Policy optimization in reinforcement learning for column generation

Anonymous authors
Paper under double-blind review

Abstract

Column generation (CG) is essential for addressing large-scale linear integer programming problems in many industrial domains. While its importance is evident, the CG algorithms face convergence issues, and several heuristic algorithms have been developed to address these challenges. However, few machine learning and reinforcement learning methods are available that enhance the existing CG algorithm. This paper introduces a new policy optimization RL framework to improve the existing DQN-based CG framework, particularly training time, called **PPO-CG**. When applied to the Cutting Stock Problems (CSP), our approach requires merely **20%** of the training time observed with the DQN-based method and only **35%** in Vehicle Routing Problems with Time Windows (VRPTW). In addition, our approach suggests a novel method for solving the node selection problem in the framework of reinforcement learning on graphs. Our code is available in this link [1]

1 Introduction

Combinatorial optimization problems are widely applied in various industrial domains such as logistics, telecommunications, and transportation. Solving large-scale optimization problems efficiently is crucial for practical applications. In integer linear programming (ILP), the column generation (CG) technique is commonly used to solve large-scale integer programming problems.

The CG algorithms take advantage of the fact that the optimal solution only needs a part of the entire column. Therefore, it would be inefficient to consider the entire matrix. To begin with, the CG algorithm selects a subset of columns from the Master Problem (MP) and solves the relaxed linear programming problem for the selected columns, called the Restricted Master Problem (RMP). Then, using the dual variable, the CG algorithm solves the Pricing Problem (PP). The solutions of PP are new columns, which have the potential to improve the objective function. The procedure continues until there are no more columns to add.

Despite its usefulness, it is well-known that CG algorithms have convergence issues. Several heuristic algorithms have been developed to address these challenges. For a comprehensive overview, please refer to (Lübbecke & Desrosiers, 2005; Vanderbeck, 2005) and the references within.

With the advancement of machine learning (ML) and reinforcement learning (RL), researchers are increasingly interested in solving combinatorial optimization problems and enhancing existing heuristic algorithms. For a review of this research domain, we direct readers to (Mazyavkina et al., 2021; Bengio et al., 2021; Cappart et al., 2023). Moreocer, recent work by (Berto et al., 2023) presents a unified RL framework for combinatorial optimization problems.

In line with these growing research interests, the application of ML or RL to improve existing heuristic algorithms is gaining significant attention. In (Khalil et al., 2017), the authors leveraged supervised learning to enhance the branch-and-bound heuristic. The research presented in (Tang et al., 2020) employed RL to refine the cutting plane method, yielding performance surpassing that of human-engineered heuristic algorithms. Meanwhile, (Wu et al., 2021) introduced a customized actor-critic approach to improve existing large neighborhood search algorithms.

[1] https://anonymous.4open.science/r/PPO-CG/README.md

Compared with other heuristic algorithms, very little literature exists on CG algorithms employing ML or RL. To the best of our knowledge, (Morabit et al., 2021) is an early attempt to use ML to improve CG. The authors generate multiple columns in each iteration and employ an "expert" system âĂŤrepresented as a mixed integer linear programming (MILP)âĂŤ to supervise the neural network's training on column selection. Datasets are collected from the "expert" before this training process. The authors then encode the RMP into bipartite graphs with column and constant nodes, as introduced in (Gasse et al., 2019). The neural network is trained in a supervised manner to mimic the behavior of the "expert." A limitation of this approach is that solving time-consuming MILP problems is essential in the data collection phase.

The RL approach to CG is first proposed in (Chi et al., 2022), called **RLCG**. In (Chi et al., 2022), the authors follow the methodology from (Morabit et al., 2021), employing the DQN algorithm for node selection. Additionally, they use a GNN as an approximator for the Q-function of the encoded RMP.

We are interested in improving the existing DQN-based approach using different RL algorithms, such as policy-based algorithms. In this work, we provide an improved version of the RL framework for CG utilizing Proximal Policy Optimization (PPO), called **PPO-CG**. In the process, we combine actor and critic networks with GNN. The suggested method can have the potential to be utilized in other problems where the states are represented as a graph, and an action is selecting nodes to include.

We conduct our experiments on two main tasks in the CG algorithms, namely, the Cutting Stock Problems (CSP) and the Vehicle Routing Problems with Time Windows (VRPTW). Our approach, **PPO-CG**, requires only 20% of the training time observed in **RLCG** for the CSP task and 35% for the VRPTW task. Moreover, through the experiment, we find that **RLCG** is not robust and highly dependent on the hyperparameter tuning.

In summary, this paper offers the following contributions:

- Introduces a novel RL framework for column generation that, compared to the DQN-based approach, **RLCG**, achieves comparable performance and significantly reduces training time for both CSP and VRPTW tasks.
- Proposes a new method to integrate GNN with the actor-critic network during CG iterations.
- Presents a potential approach to other RL tasks where states are represented as graphs, and actions are characterized as node selection problems.

This paper is organized as follows. Section 2 discusses related works. Section 3 details the methodology, including the architecture and algorithms. Section 4 presents the experimental setup, including dataset selection and evaluation metrics. Section 5 reports and discusses the conclusions and the limitations.

2 Related Works

2.1 Basic Column Generation

Column generation techniques are widely used to solve large-scale problems. Although our problem focuses on Integer Linear Programming, we shall provide a CG method for linear programming in this subsection.

Let us consider the following MP:

$$\begin{aligned} \min_{x} \ & c^T x \\ \text{s.t. } & Ax \leq b, \quad x \geq 0, \end{aligned} \tag{MP}$$

where the matrix $A \in \mathbb{R}^{n \times m}$, vectors $x, c \in \mathbb{R}^m$ and $b \in \mathbb{R}^n$. If the number of columns m is very large compared to n, let us consider the following RMP

$$\begin{aligned} \min_{x'} \ & (c')^T x \\ \text{s.t. } & A'x' \leq b, \quad x' \geq 0, \end{aligned} \tag{RMP}$$

where $A' \in \mathbb{R}^{n\times m'}$, vectors $x', c' \in \mathbb{R}^{m'}$ with $1 \leq m' \leq m$. Here, The columns of A' are a subset of the columns of A. Denoting a dual variable of equation RMP by $\lambda \in \mathbb{R}^n$, we solve the following pricing problem (PP):

$$\delta_i = c'_i - \sum_j A_{ji}\lambda_j. \tag{PP}$$

If there exists i such that a reduced cost $\delta_i < 0$, then we add i-th column of A to A' and iterate this process until no more column is selected (see Figure 1).

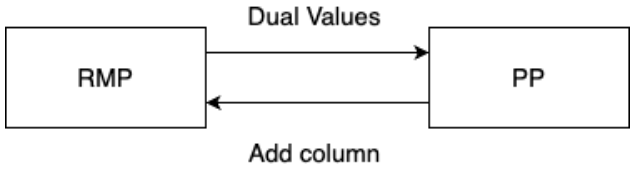

Figure 1: Oveall column generation process

The column generation process for ILP is very similar, with some modifications for each task. For details about ILP formulation and CG formulation of CSP and VRPTW, we refer to Appendix 1 and Appendix 2.

2.2 Proximal Policy Optimization

The DQN algorithms introduced by (Mnih et al., 2015) have achieved remarkable success in various tasks, including Atari games and robotic control. However, their robustness and scalability still require improvement. Moreover, vanilla DQN tends to underperform in environments with continuous or high-dimensional action spaces. See (Lillicrap et al., 2015) and the reference therein for details. Addressing such limitations, the seminal work,(Schulman et al., 2017) introduces the Proximal Policy Optimization (PPO) methods, which have shown promising results across diverse domains. In this work, we apply PPO to the CG iteration. The main intuition is that the action in the CG is choosing the next column to add, which is very high dimensional for the large ILP, and thus, PPO algorithms can work well in this type of problem.

Let us explain the PPO algorithm, and for a general overview of RL algorithms, we refer to (Achiam, 2018). For an action a and a state s, we denote a parameterized policy by $\pi_\theta = \pi_\theta(a|s)$. Let Q^{π_θ} be a on-policy action-value function and V^{π_θ} be a on-policy value function which satisfies

$$V^{\pi_\theta}(S) = \mathbb{E}_{a\sim\pi_\theta}[Q^{\pi_\theta}(s,a)].$$

We then define the advantage function by

$$A^{\pi_\theta}(s,a) := Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s). \tag{1}$$

Let $D = \{\tau\} = \{s_0, a_0, s_1, \cdots, s_T\}$ be a set of trajectories obtained from running policy $\pi_{\theta_{\text{old}}}$. For $t \in \{1, \cdots, T\}$, we denote the ratio function by

$$r_t(\theta) := \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \tag{2}$$

The PPO algorithm is an on-policy updating a parametrized policy by maximizing

$$L^{clip}(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{old}}}\left[\min(A_t^1, A_t^2)\right], \tag{3}$$

where

$$A_t^1 = r_t(\theta)A_t \text{ and } A_t^2 = \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t.$$

Note that $\varepsilon > 0$ is a hyperparameter to be determined later in Section 4. The clip function is defined as

$$\text{clip}(t, t_{\min}, t_{\max}) := \max(t_{\min}, \min(t, t_{\max})),$$

for $t, t_{\min}, t_{\max} \in \mathbb{R}$. For the simplicity of the notation, we also denote A_t by the advantage function with s_t, a_t.

3 Proposed Methods

In this section, we provide our proposed method, which is motivated by the methods provided in (Morabit et al., 2021) and (Chi et al., 2022). The DQN-based approach in (Chi et al., 2022) is called **RLCG**, and our method is named **PPO-CG**.

3.1 MDP formulation

To train a new RL framework, we represent the CG iteration process as the Markov decision process (MDP). Let us denote $\mathcal{S}$ and $\mathcal{A}$ as a state and action space respectively. Then we denote $\mathcal{T} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as a transition map defined as $T(s, s', a) = P(s'|s, a)$. Also, we denote the reward by $reward : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a discount factor by $\gamma \in [0, 1]$.

State $\mathcal{S}$

For each iteration of the CG, we represent the matrix in (RMP) as a bipartite graph composed of column nodes $\mathcal{X}$ and constraint nodes $\mathcal{B}$, as in (Gasse et al., 2019). There exists an edge connecting $(x, b) \in \mathcal{X} \times \mathcal{B}$ if the column contributes to the constraint c (see Figure 2). For each node, we set the node features to be specified in Section 4.

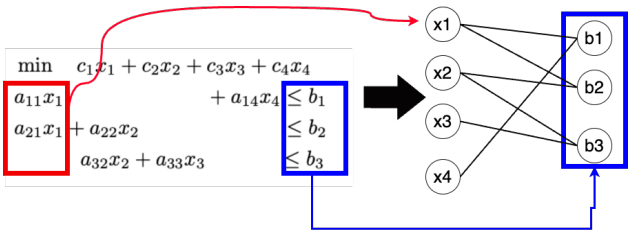

Figure 2: Each column is represented as the node in column nodes in $\mathcal{X}$, and constraints are represented as constraint node class $\mathcal{B}$. If a column x_1 contributes to the constraints b_1, for instance, there exists an edge between x_1 and b_1.

Action $\mathcal{A}$ and Transition $\mathcal{T}$

By solving equation PP, we find candidate columns with negative reduced cost. The action is to choose the next column or node to add to the current RMP or graph (see Figure 3). The maximum number of candidates at each iteration is also a hyperparameter. Since the graph is used to represent equation RMP, we will differentiate for the rest of this paper.

Reward

Our purpose in training the RL network is to get a higher objective function within fewer iterations. Thus, we set the reward as

$$reward_t = \alpha \left(\frac{obj_{t-1} - obj_t}{obj_0} \right) - p,$$

where $\alpha > 0$ and $p > 0$ are hyperparameters. The parameter $p > 0$ is the iteration penalty, giving a negative reward if the model can not finish iteration in fewer iterations.

3.2 Architecture of PPO-CG and overall framework

In this subsection, we introduce the architecture of **PPO-CG** and how it is used to improve the existing CG algorithms.

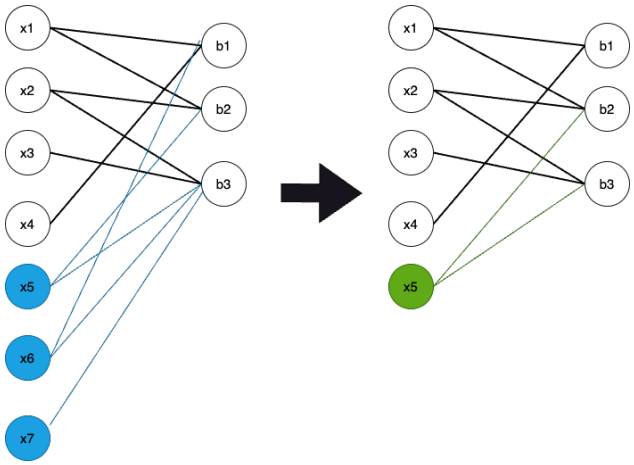

Figure 3: In this example, after PP is solved, there exist three candidates (blue nodes), then choose one node (green node).

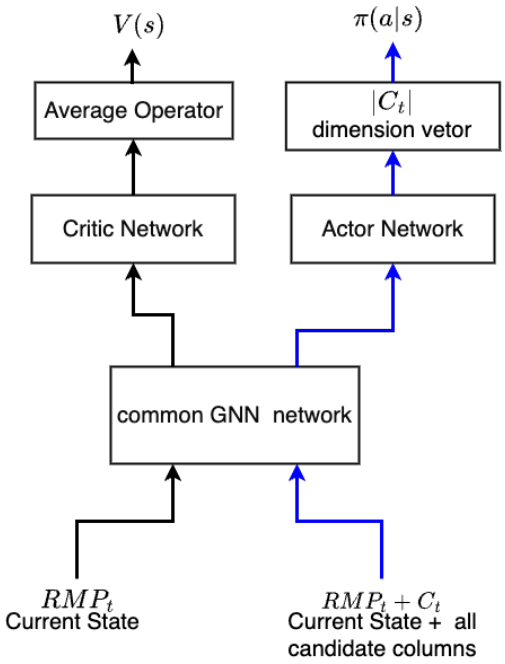

Figure 4: actor and critic network with RMP_t and $RMP_t + C_t$.

Model architecture

We utilize both the actor network and the critic network, which share common layers. The GNN layers introduced in (Morabit et al., 2021) are employed.

Let RMP_t denote equation equation RMP or the corresponding graph. We use $RMP_t + C_t$ to represent the graph of *equation* RMP with all candidate nodes. Moreover, the notation $|C_t|$ implies the number of candidate nodes, and $|RMP_t|$ denotes the number of column nodes of the current state graph.

At each iteration, t, solving equation PP gives $|C_t|$ candidate nodes to choose from. Then, we take RMP_t and $RMP_t + C_t$ as inputs of **PPO-CG**. First, RMP_t goes through a common layer and then the critic network. Next, $RMP_t + C_t$ goes through the common layer and the actor network.

Both the actor and critic networks have the same structures. Both networks return $|C_t|$-dimension vectors. We get the value function $V(s)$ by taking an average of $|C_t|$-dimension vectors. We get a policy $\pi(a|s)$ from the output of the actor network. For an overview of the proposed architecture, see Figure 4.

Since RMP_t and $RMP_t + C_t$ have almost the same structure except for some nodes, using a common layer for the actor and critic network seemed natural.

Overall framework

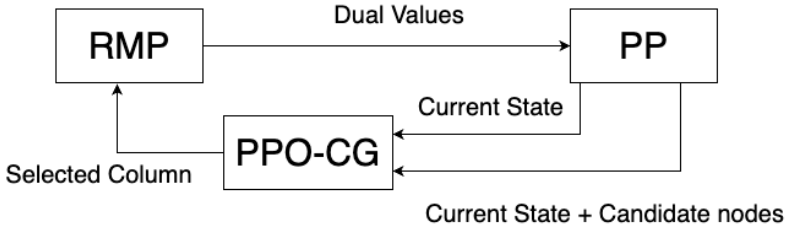

Figure 5: **PPO-CG** framework.

From the model architecture depicted in Figure 4, we now provide an overall framework of **PPO-CG**, summarized in Figure 5.

In the training procedure, for each action, the environment or ILP instance returns a reward. Then collect the trajectory $D = \{\tau\}$ until the CG process is over. Then, update parameters θ that maximizing equation 3. We refer to Algorithm 1 for details.

Algorithm 1 Triaining procedure

```
Initialize policy parameters θ, old policy parameters θ_old
for iteration = 1, 2, ... do
    Initialize ILP problem and environment
    for epoch = 1 to E do
        Collect the trajectories D = {τ_i} by running π_θold
        Compute A_t for each trajectory D
        Compute: r_t(θ) and L^CLIP(θ) .
        Update policy by maximizing L^CLIP(θ).
        Update π_θold ← π_θ
    end for
end for
```

Once training **PPO-CG** is done, then the model only uses the actor network to choose the next column in the CG iteration in Figure 5.

3.3 Comparision with RLCG

Since our architecture is based on **RLCG**, it would be great to point out differences made on **PPO-CG**. In **RLCG**, only the actor network is used. Also, it takes only $RMP_t + C_t$ as input only uses the actor network. The output of the actor network is considered as an action-value function, $Q(s, a)$. Whereas we take both RMP_t and $RMP_t + C_t$, and the output of the actor is considered as the policy $\pi(a|s)$ and the average of the output of critic network is considered as the value function $V(s)$.

4 Experiments

In this section, we outline the details of our experimental process. For specifics, such as the **RLCG** framework hyperparameters, please refer to Appendix 3. To ensure a fair comparison, we adopted the experimental settings from the official implementation of Chi et al. (2022)[2]. Furthermore, for practical reasons, we conducted our experiments on different tasks using different machines. Details are provided in each subsection.

Moreover, we used Tensorflow 2.13 and the free version of Gurobipy 10.0.3.

4.1 CSP tasks

Machine specification

For this task, we use NVIDIA RTX A5000 24GB GPU with Intel(R) Core(TM) i9-10980XE CPU @ 3.00GHz.

Dataset

We train **PPO-CG** and **RLCG** using BPPLIB from (Delorme et al., 2018). In the training process, we use ILP instances with roll length sizes of 50, 100, and 200. The total number of instances is 439. In the test process, we use ILP instances with roll length sizes of 200 and 750 with 86 and 21 instances, respectively. For details, we refer to Appendix 4.

Hyperparameters & node features

We set the learning rate for both the critic network and actor network to be $1e^{-4}$, the hidden dimension of the GNN model is 32, step penalty of $p = 10$, epoch size $E = 20$, $\varepsilon = 1e^{-2}$, action candidate size $= |C_t| = 10$. We set objective hyperparameter $\alpha = 100$ and reward decay exponent $\gamma = 0.999$.

The variable nodes that belong to $\mathcal{X}$ have 9 node features, and constraint nodes that belong to $\mathcal{B}$ have 2 node features. We refer details in Appendix 6.

Traing process

We train our model, **PPO-CG**, with 439 instances in 66.79 **hours**, whereas with the same instances, **RLCG** takes 319.86 **hours**. For the comparison in training time for each instance, see Figure 6. We also provide statistical information about the number of iterations in the training process. See Table 1.

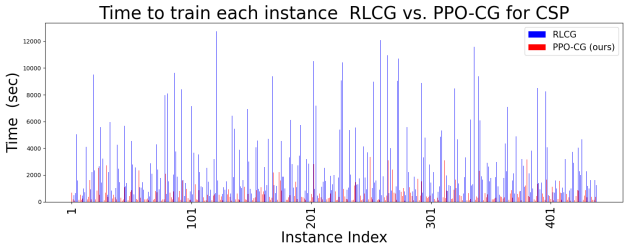

Figure 6: Training time for each instance in CSP

Note that the number of average steps does not have a dramatic change. However, the time to train each model has a dramatic difference since **RLCG** updates its model parameter every step after equation PP is solved, whereas in **PPO-CG**, it updates the parameter after the instance is solved. Therefore **PPO-CG** takes much less time for training.

[2]https://github.com/khalil-research/RLCG/

Table 1: Statisctic information about the number of steps in training CSP

Method	Problem size	μ	σ
PPO-CG	50	37.54	8.25
	100	68.14	14.01
	200	131.06	35.7
RLCG	50	36.3	8.47
	100	67.38	16.57
	200	133.04	41.46

Test results

In this part, we provide a comparison between test results in our model **PPO-CG**, DQN-based model **RLCG**, greedy method, and expert methods. We use three metrics to compare each method: objective function, time to execute, and number of iterations.

First, we provide test results on roll length $n = 200$ and $n = 750$, respectively. Figure 7 and Figure 8 provide comparison results between **PPO-CG**.

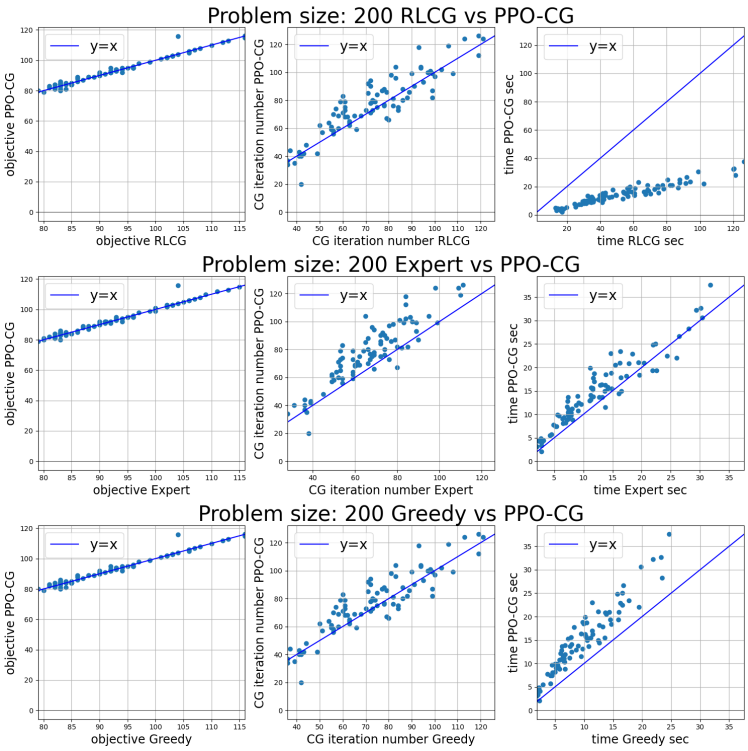

Figure 7: Test results on problem size 200 in CSP

In Figure 9, we provide a bar char for each method with a different size of n.

Based on Figures 7 and 8, there isn't a significant difference in the objective function across various problem sizes and methods. As depicted in Figure 9, the Greedy and Expert methods are more effective when $n = 200$. However, for larger problems,$n = 750$, **PPO-CG** excels in execution time and iteration count. We believe the superior performance of the Greedy algorithm for smaller problem sizes can be attributed to its independence from the GPU. This independence eliminates data transitions between the CPU and GPU, resulting in increased efficiency. Meanwhile, the Expert method performs well when $n = 200$, but as

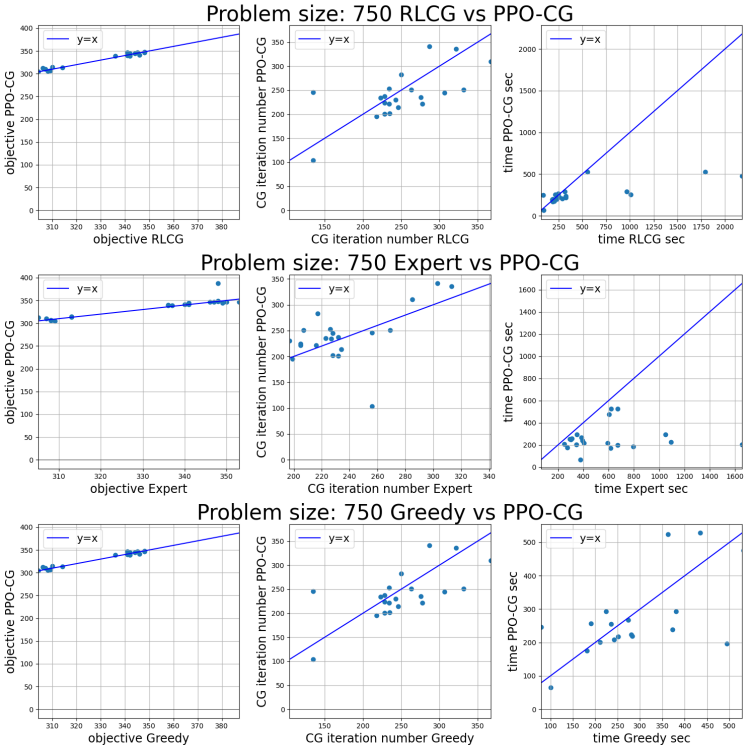

Figure 8: Test results on problem size 750 in CSP

the problem size expands, its execution time grows, making it the slowest among the four methods. This increase is primarily because the Expert method requires solving MILP. For smaller problems, the increment in solving additional MILP aids in reducing execution time and iteration number. However, this benefit diminishes for larger problems. Consequently, for these larger issues, the RL-guided method demonstrates notable improvements with small variances, even with data transition costs between CPU and GPU.

4.2 VRPTW tasks

Machine specification

For this task, we have used NVIDIA GeForce RTX 4090 with AMD Ryzen 9 7950X3D 16-Core Processor.

Dataset

We train **PPO-CG** and **RLCG** using Solomon benchmark from (Solomon, 1987). In the training process, we use ILP instances with different types of problems with various numbers of customers. The total number of instances is 234, but 219 instances are used since the rest 15 instances take too long time to solve. In the test process, we use ILP instances with 37 instances, respectively. For details, we refer to Appendix 4.

Hyperparameters & node features

Many of the hyperparameter settings are similar to CSP. We set the learning rate for both the critic network and actor network to be $1e^{-4}$, the hidden dimension of the GNN model is 32, step penalty of $p = 10$, epoch size $E = 20$, $\varepsilon = 1e^{-2}$. Since the heuristic algorithm for solving equation PP in VRPTW does not create a fixed number of columns, action candidate size $|C_t|$ is not fixed. We set objective hyperparameter $\alpha = 0.5$ as in the official implementation of **RLCG** and the reward decay exponent $\gamma = 0.999$. For hyperparameter settings of **RLCG**, we refer to Appendix 3.

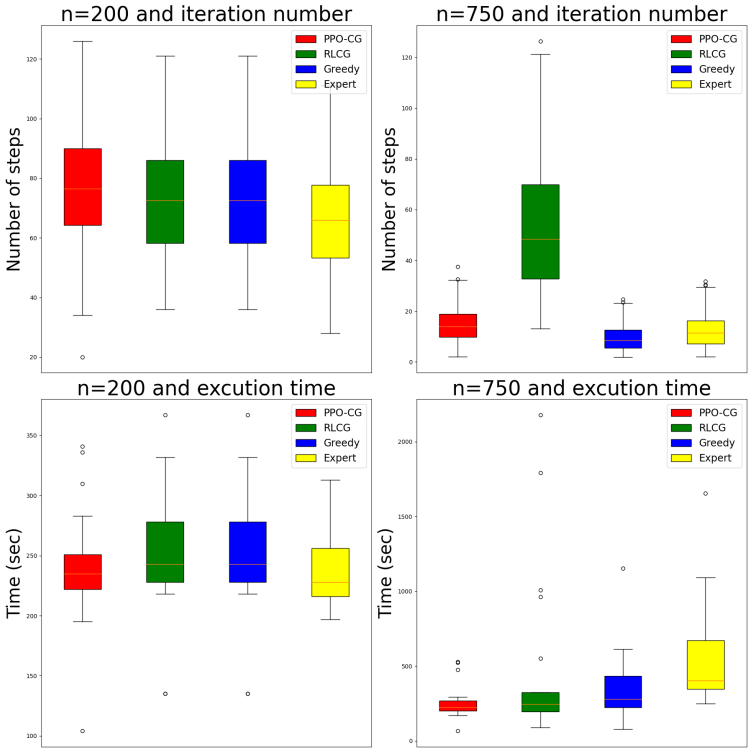

Figure 9: Bar chart for four methods for $n = 200$ and $n = 750$ in CSP

The variable nodes that belong to $\mathcal{X}$ have 8 node features and constraint nodes that belong to $\mathcal{B}$ have 2 node features. For details, we refer to Appendix 6.

Traing process

Our model, **PPO-CG**, trains 219 instances in 16.70 **hours**, whereas RLCG trains the same instance in 47.88 **hours**. For details, please refer to Figure 10. In Table 2, we also provide statistical information about the number of steps taken in the training process.

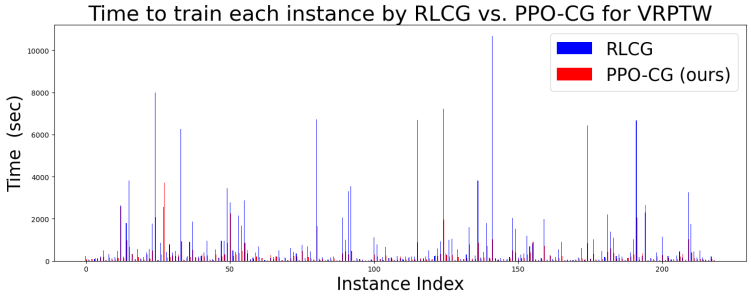

Figure 10: Training time for each instance

We observe that **PPO-CG** has a lower number of average iteration steps.

Remark: additional detail In VRPTW, the heuristic algorithm for solving equation PP takes too much time and does not generate a fixed number of candidate nodes. For these reasons, we use the *argmax* argument to choose the next node as in **RLCG**, and instead of generating E trajectories, we only generate one trajectory and update parameters E times. To our surprise, this method has shorter training time and

Table 2: Statistic information about the number of steps in training VRPTW

Method	μ	σ
PPO-CG	29.33	17.89
RLCG	49.49	28.82

shows almost similar performance results compared with creating E numbers of trajectories. For details, we refer to Appendix 5.

Test results

In this part, we provide a comparison between test results in our model **PPO-CG**, DQN-based model **RLCG**, and greedy method. As in the CSP task, we use three metrics to compare each method: objective function, time to execute, and number of iterations. Figure 11 shows a comparison result for each method with **PPO-CG**. In Figure 11, we excluded an outlier during plotting to show the overall depiction clearly.

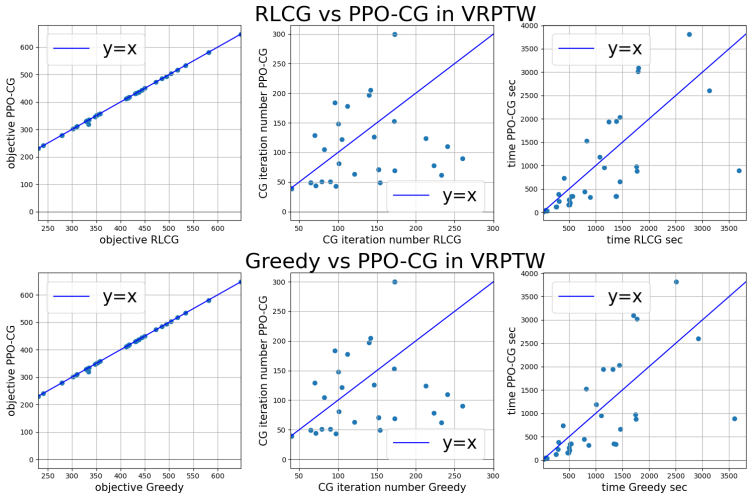

Figure 11: Test results on VRPTW

Additionally, a bar chart is presented in Figure 12 to enhance the clarity of the results further.

For the extreme case is when **PPO-CG** takes 26200.88 seconds to solve with 217 iterations, **RLCG** takes 20138.50 seconds with 153 iterations, and the Greedy algorithm takes 19915.74 seconds with 153 iterations

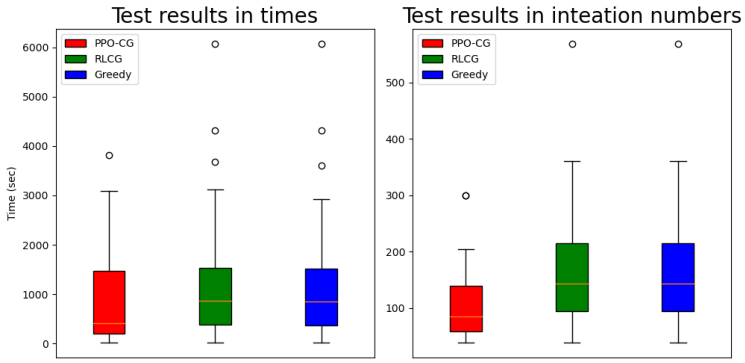

Figure 12: Bar chart for three methods in VRPTW

We can observe that **PPO-CG** performs better than the other two models in terms of time and iteration numbers as problems get more complicated. Also, our model shows a smaller Interquartile Range (IQR) range compared to other models. However, in execution time, **PPO-CG** does not show a smaller IQR, which can be a topic for future study, as well as analysis of the extreme results, as mentioned in the previous paragraph.

4.3 Convergence Analysis

In Figure 13, we provide a convergence analysis of test results on CSP and VRPTW. The objective numbers are scaled from 0 to 1, and we added ± 1 standard deviation. As mentioned earlier, **PPO-CG** tends to converge faster when the problem gets larger and more complicated.

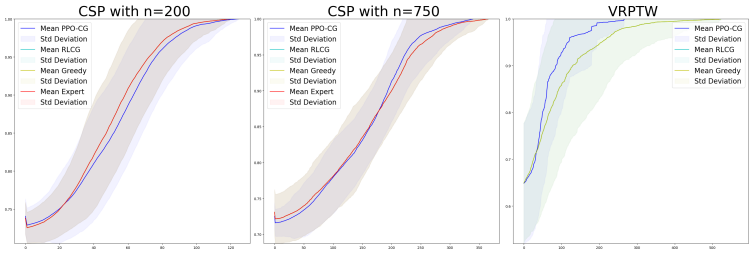

Figure 13: convergence graph on relative objective function

4.4 Remark on the robustness of RLCG and PPO-CG

We want to point out that although the approach suggested in **RLCG** is very interesting, DQN might not be an appropriate approach to improve the CG algorithms on a large scale. For a fair comparison, we do not change many hyperparameters and settings of **RLCG** compared to the official implementation. Moreover, due to very long training hours, hyperparameter-tuning is very challenging. However, we find that **RLCG** tends to behave like a Greedy algorithm if hyperparameter-tuning is not properly done. Whereas our model is very robust given that very little hyperparameter tuning is done, our model still outperforms the Greedy algorithm in many metrics when the problems are complicated enough.

5 Conclusions and Limitations

5.1 Conclusion

In this work, we propose a novel RL framework for improving the CG algorithm. We show that our model trains much faster and it is more robust than the existing DQN-based model developed in (Chi et al., 2022).

Also, we conduct experiments on two main tasks CSP and VRPTW. We find improvement in execution time and iteration number given that the ILP problem is complicated enough.

Beyond the context of CG, the suggested method can have the potential to apply to other tasks such that the state can be represented as a graph and the action behaves like node selection.

5.2 Limitation and future work

Our method still takes much time compared to the increase in performance. The main reason is that this method highly depends on solving equation PP and if solving equation PP takes a long time, inevitably, the training RL model takes a long time which needs further development.

Moreover, since this method is based on GNN, it still needs further improvement to be a more robust model.

Acknowledgements

We thank the anonymous reviewers for their attention and time spent reviewing this manuscript.

References

Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

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

Federico Berto, Chuanbo Hua, Junyoung Park, Minsu Kim, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Joungho Kim, and Jinkyoo Park. Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark. *arXiv preprint arXiv:2306.17100*, 2023.

Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. *J. Mach. Learn. Res.*, 24:130–1, 2023.

Alain Chabrier. Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, 33(10):2972–2990, 2006.

Cheng Chi, Amine Aboussalah, Elias Khalil, Juyoung Wang, and Zoha Sherkat-Masoumi. A deep reinforcement learning framework for column generation. *Advances in Neural Information Processing Systems*, 35: 9633–9644, 2022.

Maxence Delorme, Manuel Iori, and Silvano Martello. Bpplib: a library for bin packing and cutting stock problems. *Optimization Letters*, 12:235–250, 2018.

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

Nasser A El-Sherbeny. Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University-Science*, 22(3):123–131, 2010.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

George Ioannou, Manolis Kritikos, and G Prastacos. A greedy look-ahead heuristic for the vehicle routing problem with time windows. *Journal of the Operational Research Society*, 52(5):523–537, 2001.

Brian Kallehauge, Jesper Larsen, Oli BG Madsen, and Marius M Solomon. *Vehicle routing problem with time windows*. Springer, 2005.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Marco E Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations research*, 53 (6):1007–1023, 2005.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35:254–265, 1987. URL `https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/`.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pp. 9367–9376. PMLR, 2020.

François Vanderbeck. Implementing mixed integer column generation. In *Column generation*, pp. 331–358. Springer, 2005.

Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning large neighborhood search policy for integer programming. *Advances in Neural Information Processing Systems*, 34:30075–30087, 2021.

A Problem description and the ILP formulation of CPS and VRPTW

In this section, we shall provide details on the formulation of the CSP and VRPTW.

CSP

The purpose of the CSP is to cut as few stocks as possible but satisfy the given demand $\{d_j\}_{j=1}^n$ for each item. Suppose that the given length of the stock is L, there are n types of items, and for each item, w_j length is required. The formulation of CSP follows from (Delorme et al., 2018). We denote u by the number of all the cutting patterns of the stock. For each item $j \in \{1, \cdots, n\}$, there are d_j demands. Denoting a_{ji} by the number of item j in the i-th pattern. Let x_i denote the decision variables such that $x_i = 1$ if i-th pattern is chosen, 0 otherwise. Then, we need to minimize the following integer linear programming:

$$\min_x \sum_{i=1}^{u} x_i$$

satisfying the following constraints:

$$\sum_{i=1}^{u} a_{ji} x_i = d_j, \quad \sum_{j=1}^{n} a_{ji} w_j \leq L, \quad x_i \in \{0, 1\} \quad \text{and} \quad a_{ji} \geq 0, \text{ integer.}$$

VRPTW

This section summarizes the context presented in (Kallehauge et al., 2005). Let $\mathcal{C}$ be a set of customers and $\mathcal{G}$ be a directed graph with a number of nodes equal to $|\mathcal{C}| + 2$. We denote the set of vehicles by $\mathcal{V}$. Customers are represented by $1, 2, \cdots, n$, and each node in $\mathcal{G}$ is denoted by $0, 1, \cdots, n+1$. The routing of each vehicle starts at node 0 and ends at node $n+1$. For simplicity, we use the notation $\mathcal{N} = 0, \cdots, n+1$. For every node $i = 1, \cdots, n$, there is a corresponding customer i with a time window $[a_i, b_i]$. Vehicles must arrive at node i within the time window $[a_i, b_i]$. Also, we assume that $[a_0, b_0] = [a_{n+1}, b_{n+1}]$. The matrices c_{ij} and t_{ij} indicate the cost and time, respectively, that a vehicle takes by moving from node i to node j. The matrices c_{ij} and t_{ij}âĂŃ indicate the cost and time, respectively, taken by a vehicle to move from node i to node j. It is assumed that c_{ij} and t_{ij} satisfy Let q denote the capacity of the vehicle, and each customer i has a demand d_i. We want each customer to be served exactly once and satisfy one's demand. To formulate ILP, let us introduce two decision variables, x_{ijk} and s_{ik}. For each $(i, j) \in \mathcal{N} \times \mathcal{N}$ with $i \neq n+1, j \neq 0$ and $i \neq j$, we define

$$x_{ijk} = \begin{cases} 1 & \text{if vehicle } k \text{ move from vertex } i \text{ to vertex } j \text{ directly,} \\ 0 & \text{otherwise.} \end{cases}$$

The other decision variable s_{ik} denotes the time vehicle k starts to serve customer i. In case $i = 0$, we assume that $sik = 0$, since we assume that $a_0 = 0$. If the vehicle k does not serve the customer i, it is an irrelevant variable. The ILP formulation for the VRPTW follows:

$$\min \sum_{k \in \mathcal{V}} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} c_{ij} x_{ijk}$$

such that

$$\begin{aligned}
\sum_{k\in\mathcal{V}}\sum_{j\in\mathcal{N}} x_{ijk} &= 1 \quad \forall i \in \mathcal{C}, \\
\sum_{i\in\mathcal{C}} d_i \sum_{j\in\mathcal{N}} x_{ijk} &\leq q \quad \forall k \in \mathcal{N}, \\
\sum_{j\in\mathcal{N}} x_{0jk} &= 1 \quad \forall k \in \mathcal{V}, \\
\sum_{i\in\mathcal{N}} x_{ihk} - \sum_{j\in\mathcal{N}} x_{hjk} &= 0 \quad \forall h \in \mathcal{C},\ \forall k \in \mathcal{V}, \\
\sum_{i\in\mathcal{N}} x_{i,n+1,k} &= 1 \quad \forall f \in \mathcal{V}, \\
x_{ijk}(s_{ik} + t_{ij} - s + jk) &\leq 0 \quad \forall i,j \in \mathcal{N},\ \forall k \in \mathcal{V}, \\
a_i \leq s_{ik} &\leq b_i \quad \forall i \in \mathcal{N}, \forall k \in \mathcal{V}, \\
x_{ijk} &\in \{0,1\} \quad \forall i,j \in \mathcal{N},\ \forall k \in \mathcal{V}.
\end{aligned}$$

For some discussion regarding VRPTW, we refer to (Kallehauge et al., 2005).

B Remarks of CG iteration in CSP and VRPTW

CSP

The CG process of CSP is very similar, as given in Section 2.1, with minor modifications. We take $b = (d_1, d_2, \cdots d_n)$, and in solving (PP), add constraint, $\sum_{j=1}^{n} w_j A_{ji} \leq L$ with the assumtion that A_{ji} belogns to non negative integer.

VRPTW

Due to its complexity, describing the CG procedure of VRPTW in detail is out of the scope of this paper. However, we remark that we utilize an open source code [3] an experiment done in **RLCG**, which is motivated by the methods provided in (Desrochers et al., 1992; Ioannou et al., 2001; Chabrier, 2006; El-Sherbeny, 2010).

C Hyperparameters in training RLCG

In this section, we provide hyperparameters in training **RLCG**. For the CSP task, we set the learning rate to be $3e-4$, the hidden dimension of the GNN model is 32, step penalty of $p = 10$, epoch size $E = 5$, $\alpha = 100$, buffer size 2000 and $\gamma = 0.999$. For the VRPTW task, we set the learning rate to be $1e-3$, the hidden dimension of the GNN model is 32, step penalty of $p = 10$, epoch size $E = 5$, buffer size 20000, $\alpha = 0.5$ and $\gamma = 0.99$.

D Details in the dataset in the training and testing process

We adopt the procedure outlined in (Chi et al., 2022). However, for the sake of completeness, we provide the details. In the CSP task, we use 160 instances for the length of the roll 50, 160 instances for the length of the roll 100, and 120 instances for the length of the roll 120. As suggested in (Chi et al., 2022), we train both **PPO-CG** and **RLCG** from the easy problems (roll length = 50) to the hard problems (roll length = 200).

For the VRPTW task, we use (Solomon, 1987) with six differnt types of problems, i.e., C1, C2, R1, R2, RC1, RC2. For the training, C1, R1, and RC1 types are used with different sizes of customers from six to eight. In total, there are 240 instances. Due to the time constraints, only 213 instances are used.

[3] https://github.com/SimoneRichetti/VRPTW-Column-Generation/

For more details on the instances, we refer to Appendix 8.

E Comparison between generating a single Trajectory and multiple trajectories in VRPTW Task

Training each instance with only one trajectory takes 16.70 **hours**. On the other hand, generating a new trajectory every time each trajectory ends, the training time extends to 47.22 **hours**. We refer to Figure 14 for performance comparison. There are, in total, 37 instances, and as in the comparison presented in Section 42, we exclude one instance with extreme results for clarity. The objective function is identical, and for the execution time and CG iteration number, **PPO-CG** with multiple trajectories shows better results. This is an expected result, given that training with multiple trajectories takes almost three times compared to training with one trajectory. However, we believe that improvement in training with the new trajectory is minimal compared to the training costs except for a few extreme cases.

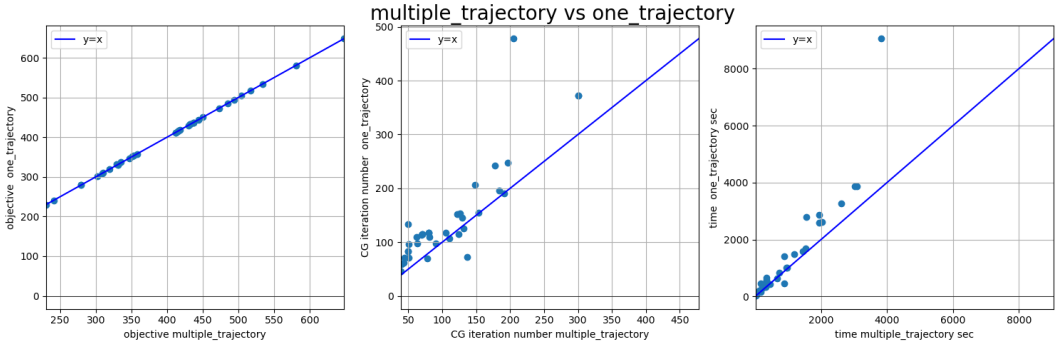

Figure 14: Training time for each instance in CSP

F Node features on CSP and VRPTW tasks

We use node features to encode information about RMP to the graph. as in (Gasse et al., 2019; Chi et al., 2022). In the CSP task, we set 9 node features for the column nodes and 2 node features for the constraint nodes. For the column node features, we use

1. reduced cost for each node,
2. number of connected nodes,
3. solution value of RMP corresponding to each column,
4. remaining length of a roll for each pattern,
5. number of iterations that each column node stays in the basis,
6. number of iterations that each column stays out of the basis,
7. if the column left the basis on the last iteration or not,
8. if the column entered the basis on the last iteration or not,
9. action node or not.

For the constraint node features, we use

1. dual value or shadow price of (PP),

2. the number of connected nodes.

For the VRPTW task, node features are very similar. For the column node features, 4-th and 9-th node features are removed, and routing cost is added. The constraint node features are the same as CSP.

G Remark on training and testing result per instances

Since it is too long to add tables for the training and test results for each instance, we added a supplementary file in the output folder, which will also be updated on the GitHub repository.