# YORDLE: An Efficient Imitation Learning for Branch and Bound

**Qingyu Qu**
Beihang University
quqingyu@buaa.edu.cn

**Xijun Li**
MIRA Lab, USTC
xijunli@miralab.ai

**Yunfan Zhou**
Chinese University of Hong Kong (Shenzhen)
yunfanzhou@link.cuhk.edu.cn

## Abstract

Combinatorial optimization problems have aroused extensive research interests due to its huge application potential. In practice, there are highly redundant patterns and characteristics during solving the combinatorial optimization problem, which can be captured by machine learning models. Thus, the 2021 NeurIPS Machine Learning for Combinatorial Optimization (ML4CO) competition is proposed with the goal of improving state-of-the-art combinatorial optimization solvers by replacing key heuristic components with machine learning techniques. This work presents our solution and insights gained by team qqy in the dual task of the competition. Our solution is a highly efficient imitation learning framework for performance improvement of Branch and Bound (B&B), named YORDLE. It employs a hybrid sampling method and an efficient data selection method, which not only accelerates the model training but also improves the decision quality during branching variable selection. In our experiments, YORDLE greatly outperforms the baseline algorithm adopted by the competition while requiring significantly less time and amounts of data to train the decision model. Specifically, we use only $1/4$ of the amount of data compared to that required for the baseline algorithm, to achieve around $50\%$ higher score than baseline algorithm. The proposed framework YORDLE won the championship of the student leaderboard.

## 1 Introduction

In recent years, the combinatorial optimization problem (COP) has aroused extensive interests from both industry and academia since many real-life critical optimization problems such as scheduling, planning, bin packing, etc. can be formulated as COP. People can benefit a lot from solving these problems. In general, the COP aims to find an optimal configure in discrete spaces, and is one of the most common mathematical topics in industrial applications. Many traditional algorithms are proposed to solve kinds of COPs [1]. Through many years of practice, kinds of COPs and corresponding algorithms have been verified their effectiveness in optimizing real-life problems. In practical scenarios, there are many highly similar COPs which are only slightly different in coefficients. For example, managing a large-scale production planning requires solving very similar COPs on a daily basis, with a fixed logistic network and bill of material (BOM) while only the demand changes over time. This change of demand is hard to capture by hand-engineered expert rules, and ML-enhanced approaches offer a possible solution to detect typical patterns in the demand history, to further enhance traditional combinatorial optimization algorithms. Therefore, there is a trend of using machine learning (ML) techniques to boost above algorithms [2] recently.

Preprint. Under review.

Among many kinds of COPs, we mainly consider the Mixed Integer Linear Programming (MILP) since its wide adoption in practical modeling. The branch-and-bound (B&B) algorithm is placed at the core of solving MILP. B&B algorithm enumerates the candidate solutions systematically by means of state space search, where the set of candidate solutions is considered to form a search tree with the full set at the root. The efficiency of B&B algorithm mainly depends on branching variable selection and node selection. Usually, choosing good variables to branch on can lead to a dramatic reduction in terms of the number of nodes needed to solve an instance. At present, there is still no universally accepted method for the strategy of branching variable selection. Traditional methods are mostly simple heuristic rules, such as the Most infeasible branching, Pseudocost branching (PC), Strong Branching (SB), Hybrid Strong/Pseudocost branching, Pseudocost branching with strong branching initialization, Reliability branching, etc. [1]. Alvarez et al. adopted machine learning algorithms early to learn the strategies of branching variable selection in the B&B algorithm [3]. Such kind of learning-based strategies are also known as learning to branch. Learning to branch is different from the traditional optimization methods. It introduces the concept of learning in the optimization process to help search the optimal solution more effectively. Balcan has shown empirically and theoretically that it is possible to learn high-performing branching strategies for a given application domain [4]. Learning branching policies for MILP has become an active research area. Most relevant researches use supervised or imitation learning to imitate SB method and specialize it to distinct classes of problems.

Supervised machine learning and imitation learning are currently the mainstream approaches for learning to branch. Alvarez et al. proposed a new approach that uses supervised learning to improve the performances of optimization algorithms in the context of MILP [3]. Khalil et al. proposed a machine learning framework for variable branching in MILP. Based on the data collected by SB method, they learned an easy-to-evaluate surrogate function that mimics the SB method, by means of solving a learning-to-rank problem [2]. And it is competitive with a state-of-the-art commercial solver. Gasse et al. proposed a new graph convolutional neural network (GCNN) model for learning to branch, which leverages the natural variable-constraint bipartite graph representation of MILP. They trained the GCNN model via imitation learning from the SB method, and demonstrated that this model produced policies that improved upon state-of-the-art machine learning methods for branching [5]. Gupta et al. proposed a new hybrid architecture for efficient branching on CPU machines, which combined the expressive power of GCNNs with computationally inexpensive multi-layer perceptrons (MLPs) for branching [6]. More related researches can refer to the survey provided by Huang et al [7]. However, *these algorithms often require a large amount of expert data. Considering that there is no commonly accepted expert rules for B&B, improper expert data may be misleading for training.* To address this issue, we propose a learning-based framework to generate high-performance expert data for a given metric.

In this work, we develop a highly efficient imitation learning framework for B&B, named YORDLE [1], which not only greatly accelerates the learning convergence but also improve the quality of branching variable selection. Specifically, the framework consists of four parts, namely data collection, data selection, model learning, and ML-based branching. In the data collection phase, we adopt a hybrid strategy to collect expert data. And then in the data selection phase, we select the state-action pairs that possess higher cumulative reward. In the model learning phase, a graph convolutional neural network is adopted to train a branching strategy. Finally in the ML-based branching phase, the trained branching strategy is evaluated in parallel. The technical contributions are summarized as follows:

- A hybrid sampling method based on pseudo cost and active constraint method is designed to collect data for demonstration. The demonstration data will be taken as input into a graph neural network (GNN) to train a branching policy.
- Before training branching policy, the demonstration data is filtered via a Best-Action Imitation Learning algorithm (BAIL) which selects state-action pairs that possess higher cumulative reward among all state-action pairs within the demonstration data, which results in faster learning convergence.
- YORDLE is tested over practical MILP dataset obtained from Combinatorial Optimization (ML4CO) NeurIPS 2021 competition [8], which greatly outperforms the state-of-the-art imitation learning-based B&B algorithm with respect to dual integral.

[1] YORDLE are a race of spirits in a game named "League of Legends" developed by Riot Games. Usually, YORDLE are much smaller than humans. Furthermore, they are commonly characterized by their creation and utilization of complex tools. In a word, YORDLE are usually small but powerful, which is consistent with the characteristics of our framework.

- In ML4CO competition, we only submit two branching model (for 'Item Placement' and 'Anonymous' dataset respectively) trained using `YORDLE` and the branching model for 'Load Balancing' dataset still adopts the baseline algorithms, which won the championship of the student leaderboard in the dual task.
- After the competition, we continue to test our framework over 'Load Balancing' dataset, which suggests highly competitive performance compared to winner of global leaderboard.

Beside, two key insights are gained during the competition, which are as following:

**Remark 1.** It is commonly believed that the strong branching strategy can lead to the smallest B&B tree. However, when the dual integral is adopted as the metric, the data collected by our proposed framework leads to better result than that collected by strong branching strategy. Therefore, we doubt whether the strong branching strategy is still the 'golden standard'.

**Remark 2.** During the training process, it is found that a smaller cross-entropy loss does not necessarily lead to a better score, which is worth studying in the future.

## 2 Background

### 2.1 Mixed integer linear programs

A mixed integer linear program is an optimization problem of the form

$$\arg\min_{\mathbf{x}} \left\{ \mathbf{c}^T\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{Z}^p \times \mathbb{R}^{n\text{-}p} \right\}$$

where $\mathbf{c} \in \mathbb{R}^n$ is the objective coefficient vector, $\mathbf{A} \in \mathbb{R}^{m\times n}$ is the constraint coefficient matrix, $\mathbf{b} \in \mathbb{R}^m$ is the constraint right-hand-side vector, $\mathbf{l}, \mathbf{u} \in \mathbb{R}^n$ represent the lower and upper variable bound vectors respectively, and $p$ is the number of integer variables. The linear programming (LP) relaxation of a MILP is shown below

$$\arg\min_{\mathbf{x}} \left\{ \mathbf{c}^T\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, \mathbf{x} \in \mathbb{R}^n \right\}$$

The LP solution provides a lower bound to the original MILP. Specifically, if the LP solution is subject to the integer constraint, then it is also a optimal feasible solution of the MILP. Otherwise, the LP relaxation is required to be decomposed into two sub-problems. This is done by branching on a variable that does not obey the integrality constraint in the current LP solution. The solving process terminates when the feasible regions cannot be decomposed anymore, and then a certificate of optimality or infeasibility can be provided respectively.

### 2.2 Branching rules

A key factor influencing the efficiency of B&B algorithm is how to select a fractional variable to branch. In this part, three typical variable selection strategies are briefly described as follows.

The idea of Strong Branching is to evaluate which of the fractional candidate variables gives the best progress before actually branching on any of them. For each candidate variable, this evaluation process is realized by solving the LP relaxations of the two sub-problems. Thus, a huge amount of computation is required when adopting SB method.

Pseudocost branching is a sophisticated rule in the sense that it keeps a history of the success of the variables on which already has been branched [9]. $\Psi_j^+$ ($\Psi_j^-$) denotes the average unit objective gain taken over upwards (downwards) branching on $x_j$ in previous nodes. Pseudocost branching at node $N$ with LP relaxation solution $\check{x}$ consists in computing values:

$$PC_j = score((\check{x}_j - \lfloor \check{x}_j \rfloor)\Psi_j^-, (\lceil \check{x}_j \rceil - \check{x}_j)\Psi_j^+)$$

and choosing the candidate variable with highest such value. Compared with the SB method, PC method is simpler but faster.

Patel proposed an active constraint method that relies on estimating the impact pf the candidate variables on the active constraints in the current LP relaxation [10]. This method aims to find the first feasible solution of MILP as quickly as possible. The goal of the method is to select the branching candidate variable so that the LP relaxation optimum points for the two child nodes are as far apart as possible (in the sense of Euclidean distance). This scheme can lead to significantly different solutions that one of the child nodes will be quite good while the other one is poor. It is accomplished by choosing the candidate variable that most affects the active constraints at the parent node LP relaxation optimum.

## 3 Overview of YORDLE

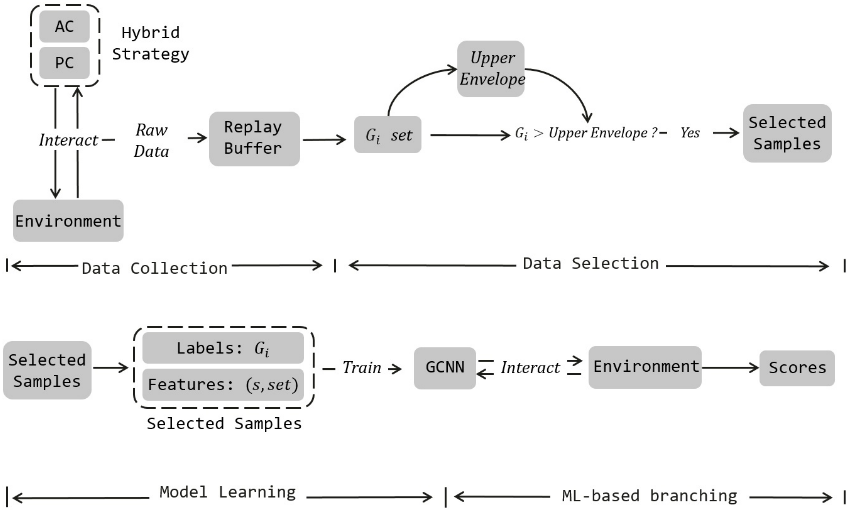


Figure 1: Framework of YORDLE. This framework consists of four phases, including data collection phase, data selection phase, model learning phase and ML-based branching phase. In the data collection phase, a hybrid strategy is adopted to collect expert data. And then in the data selection phase, we select the state-action pairs that possess higher cumulative reward. In the model learning phase, a graph convolutional neural network is adopted to train a branching strategy. Finally in the ML-based branching phase, the trained branching strategy is evaluated in parallel.

In our work, a framework, named YORDLE, is introduced for learning to branch based on BAIL. And it is proceeded in the following four phases:

1. In the data collection phase, we collect the data with a hybrid strategy based on PC and AC strategies. Then the data is sent to the next phase.

2. In the data selection phase, we develop a data selection method based on BAIL, and select the state-action pairs that possess higher cumulative reward. These selected state-action pairs are provided to the network model as expert demonstrations.

3. In the model learning phase, similar to the Gasse's work, a graph convolutional neural network (GCNN) is adopted to exploit the natural bipartite graph representation of MILP problems. Differently, the criteria for generating candidate branching variables is cumulative reward, instead of instant reward.

4. In the ML-based branching phase, a parallel evaluation platform is proposed to accelerate the instance evaluation process. Besides, inspired by the reinforcement learning, the evaluation criterion of the best strategy is chosen to be the cumulative reward rather than the cross-entropy loss.

The framework is shown in Figure 1. Next, we describe each of the phase in detail.

### 3.1 Data collection

In this phase, a hybrid strategy is adopted to collect the data to form a training dataset. The setting of the hybrid strategy is as follows in order to increase the coverage of the dataset.

$$strategy = \begin{cases} PC & (db \leq DB_0, r \leq R_0) \text{ or } (db > DB_0, r > R_0) \\ AC(r_1, r_2, r_3, r_4) & (db \leq DB_0, r > R_0) \text{ or } (db > DB_0, r \leq R_0) \end{cases}$$

where *db* represents the value of dual bound of the instance, $DB_0$ represents a certain value of dual bound, $R_0 \in (0,1)$ represents a sampling probability, $r, r_1, r_2, r_3, r_4 \in [0,1]$ represent random quantities.

At each node $N_i$, the training data comprises [11]:

- Observation: A node bipartite graph representation of B&B states used in Gasse et al., using the ecole.observation.NodeBipartite observation function. On one side of that bipartite graph, nodes represent the variables of the problem, with a vector encoding features of that variable. On the other side of the bipartite graph, nodes represent the constraints of the problem, similarly with a vector encoding features of that constraint. An edge links a variable and a constraint node if the variable participates in that constraint, that is, its coefficient is nonzero in that constraint. The constraint coefficient is attached as an attribute of the edge.
- Action set: The set of candidate variables.
- Action: The selected candidate variable to branch on.
- Reward: The reward is defined as the dual integral since the previous state, where the integral is computed with respect to the solving time.
- Next observation: The node bipartite graph representation of the next node.
- Next action set: The set of candidate variables corresponding to the next node.
- Done: The termination flag.

To demonstrate the framework of YORDLEin detail, those variables and sets are denoted by certain symbols, which are shown in Table 1:

Table 1: Meaning of Symbols

| Symbols | Meaning |
|---|---|
| $obs$ | observation |
| $set$ | action set |
| $a$ | action |
| $r$ | reward |
| $obs'$ | next observation |
| $set'$ | next action set |
| $d$ | done |

### 3.2 Data selection

In this phase, a data selection method based on BAIL algorithm [12] is developed to obtain a better training dataset. The details are described as follows.

A training dataset has been obtained in the data collection phase, which is denoted by $\mathcal{B} = \{(obs_i, set_i, a_i, r_i, obs'_i, set'_i, d_i), i = 1, ..., m\}$, where subscript $i$ is a stamp to identify data points. For each data point $i \in \{1, ..., m\}$, we calculate the cumulative reward from the current state $(obs, set)$ to the end of the episode, which is shown as

$$G_i = \sum_{t=i}^{T} \gamma^{t-i} r_t$$

where *T* represents the step at which the episode ends for the episode that contains the $i_{th}$ data point, $\gamma$ represents the discount factor. This process is demonstrated in Figure 2.

In order to calculate the cumulative reward for a state $(obs, set)$, all the paths starting from $(obs, set)$ are required to be given. One way to do this is to put all states into a buffer and traverse them to get all possible paths. However, regarding the B&B problem, the dimensionality of the state $(obs, set)$ is so large that it will take up too much storage resources. This is detrimental to our framework. To address the problem, we introduce the following function that maps the state $(obs, set)$ to a low-dimensional vector $s$.

$$f_{dim}(obs, set) = s$$

where $f_{dim}$ ensures that $(obs, set)$ and $s$ are in one-to-one correspondence.

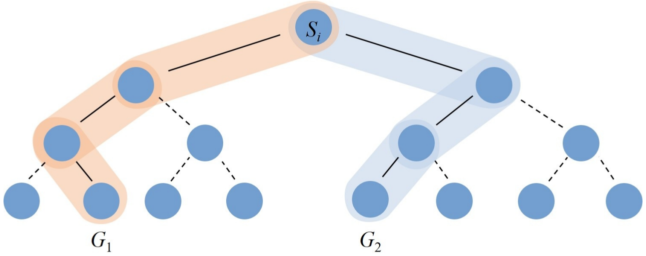


Figure 2: Cumulative Reward of the Forward Trace. All states $(obs, set)$ are firstly mapped to some low-dimensional states $s_i$. And then all the obtained low-dimensional states $s_i$ are stored into a buffer and traversed to get all possible paths. Then for each state, the cumulative rewards corresponding to all the paths starting from this state are calculated and finally the state-action pairs that possess higher cumulative reward are chosen for training.

For a fixed $\lambda \geq 0$, denote $\mathcal{G} = \{(obs_i, set_i, G_i), i = 1, ..., m\}$. Let $V_\phi(obs, set)$ denote a graph convolutional neural network characterized by $\phi = (w, b)$ that takes $obs_i$ and $set_i$ as input and outputs a real number to fit the cumulative reward. Then $V_{\phi^\lambda}(obs, set)$ is regarded as a $\lambda$-regularized upper envelope for $\mathcal{G}$ if $\phi^\lambda$ is an optimal solution for the following constrained optimization problem:

$$\min_\phi \sum_{i=1}^{m} [V_\phi(obs_i, set_i) - G_i]^2 + \lambda \|w\|^2 \quad s.t. \quad V_\phi(obs_i, set_i) \geq G_i, i = 1, ...m$$

In the work of Chen et al.[11], an unconstrained optimization problem with a penalty loss function (with $\lambda$ fixed) is introduced to obtain an approximate upper envelope of the data $\mathcal{G}$, which is shown as

$$L^K(\phi) = \sum_{i=1}^{m} (V_\phi(obs_i, set_i) - G_i)^2 \{\mathbf{1}_{(V_\phi \geq G_i)} + K \cdot \mathbf{1}_{(V_\phi < G_i)}\} + \lambda \|w\|^2$$

where $K \gg 1$ represents the penalty coefficient, $\mathbf{1}_{()}$ represents a indicator function. For a certain finite *K*, the penalty loss function will lead to an approximate upper envelope $V_\phi(obs_i, set_i)$.

Then we select all $(obs_i, set_i, a_i)$ pairs from the batch data set $\mathcal{B}$ such that

$$G_i > xV_\phi(obs_i, set_i)$$

where *x* is set such that the top $p\%$ of the data points are selected, where *p* is a hyper-parameter. In this paper, *p* is set as 15.

### 3.3 Model learning

In this work, the imitation learning is applied to learn the policy from the selected data set $\mathcal{D}$. Specifically, it is trained by minimizing the cross-entropy loss

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{(obs_i, set_i, a_i) \in \mathcal{D}} \log \pi_0(a \mid obs_i, set_i)$$

Then a graph convolutional neural network [5] is adopted to parametrize the candidate variable selection policy. In detail, the input of the GCNN model is the bipartite state representation $s_t =$

$(\mathcal{G}, \mathbf{C}, \mathbf{V}, \mathbf{E})$. The graph convolution can be broken down into two successive passes, one from variables to constraints and one from constraints to variables, which are shown as

$$c_i \leftarrow f_c(c_i, \sum_{j}^{(i,j)\in\varepsilon} g_c(c_i, v_j, e_{i,j}))$$

$$v_i \leftarrow f_v(v_j, \sum_{j}^{(i,j)\in\varepsilon} g_v(c_i, v_j, e_{i,j}))$$

where $f_c$, $f_v$, $g_c$ and $g_v$ are two-layer perceptrons with ReLU activation functions.

Different with the work of Gasse et al. [5], we produce the probability distribution over the candidate branching variables (i.e., the non-fixed LP variables) based on the cumulated rewards of each candidate variable, instead of the strong branching score. In this way, we can obtain a more effective and non-myopic policy.

### 3.4 ML-based branching

In the evaluation stage, the trained GCNN model is used to replace the expert strategy to make decision while executing branch and bound. The environment will run a full-fledged branch-and-cut algorithm with SCIP, and the trained model will only control the solver's branching decisions. Also, all primal heuristics will be deactivated, so that the focus is only on proving optimality via branching. In each instance of MILP problem, the metric to evaluate a branch and bound algorithm is dual integral, which is expressed as follow:

$$T\mathbf{c}^T\mathbf{x}^* - \int_{t=0}^{T} \mathbf{z}^* dt$$

where $\mathbf{z}^*$ is the best dual bound at time t, and $T\mathbf{c}^T\mathbf{x}^*$ is an instance-specific constant that depends on the optimal solution $\mathbf{c}^T\mathbf{x}^*$. The dual integral is to be minimized, and takes an optimal value of 0. Equivalently, the cumulative reward, which is given by a constant minus the dual integral, is to be maximized in our task. Intuitively, Figure 3 shows the dual integral. Considering that the valid set includes a large number of instances, a parallel evaluation mechanism is introduced to accelerate the evaluation process. However, despite the shortened evaluation time, it also misleads our experimental results to some extent, as will be explained in the experiment.

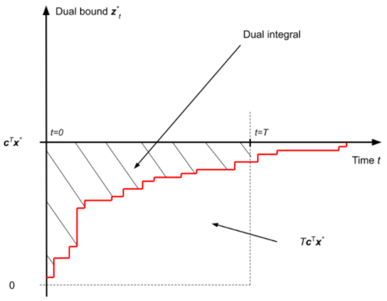


Figure 3: Dual Integral of a Instance

# 4 Evaluation

### 4.1 Competition results

In the competition, the performance of submitted models are evaluated in three problem benchmarks from diverse application areas, namely item placement, load balancing and anonymous problem. Each of the benchmark dataset consists of many MILP instances, which are split into two distinct collections: train and valid. The cumulative reward of each task is given by performing the model in a hidden test datasets. We submitted YORDLE and named our team as qqy.

Due to the time constraint of this competition, we only implement `YORDLE` over the dataset of 'Item Placement' and 'Anonymous'. It takes a lot of time to collect data on the load balancing problem, and it is too late for us to implement our framework on the dataset 'Load Balancing' because designing this framework consumed a lot of our time. In fact, we adopt the method proposed by Gasse et al. [5] on the dataset 'Load Balancing'. Therefore, in this part, we only care about the competition results on the 'Item Placement' and 'Anonymous'. And some discussions about the 'Load Balancing' will be given in the next part.

Table 2 shows the scores of the top 10 leading teams in global leaderboard for the 'Item Placement' and 'Anonymous', alongside with the ranks of them. Table 3 shows the scores of the top 5 leading teams in student leaderboard for the 'Item Placement' and 'Anonymous', alongside with the ranks of them. It is demonstrated from the result that `YORDLE` is effective on the 'Item Placement' and 'Anonymous' problems.

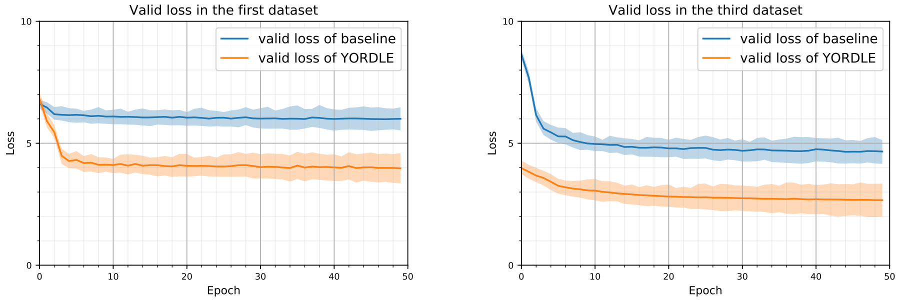
Table 2: Results on the 'Item Placement' and 'Anonymous' Problems in Global Leaderboard

| item placement | | | anonymous | | |
|---|---|---|---|---|---|
| team name | cum. reward | rank | team name | cum. reward | rank |
| Nuri | 6684.00 | 1 | Nuri | 27810782.42 | 1 |
| EI-OROAS | 6670.30 | 2 | **qqy** | **27221499.03** | **2** |
| EFPP | 6487.53 | 3 | null_ | 27184089.51 | 3 |
| lxj24 | 6443.55 | 4 | EI-OROAS | 27158442.74 | 4 |
| ark | 6419.91 | 5 | DaShun | 27151426.15 | 5 |
| **qqy** | **6377.23** | **6** | KEP-UNIST | 27085394.46 | 6 |
| KAIST_OSI | 6196.56 | 7 | lxj24 | 27052321.48 | 7 |
| nf-lzg | 6077.72 | 8 | THUML-RL | 26824014.00 | 8 |
| Superfly | 6024.20 | 9 | KAIST_OSI | 26626410.86 | 9 |
| Monkey | 5978.65 | 10 | Superfly | 26373350.99 | 10 |

Table 3: Results on the 'Item Placement' and 'Anonymous' Problems in Student Leaderboard

| item placement | | | anonymous | | |
|---|---|---|---|---|---|
| team name | cum. reward | rank | team name | cum. reward | rank |
| lxj24 | 6443.55 | 1 | **qqy** | **27221499.03** | **1** |
| ark | 6419.91 | 2 | null_ | 27184089.51 | 2 |
| **qqy** | **6377.23** | **3** | lxj24 | 27052321.48 | 3 |
| KAIST_OSI | 6196.56 | 4 | THUML-RL | 26824014.00 | 4 |
| nf-lzg | 6077.72 | 5 | KAIST_OSI | 26626410.86 | 5 |

Besides, compared with the method proposed by Gasse et al. [5], it can be observed in Figure 4 that our method converges faster and possesses smaller valid loss, which makes it easier to train. Therefore, we believe that it is a small but efficient imitation learning framework for learning to branch.



(a) Item Placement

(b) Anonymous

Figure 4: Valid Loss on the Item Placement and Anonymous Problems

### 4.2 Supplementary Experiment

After the competition, we completed the training of YORDLE on the dataset 'Load Balancing'. Considering that our device may be different with that of the competition, we evaluate the cumulative reward of both our submitted strategy (i.e. qqy) and the one trained by means of YORDLE on the local device, which are shown in Table 4. We consider the following assumptions to be reasonable.

$$\frac{R_{ldb}}{R_{loc}} = CONSTANT$$

where $R_{ldb}$ and $R_{loc}$ represent the cumulative reward of a strategy on the leaderboard and the local device respectively. In this way, it can be inferred that the cumulative reward of YORDLE on the leaderboard is about 630793.45. In fact, the strategy trained by means of YORDLE can be ranked $6_{th}$ in the global leaderboard and ranked $2_{nd}$ in the student leaderboard. Table 5 shows the scores of the top 10 leading teams in global leaderboard for the 'Load Balancing' problem, alongside with the ranks of them. Table 6 shows the scores of the top 5 leading teams in student leaderboard for the 'Load Balancing' problem, alongside with the ranks of them. It is demonstrated from the result that YORDLE is effective on the dataset 'Load Balancing' as well.

Table 4: Evaluation Results of qqy and YORDLE on the Local Device

| official cum. reward of qqy | local cum. reward of qqy | local cum. reward of YORDLE |
|---|---|---|
| 630557.31 | 630629.31 | 630865.48 |

Table 5: Results on the 'Load Balancing' Problem in Global Leaderboard

| team name | cum. reward | rank |
|---|---|---|
| EI-OROAS | 631744.31 | 1 |
| KAIST_OSI | 631410.58 | 2 |
| EFPP | 631365.02 | 3 |
| DaShun | 630898.25 | 4 |
| blueterrier | 630826.33 | 5 |
| YORDLE | **630793.45** | **6** |
| Nuri | 630787.18 | 7 |
| gentlemenML4CO | 630752.94 | 8 |
| comeon | 630750.66 | 9 |
| Superfly | 630746.96 | 10 |

Table 6: Results on the 'Load Balancing' Problem in Student Leaderboard

| team name | cum. reward | rank |
|---|---|---|
| KAIST_OSI | 631410.58 | 1 |
| YORDLE | **630793.45** | **2** |
| comeon | 630750.66 | 3 |
| qqy | 630557.31 | 4 |
| ark | 630414.45 | 5 |

Also, the valid loss is given in Figure 5. It can be observed that our method converges faster and possesses smaller valid loss as well on the 'Load Balancing' problem.

## 5 Conclusion

In this paper, we introduce our framework YORDLE, which achieved the most excellent result in student leaderboard of the ML4CO competition. Generally, YORDLE adopts an imitation learning method, which leverages a hybrid expert strategy to collect data and selects state-action pairs with higher cumulative reward to train a GCNN as the branch and bound policy. YORDLE is flexible to be extended and easy to implement while requiring shorter time and fewer data to train. During the competition, we also have several observations. First, the existing heuristic method solving MILP problem may be not the best solution. Especially, the strong branch rule, which is commonly believed to be the 'golden standard', is inferior to the sampling rule we used in this competition. Last, one of

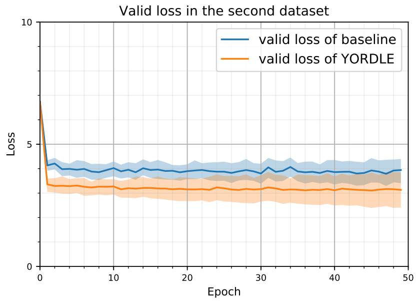


Figure 5: Valid Loss on the ‘Load Balancing’ Problems

the bottlenecks to implement a machine leaning method on branch and bound is the representation capability of the model, especially when the dataset is large. This conclusion is drawn from the observation that ML methods might not outperform the random algorithm when learning from the large dataset. Normally, extending the size of the model helps to improve the representative ability. However, it also leads to a slower solving process. Thus, we have to deal with a trade off between efficiency and effectiveness.

## References

[1] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[3] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A supervised machine learning approach to variable branching in branch-and-bound. In *IN ECML*, 2014.

[4] MF Balcan, T Dick, T Sandholm, and E Vitercik. Learning to branch. In *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, July 2018. ACM.

[5] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*, 2019.

[6] Prateek Gupta, Maxime Gasse, Elias B Khalil, M Pawan Kumar, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *arXiv preprint arXiv:2006.15212*, 2020.

[7] Lingying Huang, Xiaomeng Chen, Wei Huo, Jiazheng Wang, Fan Zhang, Bo Bai, and Ling Shi. Branch and bound in mixed integer linear programming problems: A survey of techniques and trends. *arXiv preprint arXiv:2111.06257*, 2021.

[8] NeurIPS 2021 Competition. Machine learning for combinatorial optimization, `https://www.ecole.ai/2021/ml4co-competition/`,, 2021.

[9] M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1:76–94, Dec 1971.

[10] Jagat Patel and John W Chinneck. Active-constraint variable ordering for faster feasibility of mixed integer linear programs. *Mathematical Programming*, 110(3):445–474, 2007.

[11] Antoine Prouvost, Justin Dumouchelle, Lara Scavuzzo, Maxime Gasse, Didier Chételat, and Andrea Lodi. Ecole: A gym-like library for machine learning in combinatorial optimization solvers. *arXiv preprint arXiv:2011.06069*, 2020.

[12] Xinyue Chen, Zijian Zhou, Zheng Wang, Che Wang, Yanqiu Wu, and Keith Ross. Bail: Best-action imitation learning for batch deep reinforcement learning. *arXiv preprint arXiv:1910.12179*, 2019.