FFCG: Effective and Fast Family Column Generation for Solving Large-Scale Linear Program

Yi-Xiang Hu[1], Feng Wu[1*], Shaoang Li[1], Yifang Zhao[2], Xiang-Yang Li[1*]

[1]School of Computer Science and Technology, University of Science and Technology of China
[2]School of Cyber Science and Technology, University of Science and Technology of China
{yixianghu,lishaoa,zhaoyifang}@mail.ustc.edu.cn,{wufeng02,xiangyangli}@ustc.edu.cn

Abstract

Column Generation (CG) is an effective and iterative algorithm to solve large-scale linear programs (LP). During each CG iteration, new columns are added to improve the solution of the LP. Typically, CG greedily selects one column with the most negative reduced cost, which can be improved by adding more columns at once. However, selecting all columns with negative reduced costs would lead to the addition of redundant columns that do not improve the objective value. Therefore, selecting the appropriate columns to add is still an open problem and previous machine-learning-based approaches for CG only add a constant quantity of columns per iteration due to the state-space explosion problem. To address this, we propose Fast Family Column Generation (FFCG) — a novel reinforcement-learning-based CG that selects a variable number of columns as needed in an iteration. Specifically, we formulate the column selection problem in CG as an MDP and design a reward metric that balances both the convergence speed and the number of redundant columns. In our experiments, FFCG converges faster on the common benchmarks and reduces the number of CG iterations by 77.1% for Cutting Stock Problem (CSP) and 84.8% for Vehicle Routing Problem with Time Windows (VRPTW), and a 71.4% reduction in computing time for CSP and 84.0% for VRPTW on average compared to several state-of-the-art baselines.

1 Introduction

In many real-world applications, the successful solution of large-scale Mixed-Integer Linear Programming (MILP) problems requires solving Linear Programming (LP) relaxations that have a huge number of variables. For example, in the Cutting Stock Problem (CSP) with a length of n, each integer variable represents the number of times a cutting pattern is used, and there are exponentially many (2^n) cutting patterns. However, LP involving a large number of variables (i.e., columns) often cannot be handled at once by the solver. To address this, Gilmore and Gomory (1961) proposed an iterative algorithm, called Column Generation (CG), which is widely used to solve large-scale LPs. Specifically, CG starts by solving a Restricted Master Problem (RMP) with a subset of columns and gradually includes new columns, which can improve the solution of the current RMP, by solving a pricing subproblem (SP). When no more columns with negative reduced cost are found, CG will provably converge to an optimal solution to the LP. In practice, CG is often used by LP relaxation solvers in the branch-and-price algorithm (Desaulniers, Desrosiers, and Solomon 2005) for large MILP.

Here, we focus on how to select columns in each iteration efficiently and speed up the convergence of CG. Typically, CG greedily selects a single column with the most negative reduced cost, which can be sped up by adding multiple columns at once (Desaulniers, Desrosiers, and Solomon 2002). To distinguish from single-column selection, we refer to CG with multi-column selection as Family Column Generation (FCG). Unfortunately, adding multiple columns raises another issue: if CG greedily selects all columns with negative reduced costs, many redundant columns that do not improve the objective value will be selected, which increases the computing time. Therefore, how to 1) *identify the most effective columns* and 2) *decide the appropriate column number* are keys to the performance of FCG.

In the last few years, researchers have become increasingly interested in machine learning methods for mathematical optimization including large LPs (Bengio, Lodi, and Prouvost 2021). Most related to our work, Chi et al. (2022) proposed the first Reinforcement Learning (RL) framework for CG, modeling column selection as a Markov Decision Process (MDP). Recently, Yuan, Fang, and Song (2024) devised a multiple-column selection strategy based on RL for CG. Specifically, they limited the action space size by adding only a *fixed* number of columns per CG iteration. Note that the action space grows exponentially when all possible combinations of candidate columns are considered. Most importantly, when multiple columns are added, the contribution of each column towards convergence varies with the combination of columns in different phases of CG. Thus, it is essential to efficiently select the best combination of columns from candidates at each CG iteration to avoid increasing computational overhead with redundant columns. This raises the issue of the so-called *credit assignment for columns*, which pertains to determining the contribution of each column toward CG convergence.

To address the aforementioned challenges, we propose a novel Fast Family Column Generation, named FFCG. Our main contributions are summarized as follows:

*Corresponding authors.

- **Fast multiple-column selection in an iteration:** We propose a novel RL-based family column generation, which selects a *variable* number of columns in an iteration. In each time slot, the size of the action space is reduced from $O(2^n)$ to $O(n^2)$. We show that FFCG offers better tradeoffs between speeding up the convergence of CG and reducing the total computational time.
- **Reward design and analysis:** To address the credit assignment problem for columns, we carefully design the reward function and evaluate the contribution of each column in each iteration. We also analyze how this design helps FFCG reduce unnecessary computing expenses.
- **Substantial improvements over baselines:** We evaluate FFCG using the common benchmarks for CSP and VRPTW. In our experiments, FFCG converges faster on the benchmarks and reduces the total computing time by 71.4% and 84.0% on average compared to several state-of-the-art methods for CSP and VRPTW.

The rest of the paper is organized as follows. Firstly, we briefly review previous research on this topic. Subsequently, we present both the standard and RL-based CG methods to solve large LPs. Then, we propose our FFCG framework, which encompasses formulation, analysis, training, and execution. Finally, we assess the effectiveness of FFCG on CSP and VRPTW, in comparison to the baseline approaches, followed by our conclusion and discussion.

2 Related Work

In this section, we review the acceleration methods for CG, with a focus on recent advancements in ML-based CG.

2.1 Acceleration Methods for Column Generation

To speed up CG, one method is to add multiple columns instead of just one with the most negative reduced cost. Goffin and Vial (2000) showed that the convergence process can be sped up by selecting non-correlated columns. Then, two practical strategies were proposed for selecting multiple columns (Touati, Létocart, and Nagih 2010). These strategies were designed to increase the diversity of the selected columns. However, there is no perfect strategy for selecting columns that can minimize the number of iterations for CG while also considering computing costs.

Another approach is dual stabilization, which aims to formulate a better pricing subproblem as the pricing subproblem is a bottleneck for computing time. Ben Amor, Desrosiers, and Valério de Carvalho (2006) studied the use of two types of Dual-Optimal Inequalities (DOI) to accelerate and stabilize the whole convergence process, followed by Gschwind and Irnich (2016); Václavík et al. (2018); Yarkony et al. (2020); Haghani, Contardo, and Yarkony (2022). We note that our column selection strategy does not conflict with dual stabilization techniques, heuristic, and meta-heuristic methods. They can be used synergistically for further improvement (Yuan et al. 2021; Shen et al. 2024).

2.2 Machine-Learning-based Column Generation

Except for the previously mentioned RLCG (Chi et al. 2022), Morabit, Desaulniers, and Lodi (2021) proposed machine-learning-based column selection to accelerate CG. The approach applies a learned model to select a subset of columns generated at each iteration of CG. It reduces the computing time spent reoptimizing the RMP at each iteration by selecting the most promising columns. Babaki, Jena, and Charlin (2022) formulated the task of choosing one of the columns to be included in the RMP as a contextual MDP. They proposed and explored several architectures for improving the convergence of the CG algorithm using deep learning. However, it also only adds one column at each iteration. Recently, Yuan, Fang, and Song (2024) proposed the first RL-based multiple-column selection strategy for CG, which selects k columns from the pool of n candidate columns generated by the SP. In this approach, k remains fixed. However, selecting more columns in the early stage and fewer columns in the later stage helps to expedite the convergence of CG.

3 Background

In this section, we first provide an overview of the standard CG algorithm and then briefly describe the RLCG method.

3.1 Column Generation in LP

Let us consider the following generic Linear Program (LP), called the Master Problem (MP):

$$\min \sum\nolimits_{p \in P} c_p \theta_p \tag{1}$$

subject to

$$\sum\nolimits_{p \in P} \mathbf{a}_p \theta_p = \mathbf{b};\ \theta_p \geq 0,\ \forall p \in P \tag{2}$$

where P is the index set of variables θ_p; $c_p \in \mathbb{R}$ and $\mathbf{a}_p \in \mathbb{R}^m$ are the cost coefficient and constraint coefficient vector of θ_p, respectively; and $\mathbf{b} \in \mathbb{R}^m$ is the right-hand-side vector of the constraints. We assume that the number of variables θ_p is very large and the set of objects associated with these variables can be implicitly modeled as solutions of an optimization problem.

The standard CG proceeds to solve this MP as follows. In each iteration of CG, the RMP that considers a subset $\Omega \subseteq P$ of the columns is solved first. It yields a primal solution x (assuming that $\theta_p = 0, \forall p \in \Omega \backslash P$) and a dual solution given by the dual values $\pi \in \mathbb{R}^m$ associated with the constraints. Next, the dual solution is used to identify new negative reduced cost variables θ_p, by solving the following pricing subproblem:

$$\bar{c} = \min_{p \in P} \left\{ c_p - \pi^T \mathbf{a}_p \right\} \tag{3}$$

If negative reduced cost columns are found, we append them in Ω to the RMP, and the entire procedure is iterated. Otherwise, CG stops since π is an optimal dual solution to the original problem and together with the optimal primal solution to the RMP, i.e., we have an optimal primal/dual pair.

For some problems, the search for negative reduced cost columns can be distributed across several SPs. When the RMP includes too many columns after several iterations, columns with large reduced cost can be removed.

3.2 Reinforcement Learning for CG (RLCG)

RLCG (Chi et al. 2022) formulates CG as an MDP, denoted as $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$, where: $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$, $(s', s, a) \mapsto \mathbb{P}(s'|s, a)$ the transition function, $r : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, and $\gamma \in (0, 1)$ the discount factor.

At a high level, the method works as follows. The SP is solved at each iteration and a set of near-optimal column candidates $\mathcal{G}$ is returned, which is a general feature of optimization solvers such as Gurobi (Gurobi Optimization, LLC 2024). RLCG selects a column from $\mathcal{G}$ according to the Q-function learned by the RL agent. The RL agent is fused within the CG loop and actively selects the column to be added to the next iteration using the information extracted from the current RMP and SP.

The state S is represented by the bipartite graph of the current CG iteration from the RMP and the candidate columns from the SP. As shown in the left part of Figure 1, the RMPs are encoded using bipartite graphs with columns (variables) nodes (v) and constraint nodes (c) (Gasse et al. 2019). The edge between v and c in the graph indicates the contribution of a column v to a constraint c. Note that each node is characterized by its feature vector. Each action in the action set $\mathcal{G}$ represents a candidate column (green node). The RL agent selects one column (action) a to add to the RMP for the next iteration from the candidate columns set $\mathcal{G}$ returned from the current iteration SP. Transitions are deterministic. After selecting an action from the current candidate set $\mathcal{G}$, the selected column enters the basis in the next RMP iteration. The detailed definition of the reward function is given by Chi et al. (2022). A GNN is used as a Q-function approximator, trained with experience replay (Mnih et al. 2015).

4 FFCG Framework

Here, we propose our FFCG framework. At a high level, FFCG works very similar to RLCG. At each iteration, we solve the RMP to obtain dual values and use the dual values to update the SP objective function. Then, the SP is solved. We define a solution gap for the worst-case candidate column, ensuring that its reduced cost remains close to the optimal value, thus making it a near-optimal candidate for selection. Subsequently, the set of candidate columns $\mathcal{G}$ is returned. With $\mathcal{G}$ and the context S in the bipartite graph, it selects a set of columns $\mathcal{C}$ to be added to the next iteration. This is the key difference from RLCG where only a single column is selected. While no column with negative reduced cost exists, it stops and the optimal solution is returned. Now, there are two key challenges: 1) *how to select effective columns from the candidates* and 2) *when to stop column selection*.

4.1 MDP Formulation

To formulate FCG as an MDP, a candidate column $a_{i,t}$ in each CG iteration is called an available action. Let the candidate columns in the t^{th} CG iteration $\mathcal{G}_t = \{a_{1,t}, \ldots, a_{|\mathcal{G}_t|,t}\}$ be the set of available actions in the time slot t. Note that this formulation captures the volatile actions: action sets $\mathcal{G}_t$, $\forall\ 0 < t \leq T$ (and their size) in different time slots can be different from each other. For each available action $a_{i,t}$, its context (side information) can be observed. Let $R_t(\mathcal{C}_t)$ be the quality of the selected column set $\mathcal{C}_t$ (the reward of choosing $\mathcal{C}_t$ based on the observed context S_t in time slot t). Given the available actions $\mathcal{G}_t$ to choose from in each time slot, our objective is to pick a subset of actions $\mathcal{C}_t \subseteq \mathcal{G}_t$ to maximize the total reward.

At each iteration (time slot), SP returns a set of columns $\mathcal{G}_t$ with negative reduced costs. Then every possible combination of candidate columns can be returned as $\mathcal{C}_t$. In other words, in each iteration, the RL agent can select one or more columns to be added to the next iteration. For example, as shown in the left-hand side of Figure 1, the set of available columns (the green column nodes) is $\mathcal{G}_t = \{v_4, v_5, v_6, v_7\}$, and the set of available $\mathcal{C}_t$ is $\mathcal{P}(\mathcal{G}_t)\backslash\{\emptyset\}$, where $\mathcal{P}(\mathcal{G}_t)$ is the power set of $\mathcal{G}_t$. Although adding more columns can reduce the total number of iterations, it also increases the computing costs. Therefore, the RL agent should learn to select the most effective columns set $\mathcal{C}_t^*$, balancing both speed and costs.

In what follows, we will describe how to select a subset of columns $\mathcal{C}_t \subseteq \mathcal{G}_t$ based on the context S.

4.2 Column Selection

To avoid the exponential growth in the action space, we select the candidate columns one by one. In other words, we turn the factored action space into a sequential choice, inspired by Wen et al. (2022), and only one column is selected in each turn. Note that this is similar to the local search in the huge action space, where a move in each dimension is decided at a time. Although it has no guarantee to select the optimal combination, this simple strategy performs well in practice given a good marginal value, as shown later in our experiments. Moreover, it significantly reduces the action space considered in RL. There are $O(|\mathcal{G}_t|)$ actions in each step, and the number of steps in each time slot is $O(|\mathcal{G}_t|)$. Thus, the action space size of FFCG is $O(|\mathcal{G}_t|^2)$, while the action space size of RLCG is $O(2^{|\mathcal{G}_t|})$. In addition, we add the STOP action (a blank column) to the action space for convenience. When the STOP action is selected, the column selection is stopped and the selected columns are returned.

Algorithm 1 outlines the main procedure utilized by the RL agent to make its selection of suggested columns. Firstly, the agent gets the expected marginal Q-value $\hat{Q}_{\Delta,t}$ of each action (select a candidate column). The action with the greatest expected marginal Q-value would be added greedily in the suggested columns set $\mathcal{C}_t$. Then, the STOP action is added to the action space $\mathcal{G}_t$, and the last selected action is removed. We repeat the above steps until the STOP action selected. In the end, the STOP action (a blank column) is removed from $\mathcal{C}_t$ and the suggested columns set is returned.

To learn the marginal Q-value, the key challenge is how to approximately compute the expected marginal reward.

4.3 Reward Design

For given suggested columns $\mathcal{C}_t$, we design the reward function consisting of two components: 1) the change in the RMP objective value, where a bigger decrease in value is preferred; and 2) the penalty for redundant columns, which

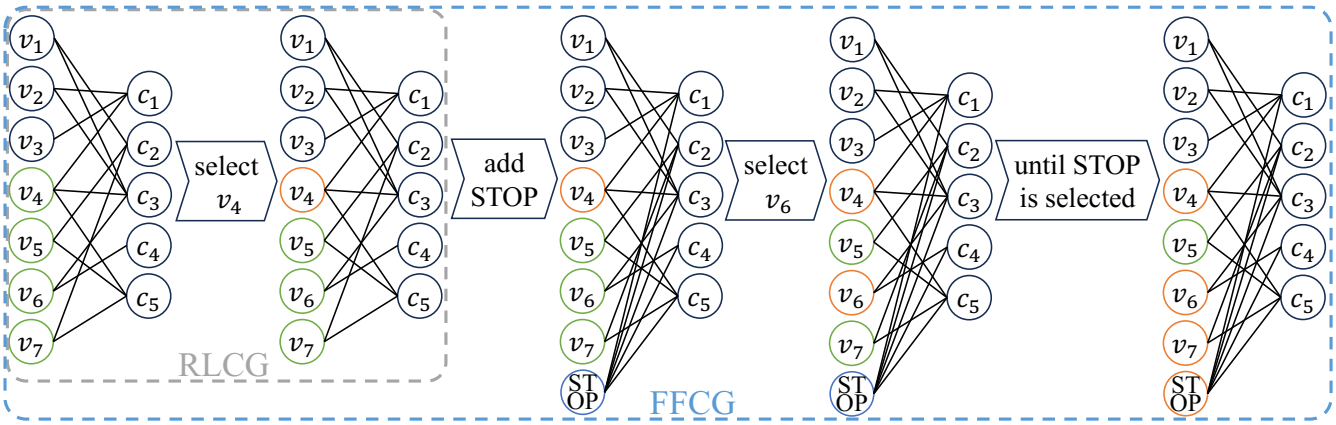


Figure 1: Column Selection in FFCG: First, the available action set is $\mathcal{G} = \{v_4, v_5, v_6, v_7\}$ (green node), and the RL agent selects v_4 (the selected node is orange). Then, a new available action STOP (blue node), which means stop column selection and return the selected columns, is added to action space $\mathcal{G} - \{v_4\}$. The RL agent repeatedly updates the state and selects from the remaining actions until the action STOP is selected. Finally, the suggested columns set $\mathcal{C} = \{v_4, v_6, v_7\}$ is returned.

Algorithm 1: Column Selection

Input: Actions set $\mathcal{G}$, and context S
Output: Suggested columns set $\mathcal{C}$

$\mathcal{C} \leftarrow \emptyset, k \leftarrow 0$
$\hat{Q}_\Delta \leftarrow \text{MarginalQValueApproximator}(\mathcal{G}, \mathcal{C}, S)$
$a_k \leftarrow \arg\max_{a \in \mathcal{G}} \hat{Q}_\Delta(\mathcal{C} \cup \{a\}, a)$
$\mathcal{C} \leftarrow \mathcal{C} \cup \{a_k\}$ ▷ *Select a column*
$\mathcal{G} \leftarrow (\mathcal{G} - \{a_k\}) \cup \{\text{STOP}\}$ ▷ *Add STOP action*
$S \leftarrow \text{UpdateContext}(\mathcal{G}, \mathcal{C}, S)$
$k \leftarrow k + 1$
while STOP is not selected **do**
  $\hat{Q}_\Delta \leftarrow \text{MarginalQValueApproximator}(\mathcal{G}, \mathcal{C}, S)$
  $a_k \leftarrow \arg\max_{a \in \mathcal{G}} \hat{Q}_\Delta(\mathcal{C} \cup \{a\}, a)$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{a_k\}$ ▷ *Select a column*
  $\mathcal{G} \leftarrow \mathcal{G} - \{a_k\}$
  $S \leftarrow \text{UpdateContext}(\mathcal{G}, \mathcal{C}, S)$
  $k \leftarrow k + 1$
end while
return $\mathcal{C} \leftarrow \mathcal{C} - \{\text{STOP}\}$

are no improvements to the objective value. Together, they incentivize the RL agent to converge faster and avoid selecting redundant columns.

Specifically, the reward of a subset of columns $\mathcal{C}_t$ at time slot t is defined as:

$$R_t(\mathcal{C}_t) \triangleq \alpha \left(\frac{obj_{t-1} - obj_t}{obj_0} \right) - \beta |\mathcal{C}_t - \mathcal{C}'_t| \qquad (4)$$

where obj_0 is the objective value of the RMP in the first CG iteration, which is used to normalize the objective value change ($obj_{t-1} - obj_t$) across instances of various sizes; α is a non-negative hyperparameter that weighs the normalized objective value change in the reward; $\mathcal{C}'_t \subseteq \mathcal{C}_t$ is the optimal subset of $\mathcal{C}_t$, in which all columns make improvement to the RMP objective value; $|\mathcal{C}_t - \mathcal{C}'_t|$ counts the number of redundant columns that are in $\mathcal{C}_t$ but not in $\mathcal{C}'_t$; β is a non-negative hyperparameter that weighs the penalty of selecting redundant columns. Increasing the values of parameters α will enable the RL agent to choose more columns. On the other hand, a higher value of β will prevent the RL agent from selecting too many unnecessary columns.

Marginal rewards. Note that the aforementioned reward is the total reward of the suggested columns $\mathcal{C}_t$. However, the total rewards achieved by selected columns are not a simple sum of individual rewards but demonstrate a feature of diminishing returns determined by the relations between selected columns (e.g. relevance and redundancy) (Chen, Xu, and Lu 2018). In Algorithm 1, we only use the marginal Q-value of each column instead of the total Q-value to form $\mathcal{C}_t$. To this end, we must assign the credit to each column and compute the marginal reward of an individual column.

Here, we denote the marginal reward of an individual column a ($a \in \mathcal{C}_t$) to a set $\mathcal{C}_t$ by

$$r_{\Delta,t}(\mathcal{C}_t, a) \triangleq R_t(\mathcal{C}_t) - R_t(\mathcal{C}_t - \{a\}) \qquad (5)$$

For redundant columns that do not improve the CG convergence ($a \notin \mathcal{C}'_t$), the marginal reward of them is $-\beta$. For effective columns ($a \in \mathcal{C}'_t$), the contribution weight of each column a is

$$\phi_t(\mathcal{C}_t, a) \triangleq \frac{r_{\Delta,t}(\mathcal{C}_t, a)}{\sum_{a' \in \mathcal{C}'_t} r_{\Delta,t}(\mathcal{C}_t, a')} \qquad (6)$$

Now, the reward of each column is proportional to its marginal reward. Thus, the reward of an individual column a ($a \in \mathcal{C}_t$) in time slot t is

$$r_{\Delta,t}(\mathcal{C}_t, a) = \begin{cases} \phi_t(\mathcal{C}_t, a) R_t^{obj}(\mathcal{C}_t) & a \in \mathcal{C}'_t, \\ -\beta & a \notin \mathcal{C}'_t. \end{cases} \qquad (7)$$

where the term $R_t^{obj}(\mathcal{C}_t) = \alpha(\frac{obj_{t-1} - obj_t}{obj_0})$ denotes the total contribution of all effective columns $a \in \mathcal{C}'_t$ to the objective value change, i.e., $R_t(\mathcal{C}_t)$ without the second term.

Algorithm 2: FFCG Training and Execution

Input: Problems $\{p_i\}_{i=1}^{M}$ and hyperparameters α, β

Output: Q function approximator $\hat{Q}^*$ at training time or optimal solutions at execution time

if at training time **then**
  Initialize replay memory D to capacity N
  Initialize Q function approximator $\hat{Q}$ with random weights θ and target $\hat{Q}^*$ with weights $\theta^- = \theta$
end if
for $i \leftarrow 1$ to M **do**
  $t \leftarrow 0$
  $\mathrm{RMP}_0 \leftarrow \mathrm{Initialize}(p_i)$
  Solve RMP_0 to get dual values
  Use dual values to construct SP_0
  $\mathcal{G}_0 \leftarrow \mathrm{GetCandidateColumns}(\mathrm{RMP}_0, \mathrm{SP}_0)$
  while CG algorithm has not converged ($\mathcal{G}_t \neq \emptyset$) **do**
    $S_t \leftarrow \mathrm{Context}(\mathrm{RMP}_t, \mathrm{SP}_t, \mathcal{G}_t)$
    $\mathcal{C}_t \leftarrow \mathrm{ColumnSelection}(\mathcal{G}_t, S_t)$ ▷ *Select columns*
    if at training time and with probability ϵ **then**
      Select a random subset $\mathcal{C}_t$ from $\mathcal{G}_t$ instead
    end if
    Add columns in $\mathcal{C}_t$ to RMP_t and get RMP_{t+1}
    if at training time **then**
      Observe reward $R_t(\mathcal{C}_t)$ and $r_{\Delta,t}(\mathcal{C}_t, a)\ \forall a \in \mathcal{C}_t$
      Store transition $(RMP_t, \mathcal{C}_t, r_t, RMP_{t+1})$ in D
      $(RMP_j, \mathcal{C}_j, r_j, RMP_{j+1}) \leftarrow$ Sample random minibatch of transitions from D
      Perform a gradient descent step w.r.t network parameters θ on $[Q_{\Delta,t}(\mathcal{C}_t, a) - \hat{Q}^*_{\Delta,t}(\mathcal{C}_t, a)]^2$
      Reset $\hat{Q}^* \leftarrow \hat{Q}$ in every C steps
    end if
    Solve RMP_{t+1} to get dual values
    Use dual values to build SP_{t+1}
    $t \leftarrow t + 1$
    $\mathcal{G}_t \leftarrow \mathrm{GetCandidateColumns}(\mathrm{RMP}_t, \mathrm{SP}_t)$
  end while
  if at execution time **then**
    return Optimal solutions from RMP_t, SP_t
  end if
end for
if at training time **then**
  return Trained Q function approximator $\hat{Q}^*$
end if

For the STOP action, the reward is 0 because it does not improve the convergence of CG and does not increase unnecessary computing expenses. For the unselected action $a \in \mathcal{G}_t - \mathcal{C}_t$, if it should be selected ($r_{\Delta,t}(\mathcal{C}_t \cup \{a\}, a) > 0$), then the reward is β, otherwise the reward is $-\beta$.

4.4 Training and Execution

Theoretically, given the total reward $R_t(\mathcal{C}_t)$, we can enumerate all possible $\mathcal{C}_t \subseteq \mathcal{G}_t$ and select the best one. However, this brute-force method is not scalable because the number of $\mathcal{C}_t$ grows exponentially with the number of candidate columns. Therefore, we compute the marginal Q-value of an individual column $Q_{\Delta,t}$ based on $R_t(\mathcal{C}_t)$, and select the suggested columns as shown in Algorithm 1. This is more practical in terms of the computational costs. Unfortunately, the total reward $R_t(\mathcal{C}_t)$ depends on obj_t and $\mathcal{C}'_t$ both of which are unknown before column selection. Therefore, we train a marginal Q-value approximator $\hat{Q}$ for $Q_{\Delta,t}$ with experience replay (Mnih et al. 2015). This is done by minimizing the mean squared loss between $Q_{\Delta,t}(\mathcal{C}_t, a)$ and $\hat{Q}_{\Delta,t}(\mathcal{C}_t, a)$. The FFCG training and execution is shown in Algorithm 2.

Specifically, at the training time, we use a ϵ-greedy strategy to select columns and compute $r_{\Delta,t}(\mathcal{C}_t, a)$ based on $R_t(\mathcal{C}_t)$ after observing obj_t and $\mathcal{C}'_t$. Then, we perform a gradient descent step to update the network parameters θ of the marginal Q-value approximator. During execution, we use the RL agent with the marginal Q-value approximator to select columns according to Algorithm 1. After that, the CG proceeds to the next iteration until it converges.

Note that the key difference between Algorithm 2 and RLCG is the column selection policy. It is replaced by our polynomial-time multiple-column selection algorithm (Algorithm 1) to accelerate CG. The way we train the marginal Q-value approximator is similar to DQN (Mnih et al. 2015) that RLCG uses to learn the Q-function. To some extent, RLCG can be viewed as a special case of our method, when we only select one column in Algorithm 1. Compared to RLCG, the major improvements in our approach are the total and marginal reward design and the selection strategy for multiple columns.

5 Experiments

We evaluate our proposed FFCG on two sets of problems: the CSP and the VRPTW. Both problems are well-known for the linear relaxation effectively solved using CG. Experimental results demonstrate that FFCG outperforms several widely used single-column selection strategies and multiple-column selection strategies. Furthermore, we analyze how FFCG speeds up CG convergence.

5.1 Experimental Tasks

Cutting Stock Problem. The CSP revolves around efficiently dividing standard-sized stock materials, like paper rolls or sheet metal, into specified sizes while minimizing the surplus material that goes to waste. Computationally, it delves into NP-hard territory and can be reduced to the knapsack problem. Given the combinatorial intricacy inherent in the CSP and its formidable array of potential patterns (variables), the CG technique emerges as a pragmatic solution. It adeptly addresses the LP relaxation of the CSP through an iterative approach, bypassing the exhaustive enumeration of all feasible patterns. The detailed formulation is given in Appendix A.

Vehicle Routing Problem with Time Windows. The Vehicle Routing Problem (VRP) involves finding a set of minimum-cost vehicle routes, originating and terminating at a central depot, that together cover a set of customers with known demands. Each customer has a given demand and is serviced exactly once, and all the customers must be assigned to vehicles without exceeding vehicle capacities.

Strategy	$n = 50$		$n = 200$		$n = 750$		$n = 1000$	
	#Itr	Time	#Itr	Time	#Itr	Time	#Itr	Time
Greedy-S	53.10	4.19	147.30	9.47	222.20	16.96	386.14	67.34
MLCG-S	43.48	5.10	145.74	21.78	232.93	31.93	295.09	70.89
RLCG-S	43.21	3.76	152.80	8.77	237.67	16.10	300.86	46.88
Greedy-M	**11.80**	1.34	**35.15**	2.69	**51.13**	5.38	**66.45**	11.51
RLMCS-M	13.87	1.69	45.67	5.59	71.33	12.93	86.50	17.80
FFCG (Ours)	11.81	**1.30**	38.57	**2.51**	56.27	**4.50**	78.86	**9.65**

Table 1: Experimental results on CSP with different size n. It reports the average number of iterations per instance, and the total runtime in seconds (lower is better).

VRPTW is a variation of the VRP where the service at any customer must be started within a given time interval, called a time window. The detailed formulation of VRPTW is given in Appendix B. We use the well-known Solomon benchmark (Solomon 1987) for training and testing. The dataset generation and division are described in Appendix F.

5.2 Hyperparameter Configuration

We meticulously fine-tune the central hyperparameters, α and β, embedded within the reward function (Equation 4). The process encompasses an exhaustive grid spanning 25 potential configurations, from which we sample 14 distinct setups. These configurations are subsequently employed to train the agent, and their efficacy is gauged using a dedicated validation set. Further elaboration on this methodology is available in Appendix E. The specific parameter configuration adopted for addressing both CSP and VRPTW is as follows: $\alpha = 2000$, $\beta = 0.3$.

5.3 Comparison Evaluation

We compare our FFCG with several well-established single-column and multiple-column selection strategies. We select the same number of candidate columns for all column selection strategies. The candidate columns are generated as the 10 columns with the most negative reduced cost from SP. The baseline strategies for comparison are as follows.

Single-column selection strategies:

- **Greedy-S**: the traditional greedy strategy that selects the column with the most negative reduced cost at each step.
- **MLCG-S**: selection strategy using the learned MILP expert in (Morabit, Desaulniers, and Lodi 2021).
- **RLCG-S**: the RL-based single-column selection strategy by Chi et al. (2022).

Multiple-column selection strategies:

- **Greedy-M**: the simple multiple-column selection strategy that selects all candidate columns with the negative reduced costs at each step.
- **RLMCS-M**: the RL-based multiple-column selection strategy which selects 5 columns per iteration (Yuan, Fang, and Song 2024).

Strategy	$n = 50$	$n = 200$	$n = 750$	$n = 1000$
Greedy	117.97	351.52	511.33	664.55
RLMCS	69.37	228.37	356.67	432.50
FFCG	77.99	250.63	377.40	462.73

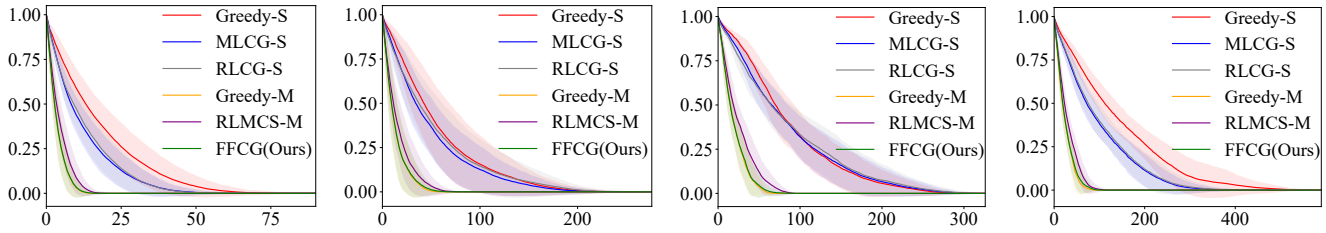
Table 2: Experimental results on CSP with different size n. It reports the average number of columns added, which is only compared between multiple-column selection strategies.

We measure 1) the average number of iterations per instance, 2) the average number of columns added per instance, and 3) the total runtime in seconds, which includes GNN inference and feature computations (if applicable). The node features of GNN are described in Appendix D. For the single-column selections, the number of selected columns is equal to the number of iterations. Thus, we only compare this term for multiple-column selections.

5.4 Experimental Results

Results on CSP. We first train FFCG on CSP instances with the roll length $n = 50, 100, 200$ and the number of item types m varying from 50 to 150. Employing a curriculum learning strategy (Narvekar et al. 2020), FFCG is trained by feeding the instances in order of increasing difficulty. We test FFCG and other compared methods using CSP instances with the roll length $n = 50, 200, 750, 1000$. The detailed dataset information is described in Appendix F.

The comparison results on CSP are reported in Table 1, Table 2, and Figure 2. To illuminate the comparative performance statistically, we visualize the CG-solving trajectories for all test instances with different roll lengths n. We record the objective values of the RMP at each CG iteration for given methods, then take the average over all instances. Note that we normalized the objective values to be in $[0, 1]$ before taking the average among instances. The result shows that FFCG outperforms the prior state-of-the-art methods. Remarkably, despite not being directly trained on CSP instances with roll lengths of 750 and 1000, FFCG demonstrates commendable performance in such challenging scenarios, thereby reflecting its robust generalization capabilities. Compared with Greedy-S, FFCG reduces the number of CG iterations by 77.1% and reduces the total computing



(a) $n = 50$, test instances (b) $n = 200$, test instances (c) $n = 750$, test instances (d) $n = 1000$, test instances

Figure 2: CSP: CG convergence plots for FFCG (green), Greedy-S (red), MLCG-S (blue), RLCG-S (gray), Greedy-M (orange), and RLMCS-M (purple). The solid curves are the mean of the objective values for all instances, and the shaded area shows the standard deviation. The x-axis shows column generation iterations, and the y-axis represents relative objective value.

Strategy	#Itr	Time	#Col
Greedy-S	28.50	2662.07	28.50
RLCG-S	16.50	1789.98	16.50
Greedy-M	4.00	511.97	40.00
FFCG (Ours)	4.33	426.99	24.33

Table 3: Experimental results on VRPTW. It reports the average number of iterations per instance, the total runtime in seconds, and the average number of columns added.

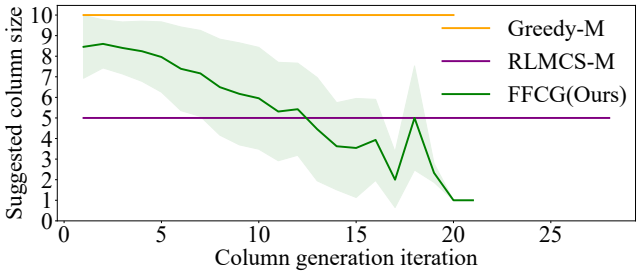


Figure 3: CSP, $n = 50$ test instances: The suggested column size per CG iteration for FFCG, Greedy-M, and RLMCS-M.

time by 71.4% for all test instances on average.

As shown, FFCG performs better than Greedy-M on harder problems. Although selecting all candidate columns reduces the number of iterations, it ends up selecting too many redundant columns and increases the computing cost.

Results on VRPTW. The results are shown in Table 3. Compared with the Greedy-S method, FFCG reduces the number of CG iterations by 84.8% and the total computing time by 84.0%. Notice that we do not compare FFCG with MLCG and RLMCS because they do not design features for VRPTW. The convergence plot is given in Appendix H.

5.5 Analysis

We further discuss why FFCG converges faster than other strategies by examining the distribution of the number of columns selected in each iteration. The cases of selecting all or only one column represent only 30.50% of the total. In major cases, the selection dynamically adjusts between these two extremes. As shown in Figure 3, in the initial rounds, FFCG selects a greater number of columns that contribute to improving the objective function (contributory columns) to accelerate the convergence of CG. The number of contributory columns in the candidate list decreases as the CG processes. At this stage, FFCG adjusts the number of selected columns to avoid selecting redundant columns, which would bring additional computational overhead. As the saying goes, to improve is to change, to be perfect is to change often. In this context, fixing the number of columns selected in each round, as done in RLMCS, is not an effective strategy. In contrast, FFCG dynamically adjusts the number of suggested columns in each iteration, based on the quality of the candidate columns and the columns in the MP.

6 Conclusion and Discussion

In this paper, we propose FFCG, a novel RL-based multiple-column selection framework for CG. Specifically, we model the column selection as an MDP problem. Then, we devise a $O(n^2)$ algorithm to select a subset of candidate columns based on the observed context. Moreover, we learn the marginal Q-value approximator and evaluate the contribution of each column in each time slot. In our experiments on two common benchmarks, CSP and VRPTW, we show that FFCG converges faster both in terms of the number of iterations and computing time compared to several greedy and ML-based baselines. Our findings also indicate that dynamically adjusting the suggested column size in each iteration is a promising approach to accelerate CG convergence.

There are several limitations of our work and we leave them for future work: (1) The reward function that we designed here still depends on hyperparameters α and β, which vary with the problem size. Further study can be done on automating the reward design to make it more suitable for the specific problem. (2) Dual stabilization, heuristic, and metaheuristic methods can be used for further improvement. (3) When the number of variables in the RMP becomes too large, non-basic columns with the current reduced cost exceeding a given threshold may be removed. This will further reduce the computing cost of redundant columns.

Acknowledgments

The research is partially supported by National Key R&D Program of China under Grant 2021ZD0110400, Anhui Provincial Natural Science Foundation under Grant 2208085MF172, Innovation Program for Quantum Science and Technology 2021ZD0302900 and China National Natural Science Foundation with No. 62132018, 62231015, "Pioneer" and "Leading Goose" R&D Program of Zhejiang, 2023C01029, and 2023C01143. We also thank Sijia Zhang, Shuli Zeng, and the anonymous reviewers for their comments and helpful feedback.

References

Babaki, B.; Jena, S. D.; and Charlin, L. 2022. Neural Column Generation for Capacitated Vehicle Routing. In *AAAI-22 Workshop on Machine Learning for Operations Research (ML4OR)*.

Ben Amor, H.; Desrosiers, J.; and Valério de Carvalho, J. M. 2006. Dual-Optimal Inequalities for Stabilized Column Generation. *Operations Research*, 54(3): 454–463.

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.

Chen, L.; Xu, J.; and Lu, Z. 2018. Contextual Combinatorial Multi-armed Bandits with Volatile Arms and Submodular Reward. In *Proceedings of the 31st Conference on Neural Information Processing Systems*. Curran Associates, Inc.

Chi, C.; Aboussalah, A.; Khalil, E.; Wang, J.; and Sherkat-Masoumi, Z. 2022. A Deep Reinforcement Learning Framework for Column Generation. In *Proceedings of the 35th Conference on Neural Information Processing Systems*, 9633–9644. Curran Associates, Inc.

Desaulniers, G.; Desrosiers, J.; and Solomon, M. 2002. Accelerating Strategies in Column Generation for VRCS. *Essays and Surveys in Metaheuristics*, 309–324.

Desaulniers, G.; Desrosiers, J.; and Solomon, M., eds. 2005. *Column Generation*. Springer.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact Combinatorial Optimization with Graph Convolutional Neural Networks. In *Proceedings of the 32nd Conference on Neural Information Processing Systems*. Curran Associates Inc.

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.

Gschwind, T.; and Irnich, S. 2016. Dual inequalities for stabilized column generation revisited. *INFORMS Journal on Computing*, 28(1): 175–194.

Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual. https://www.gurobi.com. Accessed: 2024-01-01.

Haghani, N.; Contardo, C.; and Yarkony, J. 2022. Smooth and flexible dual optimal inequalities. *INFORMS Journal on Optimization*, 4(1): 29–44.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Morabit, M.; Desaulniers, G.; and Lodi, A. 2021. Machine-Learning–Based Column Selection for Column Generation. *Transportation Science*, 55(4): 815–831.

Narvekar, S.; Peng, B.; Leonetti, M.; Sinapov, J.; Taylor, M. E.; and Stone, P. 2020. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *The Journal of Machine Learning Research*, 21(1).

Shen, Y.; Sun, Y.; Li, X.; Cao, Z.; Eberhard, A.; and Zhang, G. 2024. Adaptive Stabilization Based on Machine Learning for Column Generation. In *Proceedings of the 41st International Conference on Machine Learning*, 44741–44758. PMLR.

Solomon, M. M. 1987. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research*, 35: 254–265.

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

Václavík, R.; Novák, A.; Šůcha, P.; and Hanzálek, Z. 2018. Accelerating the Branch-and-Price Algorithm Using Machine Learning. *European Journal of Operational Research*, 271(3): 1055–1069.

Wen, M.; Kuba, J.; Lin, R.; Zhang, W.; Wen, Y.; Wang, J.; and Yang, Y. 2022. Multi-Agent Reinforcement Learning is a Sequence Modeling Problem. In *Proceedings of the 35th Conference on Neural Information Processing Systems*, 16509–16521. Curran Associates, Inc.

Yarkony, J.; Adulyasak, Y.; Singh, M.; and Desaulniers, G. 2020. Data association via set packing for computer vision applications. *Informs Journal on Optimization*, 2(3): 167–191.

Yuan, H.; Fang, L.; and Song, S. 2024. A Reinforcement-Learning-based Multiple-Column Selection Strategy for Column Generation. In *the 38th Annual AAAI Conference on Artificial Intelligence*. AAAI Press.

Yuan, Y.; Cattaruzza, D.; Ogier, M.; Semet, F.; and Vigo, D. 2021. A column generation based heuristic for the generalized vehicle routing problem with time windows. *Transportation Research Part E: Logistics and Transportation Review*, 152: 102391.