A Reinforcement-Learning-Based Multiple-Column Selection Strategy for Column Generation

Haofeng Yuan, Lichang Fang, Shiji Song*

Department of Automation, BNRist, Tsinghua University
{yhf22, fanglc22}@mails.tsinghua.edu.cn, shijis@mail.tsinghua.edu.cn

Abstract

Column generation (CG) is one of the most successful approaches for solving large-scale linear programming (LP) problems. Given an LP with a prohibitively large number of variables (i.e., columns), the idea of CG is to explicitly consider only a subset of columns and iteratively add potential columns to improve the objective value. While adding the column with the most negative reduced cost can guarantee the convergence of CG, it has been shown that adding multiple columns per iteration rather than a single column can lead to faster convergence. However, it remains a challenge to design a multiple-column selection strategy to select the most promising columns from a large number of candidate columns. In this paper, we propose a novel reinforcement-learning-based (RL) multiple-column selection strategy. To the best of our knowledge, it is the first RL-based multiple-column selection strategy for CG. The effectiveness of our approach is evaluated on two sets of problems: the cutting stock problem and the graph coloring problem. Compared to several widely used single-column and multiple-column selection strategies, our RL-based multiple-column selection strategy leads to faster convergence and achieves remarkable reductions in the number of CG iterations and runtime.

Introduction

Column generation (CG) is a widely used approach for solving the linear programming (LP) relaxations of large-scale optimization problems that have a prohibitively large number of variables to deal with. It exploits the fact that the majority of feasible variables (i.e., columns) will not be part of an optimal solution. Therefore, CG starts with a subset of columns and gradually adds new columns that have the potential to improve the current solution, e.g., columns with a negative reduced cost (assuming a minimization problem), until no such columns exist and the current solution is proven optimal (Desaulniers, Desrosiers, and Solomon 2006). CG is often combined with the branch-and-bound method to solve large-scale integer programming problems, which is called branch-and-price (Barnhart et al. 1998).

Specifically, CG follows an iterative process, as shown in Figure 1. The original large-scale problem is decomposed into the restricted master problem (RMP) and the pricing problem (PP). CG starts by solving the RMP with a small subset of columns from the original problem. At each iteration, the RMP is solved using an LP solver (e.g., the simplex algorithm), and the dual solution is used to formulate the PP. The PP is a "column generator" that generates new columns (typically with negative reduced costs) to improve the current RMP solution. If such columns are found, they are added to the RMP to start a new iteration. Otherwise, it certifies the optimality of the current RMP solution for the original problem, and CG terminates.

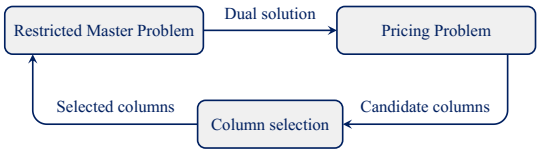


Figure 1: The iterative process of CG.

Typically, the column with the most negative reduced cost is selected to add to RMP at each iteration, which guarantees the convergence of CG and the optimality of the final solution (Lübbecke and Desrosiers 2005). However, it often suffers from slow convergence, which limits its efficiency and usability. Previous research has shown that selecting multiple columns per iteration, including sub-optimal solutions of PP (even columns with non-negative reduced costs), can lead to faster convergence (Moungla, Létocart, and Nagih 2010). This allows the RMP approximation to be improved, the optimal basis to be characterized faster, and hence to reduce the number of iterations. However, selecting multiple columns per iteration may result in a large fraction of useless columns that do not belong to the final optimal basis and increase the computation cost of RMP.

In general, the PP can generate a pool of feasible candidate columns, which are sorted according to the reduced cost, and the top-k of them are greedily selected. In order to improve the selection, Goffin and Vial (2000) suggested that the RMP description can be improved by selecting non-correlated columns. Several diversification-based multiple-column selection strategies have been developed and shown to be effective in practice (Vanderbeck 1994; Moungla, Létocart, and Nagih 2010). However, despite the

*Corresponding author.

practical effectiveness of the diversification-based selection strategies, there is still no perfect column selection strategy proven to outperform or dominate the others.

Recently, reinforcement learning (RL) has shown impressive success in optimization tasks (Mazyavkina et al. 2021; Yang, Jiang, and Song 2023; Zhang et al. 2020), which removes the need for substantial expert knowledge and presolved instances. In this paper, we propose a novel RL-based multiple-column selection strategy for CG. Specifically, we treat the iterative column selection in CG as a sequential decision task, and introduce an actor-critic style neural network that takes into account the column-constraint structure of RMP, the interrelations of candidate columns, and global properties of the problem instance. We use proximal policy optimization (PPO) (Schulman et al. 2017) to train the strategy to minimize the total number of iterations. Our RL-based multiple-column selection strategy is evaluated on two sets of problems: the cutting stock problem (CSP) (Gilmore and Gomory 1961) and the graph coloring problem (GCP) (Mehrotra and Trick 1996). Experimental results demonstrate that our RL-based strategy outperforms several widely used single-column and multiple-column selection strategies in terms of the number of iterations and runtime. The main contributions of this paper can be summarized as follows:

- We exploit RL to learn an effective multiple-column selection strategy for CG. To the best of our knowledge, it is the first RL-based multiple-column selection strategy.
- We design an actor-critic style neural network that considers the column-constraint structure of RMP, the interrelations of candidate columns, and global properties of the problem instance, which allows to learn a column-relation-aware multiple-column selection strategy.
- We apply our approach to CSP and GCP, and experimental results show that it outperforms all baseline column selection strategies on various sizes of problems. Moreover, our RL-based framework can be easily applied to other problems solved based on CG.

Related Work

In this section, we review the acceleration methods for CG in the literature, with a focus on recent advances in machine learning (ML) techniques for column selection.

Acceleration Methods for Column Generation. Various techniques have been proposed in the literature to accelerate CG (Desaulniers, Desrosiers, and Solomon 2002). One approach is to select "better" columns to add to RMP at each iteration. A classic approach is to add multiple columns rather than a single column with the most negative reduced cost. Goffin and Vial (2000) showed that the performance of CG is mathematically related to the variance-covariance matrix of selected columns: the convergence is accelerated with the selection of non-correlated columns. Moungla, Létocart, and Nagih (2010) proposed two practical multiple-column selection strategies, which enhance the diversification of selected columns. Nevertheless, the effect of column selection is still not fully understood theoretically, and there is still no perfect column selection strategy proven to achieve the minimum number of iterations for CG. For faster convergence than existing hand-crafted column selection strategies, we apply RL to learn a multiple-column selection strategy that aims at minimizing the number of iterations through the interaction with the CG solution process.

Another approach is dual stabilization, which aims to form a "better" PP. For example, du Merle et al. (1999) introduced a penalty function to reduce the oscillation of dual values. For a discussion on stabilization-based acceleration methods, please see (Pessoa et al. 2018) and the references therein. More recently, Babaki, Jena, and Charlin (2021) proposed a learning-based method for predicting the optimal stabilization center of dual values in vehicle routing problems. We remark that our column selection strategy does not conflict with dual stabilization techniques and can be used synergistically for further improvement.

Machine-Learning-based Column Selection Strategy. Over the last few years, researchers have become increasingly interested in ML to accelerate optimization tasks (Bengio, Lodi, and Prouvost 2021), and several learning-based methods have been proposed for specific problems solved by CG (Tahir et al. 2021; Yuan, Jiang, and Song 2022; Shen et al. 2022). The closest works to ours are (Morabit, Desaulniers, and Lodi 2021) and (Chi et al. 2022), both leveraging ML for a better column selection strategy.

Morabit, Desaulniers, and Lodi (2021) proposed a one-step lookahead "expert" to identify the columns that maximize the improvement of RMP in the next iteration, which is achieved by solving an extremely time-consuming mixed-integer linear programming (MILP). Then, they trained an ML model to cheaply imitate the decisions of the expensive MILP expert. They formulated the column selection procedure at each iteration as a classification task and trained the ML model in a supervised manner. The drawback of their approach is that the one-step lookahead expert is shortsighted because it only focuses on the very next iteration but disregards the interdependencies across iterations. Besides, it requires expensive pre-solved instances from previous executions of the MILP expert as training data, which may be unaffordable for large-scale applications. Moreover, the ML model only imitates the decision from the MILP expert, and thus it can never surpass the decisions for demonstration.

Chi et al. (2022) proposed a DQN-based single-column selection strategy that applies Q-learning to identify the "best" column at each iteration. They utilize the graph neural network (GNN) as a Q-function approximator to maximize the total expected future reward. While showing improved performance compared to the greedy single-column selection strategy, their framework can hardly be extended to multiple-column selection due to the exponential growth of action space and complex interdependencies in the column combinations, which limits its practical application (we implement a multiple-column variant of their DQN-based approach in our experiments). In contrast, our proposed neural network and learning scheme overcome these challenges and can derive an effective multiple-column selection strategy. Experiment results show that our RL-based multiple-column selection strategy outperforms the MILP expert used in (Morabit, Desaulniers, and Lodi 2021) and the DQN-

based strategy proposed in (Chi et al. 2022).

Basis of Column Generation

In this section, we use CSP as an example to introduce the mathematical formulation of CG. The CSP aims to determine the smallest number of rolls of length L that have to be cut to satisfy the demands of m customers, where customer i demands d_i pieces of orders of length $\ell_i, i = 1, 2, ..., m$. Gilmore and Gomory (1961) proposed the CG formulation, in which the set $\mathcal{P}$ of all feasible cutting patterns is:

$$\mathcal{P} = \left\{ \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix} \in \mathbb{N}^m \mid \sum_{i=1}^{m} \ell_i a_i \leq L \right\}.$$

Each pattern $p \in \mathcal{P}$ is denoted by a vector $(a_{1p}, \ldots, a_{mp})^{\mathrm{T}} \in \mathbb{N}^m$, where a_{ip} represents the number of pieces of length ℓ_i obtained in cutting pattern p. Let λ_p be a decision variable that denotes the number of rolls cut using pattern $p \in \mathcal{P}$. The CSP is formulated as follows:

$$\begin{aligned} \min \ & \sum_{p \in \mathcal{P}} \lambda_p \\ \text{s.t.} \ & \sum_{p \in \mathcal{P}} a_{ip} \lambda_p \geq d_i, \ i \in \{1, 2, ..., m\}, \\ & \lambda_p \in \mathbb{N}, \ p \in \mathcal{P}. \end{aligned}$$

The objective function minimizes the total number of patterns used, equivalent to minimizing the number of rolls used. The m constraints ensure all demands are satisfied.

This formulation usually has an extremely large number of decision variables as $\mathcal{P}$ is exponentially large. Therefore, the RMP is proposed for the linear relaxation with an initial set $\tilde{\mathcal{P}} \subset \mathcal{P}$. The RMP is defined as follows:

$$\begin{aligned} \min \ & \sum_{p \in \tilde{\mathcal{P}}} \lambda_p \\ \text{s.t.} \ & \sum_{p \in \tilde{\mathcal{P}}} a_{ip} \lambda_p \geq d_i, \ i \in \{1, 2, ..., m\}, \\ & \lambda_p \geq 0, \ p \in \tilde{\mathcal{P}}. \end{aligned}$$

Let $u = (u_1, \ldots, u_m)^{\mathrm{T}}$ be the dual solution of the RMP. The columns that can potentially improve the solution of RMP are given by the solution to the following knapsack problem, which is referred to as the PP:

$$\begin{aligned} \max \ & \sum_{i=1}^{m} u_i a_i \\ \text{s.t.} \ & \sum_{i=1}^{m} \ell_i a_i \leq L, \ i \in \{1, 2, ..., m\}, \\ & a_i \in \mathbb{N}, \ i \in \{1, 2, ..., m\}. \end{aligned}$$

The PP generates feasible patterns (columns), represented as vector $(a_{1p}, \ldots, a_{mp})^{\mathrm{T}}$, to be added to $\tilde{\mathcal{P}}$ for the next iteration. In general, several sub-optimal solutions of PP, which form a candidate column pool, can be obtained through dynamic programming methods or commercial solvers such as *Gurobi* (Gurobi Optimization, LLC 2023). We can select one or multiple columns from the candidate column pool to add to RMP for the next iteration (see Figure 1).

Methodology

In this section, we present the details of our RL-based multiple-column selection strategy.

MDP Formulations

We treat the CG solution process as the environment for the RL agent. We formulate the multiple-column selection task for CG as a Markov decision process (MDP):

State $\mathcal{S}$. The state describes the information about the current CG status. which is provided for the RL agent. As illustrated in Figure 2, the state is defined to include 1) a bipartite graph representation of the current RMP and candidate columns, and 2) global properties of the problem instance.

As introduced in (Gasse et al. 2019), an LP can be represented as a bipartite graph with constraint nodes $\mathcal{C}$ and column nodes $\mathcal{V}$. We further incorporate candidate columns into the bipartite graph representation, where column nodes are divided into existing columns in the current RMP and candidate columns to be selected (blue nodes and red nodes in Figure 2). An edge (v, c) exists between a node $v \in \mathcal{V}$ and a node $c \in \mathcal{C}$ if column v contributes to constraint c. State information corresponding to the columns (e.g., solution value, reduced cost) and constraints (e.g., slack, dual value) are represented as node features. In addition to the bipartite graph representation, we represent the properties associated with the problem instance (e.g., the number of constraints, maximum constraint coefficient) as an additional global feature vector (the green node in Figure 2).

Action $\mathcal{A}$. At each iteration, we select k columns from the pool of n candidate columns generated by the PP, and add them to RMP for the next iteration. The action space $\mathcal{A}$ contains all possible k-combinations of the n candidate columns, i.e., $|\mathcal{A}| = \binom{n}{k}$. The RL agent returns a probability distribution over the action space, and we sample an action from that distribution, which can be seen as a multiple-column selection strategy.

Transition $\mathcal{T}$. The transition rule is deterministic. Once an action is selected, the corresponding k columns are added to the RMP to start a new iteration.

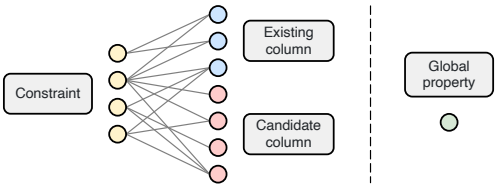


Figure 2: A toy example of state. The left part illustrates the bipartite graph representation of the current RMP, including 4 constraint nodes, 3 existing column nodes, and 4 candidate column nodes. The right part denotes the global feature vector for the problem instance.

Reward $\mathcal{R}$. The goal of the RL agent is to minimize the total number of iterations. We design a reward function consisting of a unit penalty for each additional iteration and two auxiliary components: 1) the decrease in the objective value of RMP, and 2) the sum of cosine distances of selected columns, which is inspired by the observation in (Vanderbeck 1994). The immediate reward at time step t is:

$$r_t = -1 + \alpha \cdot \left(\frac{\text{obj}_{t-1} - \text{obj}_t}{\text{obj}_0} \right) + \beta \cdot \sum_{u_i, u_j \in \mathcal{C}_s} \left(1 - \frac{\langle u_i, u_j \rangle}{\|u_i\| \cdot \|u_j\|} \right),$$

where $(\text{obj}_{t-1} - \text{obj}_t)$ is the decrease in the objective value, normalized by the objective value obj_0 of the initial RMP; $\mathcal{C}_s$ is the set of selected columns, and $\left(1 - \frac{\langle u_i, u_j \rangle}{\|u_i\| \cdot \|u_j\|}\right)$ is the cosine distance between column vectors u_i and u_j; α and β are non-negative weight hyperparameters.

Model

We use PPO (Schulman et al. 2017) as the training algorithm for our multiple-column selection strategy. PPO is a deep reinforcement learning algorithm based on the actor-critic architecture. Given a state s, the critic estimates the value function $V(s)$, and the actor gives a probability distribution $\pi = \left(\pi\left(a_1 \mid s\right), \pi\left(a_2 \mid s\right), \ldots, \pi\left(a_{|\mathcal{A}|} \mid s\right)\right)$ over the action space $\mathcal{A}$. An action is sampled from the probability distribution, and the corresponding k columns are selected and added to the RMP to start a new iteration.

We propose an actor-critic style neural network for the RL-based multiple-column selection strategy. The network consists of three components: an encoder, a critic decoder, and an actor decoder. The details of the neural network architecture are described below:

Encoder. As introduced above, the state is represented by a bipartite graph and a global feature vector. The encoder takes the bipartite graph and the global feature vector as input to produce the embeddings for the current state. The architecture of the encoder is shown in Figure 3.

Specifically, the bipartite graph and the global feature vector are embedded separately. For the bipartite graph, the encoder first computes the initial node embeddings from raw node features through a learned linear projection. Then, the node embeddings of the bipartite graph are updated through N_1 graph convolutional layers. Each layer proceeds in two phases: the first phase is performed to update the constraint node embeddings, followed by the second phase that updates the column node embeddings. Both phases are implemented using the graph isomorphism network (GIN) (Xu et al. 2019) with residual connections. Let $x_c^{(\ell)}$ and $x_v^{(\ell)}$ denote the embeddings for constraint node $c \in \mathcal{C}$ and column node $v \in \mathcal{V}$ at layer ℓ. The node embeddings are updated as follows:

$$x_c^{(\ell)} = \text{MLP}_{\mathcal{C}}^{(\ell)} \left((1+\epsilon) \cdot x_c^{(\ell-1)} + \sum_{v_i \in \mathcal{N}(c)} x_{v_i}^{(\ell-1)} \right) + x_c^{(\ell-1)},$$

$$x_v^{(\ell)} = \text{MLP}_{\mathcal{V}}^{(\ell)} \left((1+\epsilon) \cdot x_v^{(\ell-1)} + \sum_{c_i \in \mathcal{N}(v)} x_{c_i}^{(\ell)} \right) + x_v^{(\ell-1)},$$

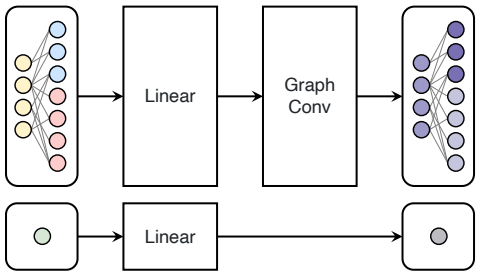


Figure 3: The architecture of the encoder. Colored nodes denote feature vectors or embeddings.

where $\text{MLP}_{\mathcal{C}}^{(\ell)}$ and $\text{MLP}_{\mathcal{V}}^{(\ell)}$ are multi-layer perceptrons (MLPs) for updating the constraint node embeddings and column node embeddings, respectively, and $\mathcal{N}(v)$ denotes the neighborhood set of node v.

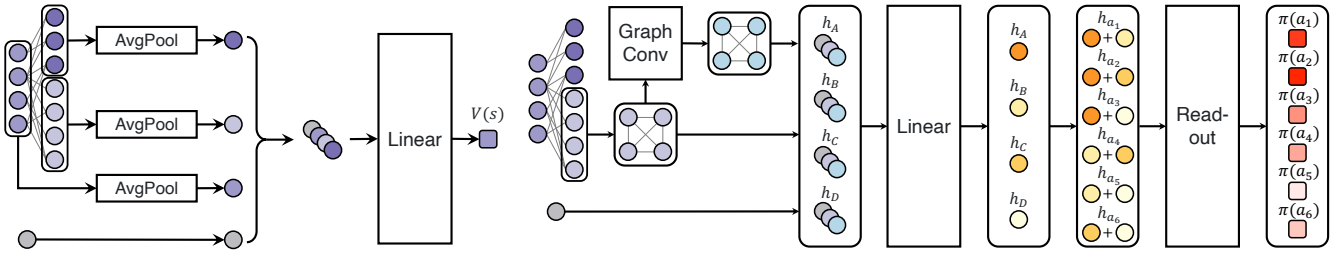
The global feature vector is embedded through N_2 linear layers, each followed by a LeakyReLU activation function.

Critic Decoder. The critic decoder maps the latent embeddings of state s into the estimated value function $V(s)$. The architecture of the critic decoder is illustrated in Figure 4(a). In the critic decoder, the node embedding vectors associated with the existing columns, candidate columns, and constraints are respectively pooled by average, and then concatenated together with the embedding vector of global features, which contains information of the current RMP as well as the global properties of the problem instance. Then, the concatenated vector is passed through an N_3 layer MLP to estimate the value function $V(s)$.

Actor Decoder. Based on the embeddings produced by the encoder, the action decoder outputs a probability distribution $\pi = \left(\pi\left(a_1 \mid s\right), \pi\left(a_2 \mid s\right), \ldots, \pi\left(a_{|\mathcal{A}|} \mid s\right)\right)$ over the action space. An action is sampled from the probability distribution, determining which k columns are selected by the RL-based multiple-column selection strategy.

The interrelations, especially the similarity between candidate columns, are crucial to the multiple-column selection. Therefore, we explicitly model the message-passing between candidate columns. We first create a complete graph, with each node corresponding to a candidate column. The node embeddings of the complete graph are initialized as the final embeddings of candidate column nodes from the bipartite graph. We associate the distance (e.g., Jaccard distance, cosine distance) between candidate column vectors as the initial edge features. As shown in Figure 4(b), the candidate column node embeddings of the bipartite graph are used to create the complete graph.

We apply the graph attention network (GAT) with edge features (Veličković et al. 2018; Kamiński et al. 2021) to update the embeddings of the complete graph through message-passing between candidate columns. Then, for each candidate column, we concatenate its node embedding from the bipartite graph, its node embedding from the complete graph, and the global embedding together. The concatenated embedding for each candidate column is processed through an N_4 layer MLP to obtain the final embed-



(a) The critic decoder (b) The actor decoder

Figure 4: The architecture of the critic decoder and actor decoder. Here shows a toy example of selecting 2 from 4 candidates.

ding vector (h_A to h_D in Figure 4(b)).

For each action a_i, we define its representation vector $h_{a_i} = \sum_{v_j \in \mathcal{C}_s(a_i)} h_{v_j}$, where $\mathcal{C}_s(a_i)$ denotes the set of columns selected in action a_i, and h_{v_j} is the final embedding vector of candidate column v_j. The probability to select a_i is computed through a learnable nonlinear readout function:

$$\pi(a_i \mid s) = \text{softmax}\left(C \cdot \tanh\left(w_o^{\mathrm{T}} \cdot \text{ReLU}\left(W_o \cdot h_{a_i}\right)\right)\right),$$

where w_o and W_o are learnable vector and matrix, respectively, and C is the clipping coefficient ($C = 10$). The actor acts as a multiple-column selection strategy, which takes the current state as input and outputs a probability distribution over the action space.

Note that we are using a learnable nonlinear mapping to derive the probability distribution for each k-combination of columns. This is by no means a simple addition of individual scores for the corresponding k columns. An action is sampled from the probability distribution, and the k candidate columns in the corresponding combination are added to the RMP for the next iteration.

Evaluation

We evaluate our proposed RL-based multiple-column selection strategy on two sets of problems: the CSP and the GCP. Both problems are well-known for the linear relaxation effectively solved using CG. Experimental results demonstrate that our RL-based strategy outperforms several widely used single-column selection strategies and multiple-column selection strategies.

Experiment Task

Cutting Stock Problem. The CG formulation of CSP has been introduced in the previous section. The problem instances are generated according to the rules for random instances in BPPLIB (Delorme, Iori, and Martello 2016, 2018), a widely used benchmark for bin packing and cutting stock problems. We divide the CSP instances into three categories: easy, normal, and hard, corresponding to the roll length $L = 50, 100$, and 200. We generated 1000, 200, and 100 instances of the three instance categories for evaluation, respectively. Instances for training are randomly generated and solved using CG as the environment for the RL agent.

Graph Coloring Problem. The GCP aims to assign a minimum number of colors to the nodes on a graph, such that every pair of adjacent nodes does not share the same color (Malaguti and Toth 2010). In the CG formulation for GCP, the RMP can be expressed as using a minimum number of maximal independent sets (MISs) to cover all the nodes, while the PP is modeled as the maximum weight independent set problem (MWISP) to search for new MISs with the most negative reduced cost. The details are described in (Mehrotra and Trick 1996). Similar to the CSP, the GCP instances are divided into three categories, corresponding to the number of nodes $N = 30, 40$, and 50 respectively. The CGP instances are generated according to the rules for random graphs in (Mehrotra and Trick 1996).

Hyperparameter Configuration

We implement our model to learn the RL-based multiple-column strategy. We select 5 out of 10 candidate columns at each iteration, which strikes a balance between the number of iterations and the cost per iteration in our task. To guarantee convergence, we force the optimal solution of the PP to always be selected. We set the number of layers $N_1 = N_2 = N_3 = N_4 = 3$ for the MLPs in the encoder and decoder. The weights in the reward function are set to $\alpha = 300$ and $\beta = 0.02$ to balance the reward scales and the discount factor γ is set to 0.9. We use PPO with a clipping threshold $\epsilon = 0.2$, and the Adam optimizer with a learning rate 1×10^{-3} to train the RL model. The hyperparameter configuration is fixed across all instance categories of CSP and GCP.

Comparison Evaluation

We compare our RL-based multiple-column strategy with several well-established single-column and multiple-column selection strategies, as well as the multiple-column selection strategy using the MILP expert proposed in (Morabit, Desaulniers, and Lodi 2021) and the DQN-based approach proposed in (Chi et al. 2022). The details of baseline strategies for comparison are as follows:

Single-column selection strategy:

- *Greedy single-column selection (Greedy-S)*: Always select the column with the most negative reduced cost.
- *Random single-column selection (Random-S)*: Randomly select a column from the candidate column pool.
- *DQN-based single-column selection (DQN-S)*: Selection strategy based on DQN in (Chi et al. 2022).

Strategy	CSP (Easy)		CSP (Normal)		CSP (Hard)	
	# Itr	Time	# Itr	Time	# Itr	Time
Greedy-S	37.68	228.92	89.20	186.78	171.44	301.07
Random-S	62.63	374.80	116.82	257.04	205.31	376.16
DQN-S	35.54	215.65	88.95	178.52	/	/
Greedy-M	12.03	75.34	27.01	62.60	52.13	96.42
Random-M	13.97	84.83	28.26	63.78	51.43	96.51
MILP-M	10.65	96.23	23.46	81.17	44.99	147.19
Diverse-M	11.45	74.59	25.03	59.32	47.24	93.60
Ours	**10.33**	**67.84**	**22.85**	**55.05**	**43.95**	**87.95**

Table 1: Comparison results on the CSP, in terms of the average number of iterations per instance and total runtime (in seconds) over the evaluation instance set.

Multiple-column selection strategy:

- *Greedy multiple-column selection (Greedy-M)*: Always select the top-k columns according to the reduced cost.
- *Random multiple-column selection (Random-M)*: Randomly select k columns from the candidate column pool.
- *MILP expert (MILP-M)*: Selection strategy using the MILP expert in (Morabit, Desaulniers, and Lodi 2021).
- *Diversification-based column selection (Diverse-M)*: A modified strategy of CGDS (Moungla, Létocart, and Nagih 2010) to fit our task: We first sort the candidate columns by their reduced costs, and prioritize the candidate columns that are disjoint from the already selected columns, if there exists one in the remaining pool.

For a fair comparison, we set the same number of candidate columns and the columns to select in all multiple-column selection strategies. The candidate columns are generated as the 10 columns with the most negative reduced cost from PP. We report the evaluation metrics of 1) the average number of iterations per instance and 2) the total runtime over the evaluation set. Generally, all heuristic strategies and the RL-based strategies (on GPU), except MILP-M, require negligible time for a selection decision, so the comparison of the average number of iterations is approximately equivalent to the comparison of runtime for these strategies. We remark that MILP-M is practically intractable because the time taken by the MILP expert in MILP-M is even larger than the time for RMP and PP (Morabit, Desaulniers, and Lodi 2021). As shown in the experimental results, even if MILP-M requires fewer iterations for convergence compared to other heuristic column selection strategies, its runtime is significantly larger due to the expensive decisions from the MILP expert.

Results on CSP. The comparison results on CSP are reported in Table 1. All the multiple-column selection strategies achieve significantly faster convergence than the single-column selection strategies, especially on large-scale problem instances. Diverse-M requires the least runtime among the baseline strategies. While MILP-M requires fewer iterations than Diverse-M, it takes a much larger runtime due to the extremely expensive column selection decisions.

Strategy	GCP (Easy)		GCP (Normal)		GCP (Hard)	
	# Itr	Time	# Itr	Time	# Itr	Time
Greedy-M	18.30	75.16	29.00	99.54	39.04	123.23
Random-M	19.04	78.81	30.91	104.79	41.01	125.26
MILP-M	17.07	93.78	25.86	128.71	34.19	188.44
Diverse-M	18.07	74.36	28.17	96.14	37.78	119.90
Ours	**15.19**	**62.06**	**24.93**	**87.31**	**34.11**	**111.56**

Table 2: Comparison results on the GCP.

The experimental results show that our RL approach can learn a stronger strategy for column selection, which is implicitly represented by the neural network. The RL-based multiple-column selection strategy outperforms all baseline column selection strategies in the three instance categories of various sizes. Compared to the best baseline strategy (Diverse-M), our RL-based multiple-column selection strategy yields a total runtime reduction of **9.05%**, **7.20%**, and **6.04%** in the three instance categories, respectively. In addition, it is worth mentioning that our RL-based multiple-column selection strategy requires even fewer iterations than MILP-M. This is mainly because the MILP expert focuses only on the current step and its goal is to minimize the objective value for the very next iteration, whereas our RL agent aims to minimize the total number of iterations and takes into consideration the long-term effect of currently selected columns on future iterations.

Results on GCP. Since the single-column selection strategies have been demonstrated to be ineffective on CSP, we conduct experiments on GCP using only multiple-column selection strategies. As reported in Table 2, it shows similar results to the experiments on CSP. Compared to Diverse-M, our RL-based strategy reduces the total runtime by **16.54%**, **9.18%**, and **6.96%** in the three instance categories, respectively. The evaluation results on CSP and GCP demonstrate the effectiveness of our RL-based multiple-column selection strategy on different types of problems.

Generalization Evaluation

Generalization across different instance sizes is a highly desirable property for learning-based models. The ability to generalize across sizes would allow the RL-based strategy to scale up to larger instances while training more efficiently on smaller instances. Table 3 presents the generalization performance of our RL-based multiple-column selection strategy, which is trained on instances from the hard category but evaluated on much larger instances. For the CSP, the model is trained on instances with $L = 200$ and evaluated on instances with $L = 500$ and $L = 1000$; for the GCP, the model is trained on instances with $N = 50$ and evaluated on instances with $N = 75$ and $N = 100$. Our RL-based multiple-column selection strategy still shows advantages over most baseline column selection strategies on the evaluation instances, which are several times larger than the training instances. It is demonstrated that our RL-based multiple-column selection strategy has learned useful and

Strategy	CSP (L=500)		CSP (L=1000)		GCP (N=75)		GCP (N=100)	
	# Itr	Time	# Itr	Time	# Itr	Time	# Itr	Time
Greedy-M	78.80	143.25	98.82	221.75	60.88	287.52	83.60	449.95
Random-M	74.96	136.66	85.74	206.81	65.50	301.77	87.73	469.44
MILP-M	67.26	247.00	**81.92**	482.13	53.40	438.38	75.22	788.65
Diverse-M	70.44	130.84	83.74	196.92	57.80	259.46	83.07	449.74
Ours	**66.94**	**118.89**	82.04	**183.76**	**52.96**	**246.63**	**74.80**	**398.37**

Table 3: Generalization performance of our RL-based multiple-column selection strategy to larger problem sizes.

Strategy	CSP (Easy)		CSP (Normal)		CSP (Hard)	
	# Itr	Time	# Itr	Time	# Itr	Time
Greedy-M	12.03	75.34	27.01	62.60	52.13	96.42
Random-M	13.97	84.83	28.26	63.78	51.43	96.51
Diverse-M	11.45	74.59	25.03	59.32	47.24	93.60
DQN-M	11.50	74.71	25.94	60.36	48.59	94.51
Variant 1[i]	10.91	71.48	24.38	58.40	45.75	92.16
Variant 2[ii]	10.40	68.21	23.03	56.87	44.92	89.30
Complete Model	**10.33**	**67.84**	**22.85**	**55.05**	**43.95**	**87.95**

[i] with the embeddings of the complete graph removed.
[ii] with the embedding of the input global features removed.

Table 4: Performance of the RL-based multiple-column selection strategies using different neural network architectures.

effective selection principles that are invariant to the size of problem instances.

Ablation Study

We have demonstrated the effectiveness of our proposed RL-based multiple-column selection strategy on CSP and GCP. To show the effect of different components of the neural network, we consider two variants of the complete model: 1) removing the embeddings of the complete graph and 2) removing the embedding of input global features. Other components remain unchanged.

We conduct the ablation evaluation on CSP and the results are reported in Table 4. Compared to the complete model, the performance of both variants is degraded on all three instance categories. The results show that the embeddings of the complete graph and the embeddings of explicit global features both provide benefits for the learned multiple-column selection strategy. Notably, the performance of the learned multiple-column selection strategy decreases obviously when the embeddings of the complete graph are removed, which further highlights the importance of the candidate column interrelations in column selection.

To show the advantage of our approach over the framework of (Chi et al. 2022), we conduct an extended experiment using a modified implementation of their DQN-based approach for multiple-column selection (DQN-M in Table 4), where we select the top-k columns based on their Q-values. DQN-M outperforms the greedy and random baselines but underperforms Diverse-M. This is because DQN-M independently selects k columns with higher Q-values: while each of them can individually lead to good convergence, the combination of them is not necessarily the best, just as 5 Michael Jordans in a basketball team may not be better than a reasonable lineup. In other words, DQN-M is selecting "the top-k columns", while our RL-based multiple-column selection strategy is devoted to selecting "the best k-combination".

Conclusion

In this paper, we propose an RL-based multiple-column selection strategy for CG. We formulate the multiple-column selection task as an MDP, and introduce an actor-critic style neural network that takes into account the column-constraint structure of RMP, the interrelations of candidate columns, as well as global properties of the problem instance. We evaluate our proposed RL-based multiple-column selection strategy on two sets of problems: the CSP and the GCP. Experimental results show that our RL-based multiple-column selection strategy outperforms all baseline single-column and multiple-column selection strategies in the three instance categories of various sizes. Extensive experiments also demonstrate the ability of our RL-based model to generalize to larger-scale problem instances.

Despite the significant performance of the RL-based multiple-column selection strategy, more progress can be made in exploring column selection strategies to select different numbers of columns adaptively based on the problem properties and solution stages. It is challenging for handcrafted rules due to the various characteristics of different types and sizes of problems, but may be learned using an RL agent through the interaction with the CG solution environment. In addition, how to incorporate the learning-based column selection strategy with other acceleration methods for CG, such as dual stabilization and PP reduction, is also a possible direction for future efforts.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61936009; and in part by the National Science and Technology Innovation 2030 Major Project of the Ministry of Science and Technology of China under Grant 2018AAA0101604. We also thank Wanlu Yang and Peng Jiang for their valuable comments and corrections.

References

Babaki, B.; Jena, S. D.; and Charlin, L. 2021. Neural column generation for capacitated vehicle routing. In *AAAI-22 Workshop on Machine Learning for Operations Research*.

Barnhart, C.; Johnson, E. L.; Nemhauser, G. L.; Savelsbergh, M. W. P.; and Vance, P. H. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46(3): 316–329.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2021. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421.

Chi, C.; Aboussalah, A. M.; Khalil, E. B.; Wang, J.; and Sherkat-Masoumi, Z. 2022. A deep reinforcement learning framework for column generation. In *Advances in Neural Information Processing Systems*.

Delorme, M.; Iori, M.; and Martello, S. 2016. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1): 1–20.

Delorme, M.; Iori, M.; and Martello, S. 2018. BPPLIB: A library for bin packing and cutting stock problems. *Optimization Letters*, 12(2): 235–250.

Desaulniers, G.; Desrosiers, J.; and Solomon, M. M. 2002. Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems. *Essays and Surveys in Metaheuristics*, 309–324.

Desaulniers, G.; Desrosiers, J.; and Solomon, M. M. 2006. *Column generation*. Springer Science & Business Media.

du Merle, O.; Villeneuve, D.; Desrosiers, J.; and Hansen, P. 1999. Stabilized column generation. *Discrete Mathematics*, 194(1-3): 229–237.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*.

Gilmore, P. C.; and Gomory, R. E. 1961. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6): 849–859.

Goffin, J.-L.; and Vial, J.-P. 2000. Multiple cuts in the analytic center cutting plane method. *SIAM Journal on Optimization*, 11(1): 266–288.

Gurobi Optimization, LLC. 2023. Gurobi optimizer reference manual.

Kamiński, K.; Ludwiczak, J.; Jasiński, M.; Bukala, A.; Madaj, R.; Szczepaniak, K.; and Dunin-Horkawicz, S. 2021. Rossmann-toolbox: A deep learning-based protocol for the prediction and design of cofactor specificity in Rossmann fold proteins. *Briefings in Bioinformatics*, 23(1): bbab371.

Lübbecke, M. E.; and Desrosiers, J. 2005. Selected topics in column generation. *Operations Research*, 53(6): 1007–1023.

Malaguti, E.; and Toth, P. 2010. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1): 1–34.

Mazyavkina, N.; Sviridov, S.; Ivanov, S.; and Burnaev, E. 2021. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134: 105400.

Mehrotra, A.; and Trick, M. A. 1996. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4): 344–354.

Morabit, M.; Desaulniers, G.; and Lodi, A. 2021. Machine-learning-based column selection for column generation. *Transportation Science*, 55(4): 815–831.

Moungla, N. T.; Létocart, L.; and Nagih, A. 2010. Solutions diversification in a column generation algorithm. *Algorithmic Operations Research*, 5(2): 86–95.

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

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shen, Y.; Sun, Y.; Li, X.; Eberhard, A.; and Ernst, A. 2022. Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring. In *AAAI Conference on Artificial Intelligence*.

Tahir, A.; Quesnel, F.; Desaulniers, G.; Hallaoui, I. E.; and Yaakoubi, Y. 2021. An improved integral column generation algorithm using machine learning for aircrew pairing. *Transportation Science*, 55(6): 1411–1429.

Vanderbeck, F. 1994. *Decomposition and column generation for integer programs*. Ph.D. thesis, UCL-Université Catholique de Louvain.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How powerful are graph neural networks? In *International Conference on Learning Representations*.

Yang, W.; Jiang, P.; and Song, S. 2023. High-speed Train Timetabling Based on Reinforcement Learning. In *IEEE Symposium Series on Computational Intelligence*, 1187–1193.

Yuan, H.; Jiang, P.; and Song, S. 2022. The neural-prediction based acceleration algorithm of column generation for graph-based set covering problems. In *IEEE International Conference on Systems, Man, and Cybernetics*, 1115–1120.

Zhang, C.; Song, W.; Cao, Z.; Zhang, J.; Tan, P. S.; and Xu, C. 2020. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*.