	1.17	**1.12**

Table 1: Comparing different policies in terms of number of wins and the average ratio of number of iterations w.r.t. the expert policy.

Table 1 compares the convergence of different methods on the set of 1,652 test instances. The first row (# Wins) shows the number of times that each method has the smallest number of iterations. Moreover, for each instance and method, we divide the number of iterations by the number of iterations of the expert policy. The second row shows the averages of these ratios over all test instances. The results indicate that all learned models outperform **IPS** and that using a GCN boosts performance. **COIL-GS** shows superior performance and requires only 12% more iterations on average than the expert policy, while **IPS** requires 58% more iterations on average.

	Loss	*Top-k Accuracy*			
		1	*10*	*100*	*1000*
Baseline	42.37	0.445	0.773	0.933	0.98
MLP	36.134	0.459	0.8	0.947	0.987
COIL-S	34.609	0.461	0.809	0.949	0.988
COIL-G	29.961	0.499	0.836	0.958	0.989
COIL-GS	**25.134**	**0.537**	**0.861**	**0.966**	**0.990**

Table 2: Validation cross-entropy and top-k of the learning models. COIL-GS achieves the best performance consistently.

To answer **Q2**, in Table 2 we present the cross-entropy loss of different models on the validation set. We also present the top-k accuracy for different values of k, which is the percentage of times that the target column appears in the k columns with the highest probability in the distribution generated by each model. The learning curves of different models are presented in Figure 4. These results clearly indicate the advantage of model **COIL-GS** over the alternatives.

We answer **Q3** by creating models which directly predict the adjusted duals used by the expert policy. Both of these models have architectures similar to **COIL-GS**, except that the layers after set encoders are removed. In the **MSE** model, the output of set encoders is used as predicted adjusted duals. This model is trained by minimizing the mean squared error (MSE) loss between the predicted and target adjusted duals. Note that the dual vector predicted by this model is not necessarily optimal. We address this problem in the **KLD** model, which learns to distribute the optimal objective among the customers. This distribution is obtained by applying the softmax function to the output of set encoders. The model is trained by minimizing the KL-divergence (KLD) between the predicted distribution and the actual distribution of total value among the target adjusted duals.

Table 3 shows the cross-entropy loss and top-k accuracy for these two models. Comparing these values with those in Table 2 shows that including the OptNet layer is essential for obtaining a reasonable performance.

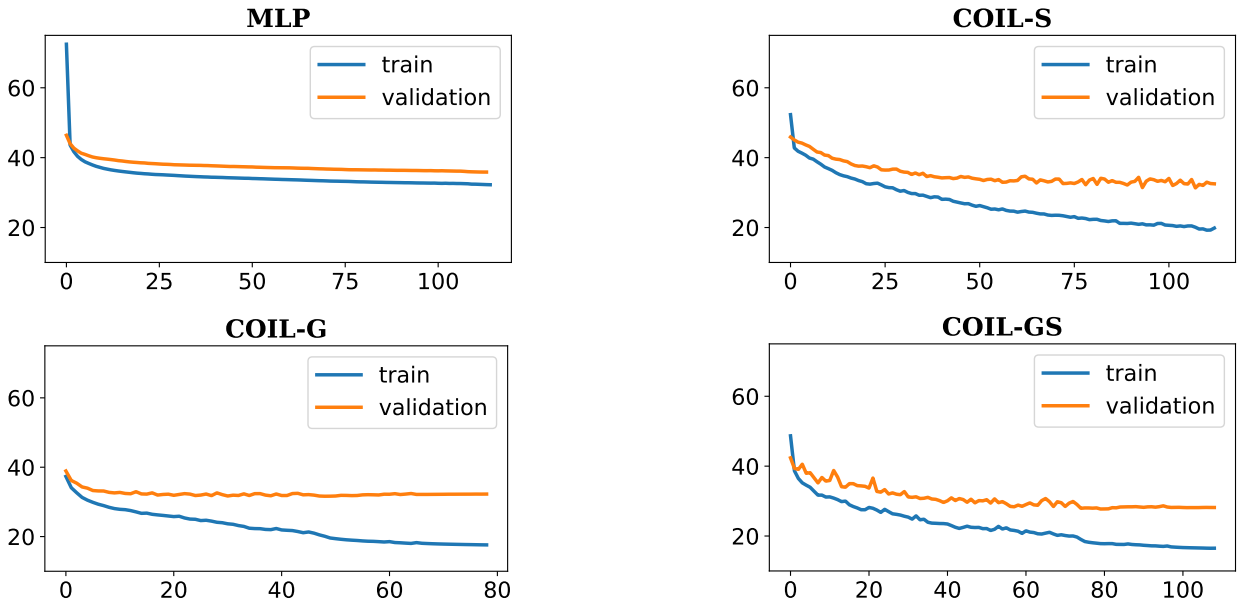

Figure 4: Learning curves for different architectures.

	Loss	*Top-k Accuracy*			
		1	*10*	*100*	*1000*
MSE	168.313	0.047	0.198	0.535	0.871
KLD	164.295	0.062	0.236	0.586	0.889

Table 3: The effect of removing the differentiable optimization layer on cross-entropy loss and top-k accuracy.

Finally, our approach is motivated by the assumption that we can learn from small instances to solve larger ones. This requires generalization to instances larger than those seen during training. We answer **Q4** by comparing **COIL-GS** on two sets of medium-size CVRP instances. The first set consists of 100 instances generated using the same mechanism as training instances, except that the number of customers is 50, and the r parameter is set to 5. The second set includes the datasets A, B, E, and P from the CVRP benchmark library *CVRPLIB*.[2] Since the enumeration of all columns is not feasible for the medium-size instances, we implemented a pricer using the algorithm of Feillet et al. (2004) in C++.

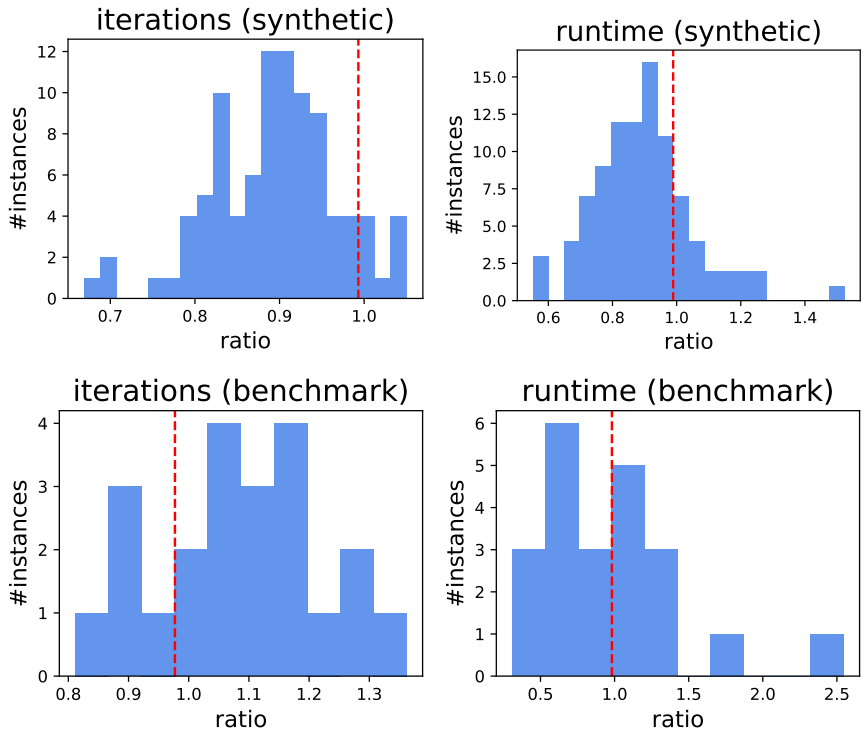

Figure 5: Ratio of the performance of *COIL-GS* over *IPS* in terms of number of iterations and runtime using larger instances.

We compare **IPS** and **COIL-GS** by taking the ratio of number of iterations when using **IPS** to that of **COIL-GS**. We also calculate similar ratios for runtime. Figure 5 shows histograms of these ratios for synthetic and benchmark instances. The instances with a ratio less than one (i.e. those on the left side of the dashed line) are those where **COIL-GS** outperforms **IPS**. These experiments demonstrate that when the larger instances are generated by a distribution similar to the training instances, the learned model generalizes well. The difference between the performance of the learned model on synthetic and benchmark instances might be a result of the variety of instances in the latter. It is worth noting that the learned model always outperforms the unstabilized column generation in all medium-size instances.

5 Conclusion and Future Work

In this paper, we propose and explore several deep-learning architectures for improving the convergence of the column generation algorithm. The empirical evaluation demonstrates the advantage of encoding the instance information (using a set encoder), the routes currently included in the problem (using a bipartite graph convolution layer), and end-to-end learning (using a differentiable optimization layer).

Although column generation is inherently solving a sequential decision making problem, the existing learning-based methods reduce it to simpler problems, such as classification (Morabit et al., 2021). The

[2] http://vrp.atd-lab.inf.puc-rio.br/

novelty of our approach is that we directly aim at solving the sequential decision making problem encountered in column generation. We present a novel approach for dealing with the exponential action space, by reducing the problem to that of choosing the best dual value in the space of optimal ones. This introduces the challenge of embedding an optimization problem in the learned model, which we address using the OptNet layers.

Our research opens up several research directions. Despite the advantages of the OptNet layer, it also turns into a bottleneck and limits the amount of training data that can be processed. First, it has to solve a QP for every instance at every forward pass. Second, our implementation of OptNet can only use one CPU thread, which severely harms the scalability of our approach. An interesting direction for future work is removing the OptNet layer and train networks with larger capacity using more data.

The convergence of a column generation algorithm depends not only on the number of iterations, but also on the time taken for solving the pricing problem at each iteration. Incorporating the latter is another avenue for future research.

The presented neural column generation can generally be applied to other problem classes than the CVRP here considered, which opens up several future research directions. As we have used some problem-specific information as features, our general framework would benefit from a more generic method for representing the instance properties.

Acknowledgments

We are thankful to Louis-Martin Rousseau, Maxime Gasse, Prateek Gupta, Seyed Mehran Kazemi, Thibaut Vidal and Giulia Zarpellon for enlightening discussions. This research was partially supported by IVADO, FRQNT, NSERC, the CIFAR AI Chairs program, Calcul Québec and Compute Canada.

References

Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 2017.

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

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

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.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *NeurIPS*, pages 15554–15566, 2019.

Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *NIPS*, pages 6348–6358, 2017.

Elias Boutros Khalil, Pierre Le Bodic, Le Song, George L. Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *AAAI*, pages 724–731. AAAI Press, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR, 2019.

Mouad Morabit, Guy Desaulniers, and Andrea Lodi. Machine-learning-based column selection for column generation. *Transp. Sci.*, 55(4):815–831, 2021.

MohammadReza Nazari, Afshin Oroojlooy, Lawrence V. Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *NeurIPS*, pages 9861–9871, 2018.

Artur Alves Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS J. Comput.*, 30(2):339–360, 2018.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 1994.

Louis-Martin Rousseau, Michel Gendreau, and Dominique Feillet. Interior point stabilization for column generation. *Oper. Res. Lett.*, 35(5):660–668, 2007. doi: 10.1016/j.orl.2006.11.004. URL https://doi.org/10.1016/j.orl.2006.11.004.

Yunzhuang Shen, Yuan Sun, Xiaodong Li, Andrew S. Eberhard, and Andreas T. Ernst. Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring. *CoRR*, abs/2112.04906, 2021.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 9367–9376. PMLR, 2020.

Paolo Toth and Daniele Vigo, editors. *Vehicle Routing*, volume 18 of *MOS-SIAM Series on Optimization*. SIAM, 2014.

Eduardo Uchoa, Diego Pecin, Artur Alves Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *Eur. J. Oper. Res.*, 257(3):845–858, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pages 5998–6008, 2017.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR (Poster)*. OpenReview.net, 2018.