A GNN-GUIDED PREDICT-AND-SEARCH FRAMEWORK FOR MIXED-INTEGER LINEAR PROGRAMMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixed-integer linear programming (MILP) is widely employed for modeling combinatorial optimization problems. In practice, similar MILP instances with only coefficient variations are routinely solved, and machine learning (ML) algorithms are capable of capturing common patterns across these MILP instances. In this work, we combine ML with optimization and propose a novel predict-and-search framework for efficiently identifying high-quality feasible solutions. Specifically, we first predict the solution distributions, then we search for the best feasible solution within a properly defined ball around the predicted solution. We show that our framework is both theoretically and computationally superior to fixing strategies. We conduct extensive experiments on four public datasets and numerical results demonstrate that our proposed framework achieves 51% and 9% better primal gaps than state-of-the-art general-purpose optimization solvers SCIP and Gurobi, respectively.

1 INTRODUCTION

Mixed-integer linear programming is one of the most widely used techniques for modeling *combinatorial optimization* problems, such as production planning Pochet & Wolsey (2006); Chen (2010), resource allocation Liu & Fan (2018); Watson & Woodruff (2011), and transportation management Luathep et al. (2011); Schöbel (2001). In real-world settings, MILP models from the same application share similar patterns and characteristics, and such models are repeatedly solved without making uses of those similarities. ML algorithms are well-known for its capability of recognizing patterns Khalil et al. (2022), and hence they are helpful for building optimization algorithms. Recent works have shown the great potential of utilizing learning techniques to address MILP problems. The work of Bengio et al. (2021) categorized ML efforts for optimization as (i) *end-to-end learning* Vinyals et al. (2015); Bello* et al. (2017); Khalil et al. (2022), (ii) *learning to configuring algorithms* Bischl et al. (2016); Kruber et al. (2017); Gasse et al. (2022) and (iii) *learning alongside optimization* Gasse et al. (2019); Khalil et al. (2016); Gupta et al. (2020). For the sake of interest, we only focus on the *end-to-end* approach. While such an approach learns to quickly identify high-quality solutions, it generally faces the following two challenges:

- **(I) high sample collection cost**. The supervised learning task for predicting solutions is to map from the instance-wise information to a high-dimensional vector. Such a learning task becomes computationally expensive since it necessitates collecting a considerable amount of optimal solutions (see, e.g., Kabir et al. (2009)).
- **(II) feasibility**. Most of the end-to-end approaches directly predict solutions to MILP problems, ignoring feasibility requirements enforced by model constraints (e.g. Yoon (2022); Nair et al. (2020)). As a result, the solutions provided by ML methods could potentially violate constraints.

We propose a novel *predict-and-search* framework to address aforementioned challenges. In principle, end-to-end approaches for MILP problems require the collection of abundant optimal solutions. However, the cost of collecting such training samples is generally impractical since obtaining optimal solutions is excessively time-consuming. A possible approach is to learn distributions of solutions as proposed by Nair et al. (2020), but it still demands sufficient solutions for calculations. Hence, we consider to approximate the distribution by weighing near-optimal solutions with their

corresponding objective values. This reduces the cost of sample collections by avoiding gathering optimal solutions as mentioned in challenge **(I)**. Regarding challenge **(II)**, we implement a *trust region* inspired algorithm that searches for near-optimal solutions within the intersection of the original feasible region and a properly defined ball centered at a prediction point. The overall framework is outlined in Figure 1.

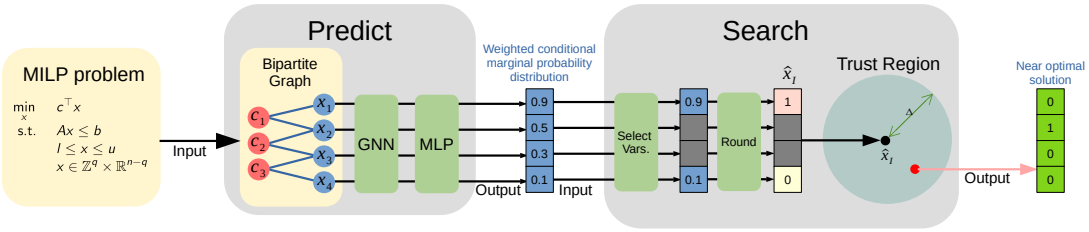


Figure 1: Our approach first predicts the weighted conditional marginal probability of each variable utilizing a *graph neural networks* (GNN) model with graph convolution and multi-layer perceptron (MLP) modules, and then searches for near optimal solutions to the original MILP problem within a well defined trust region.

The commonly used *end-to-end* approaches usually directly fix variables by the prediction, such as in Nair et al. (2020) and Yoon (2022). However, such approaches could lead to sub-optimal or even infeasible sub-problems. Rather than forcibly fixing variables, our search module looks for high-quality solutions in a sub-set of the original feasible region, which allows better feasibility while maintaining optimality.

The distinct contributions of our work can be summarized as follows.

- We propose a novel predict-and-search framework that first trains GNNs to predict the weighted conditional marginal probability of each variable and then constructs a trust region to search for high quality feasible solutions.
- We demonstrate the ability of our proposed framework to provide equivalently good or better solutions than fixing-based end-to-end approaches.
- We conduct comprehensive computational studies on several public benchmarks datasets and the computational results show that our proposed framework achieves 51% and 9% smaller primal gaps than state-of-the-art general-purpose optimization solvers SCIP and Gurobi, respectively.

We make our code publicly available at https://anonymous.4open.science/r/predict-and-search-0F6F

2 Preliminaries

Given a vector $v \in \mathbb{R}^n$ and an index set $I \subseteq \{1, 2, ..., n\}$, let $v_I \in \mathbb{R}^{|I|}$ denote a subvector that corresponds to I.

2.1 Mixed-integer linear programming

MILP techniques are used to model combinatorial optimization problems, and an MILP instance can be formulated as: $\min_{x \in D} c^\top x$, where $D \equiv \{x \in \mathbb{Z}^q \times \mathbb{R}^{n-q} : Ax \leq b, l \leq x \leq u\}$ denotes a set of feasible solutions. There are n variables, with $c, l, u \in \mathbb{R}^n$ being their objective coefficients, lower and upper bounds, respectively. Without loss of generality, the first q variables are discrete. $A \in \mathbb{R}^{m \times n}$ denotes the coefficient matrix while $b \in \mathbb{R}^m$ represents the right hand side vector. For convenience, let $M \equiv (A, b, c, l, u, q)$ denote an MILP instance. For the sake of interest, we only consider discrete variables to be binary, i.e. $x_i \in \{0, 1\}$ for $1 \leq i \leq q$. Besides, considering continuous variables will not change either the methodology or the outcome of our approach.

2.2 NODE BIPARTITE GRAPH

Gasse et al. (2019) proposed a bipartite graph representation for MILP problems. Specifically, let $G \equiv (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} \equiv \{v_1, ..., v_n, v_{n+1}, ..., v_{n+m}\}$ denotes the set of n variable nodes and m constraint nodes, and $\mathcal{E}$ represents the set of edges that only connect between nodes of different types. Variable nodes and constraint nodes are individually associated with the instance information e.g. degrees and coefficients (see appendix). The adjacency matrix $\mathcal{A}$ is a $|\mathcal{V}| \times |\mathcal{V}|$ matrix that represents the connectivity of G, as shown by Equation (1).

$$\mathcal{A} \equiv \begin{bmatrix} 0 & C^\top \\ C & 0 \end{bmatrix} \text{ where } C_{i,j} = \mathbb{1}_{A_{i,j} \neq 0}. \tag{1}$$

2.3 GRAPH NEURAL NETWORKS

Let $N(v_i) \equiv \{v_j \in \mathcal{V} : \mathcal{A}_{i,j} \neq 0\}$ denote the set of neighbors of node v_i. We construct a k-layer GNN as follows:

$$h_{v_i}^k = f_2^k \left(\left\{ h_{v_i}^{(k-1)}, f_1^k \left(\left\{ h_u^{(k-1)} : u \in N(v_i) \right\} \right) \right\} \right),$$

where function f_1^k aggregates the feature information over the set of neighboring nodes and function f_2^k combines the nodes' hidden features from iteration $(k-1)$ with the aggregated neighborhood features, and $h_{v_i}^k$ denotes the hidden state of node v_i in the k^{th} layer. Initially, $h_{v_i}^0$ is the output of an embedding function $g(\cdot)$. Note that the bipartite graph associated with an MILP instance does not include edges between variable nodes as well as constraint nodes. In order to enable information aggregation across all nodes, we use two interleaved half-convolutions Gasse et al. (2019) to replace the graph convolution. After that, the GNN we used can be formulated as:

$$\begin{aligned} h_{v_i}^{(k)} &= \text{MLP}_c^{(k)} \left(h_{v_i}^{(k-1)}, \sum_{u \in N(v_i)} h_u^{(k-1)} \right) \quad i \in \{n, n+1, ..., n+m-1, n+m\}, \\ h_{v_i}^{(k)} &= \text{MLP}_v^{(k)} \left(h_{v_i}^{(k-1)}, \sum_{u \in N(v_i)} h_u^{(k-1)} \right) \quad i \in \{1, 2, ..., n\}, \end{aligned} \tag{2}$$

where $\text{MLP}_c^{(k)}$, $\text{MLP}_v^{(k)}$ and $g(\cdot)$ are 2-layer perceptrons with ReLU activation functions. That is, a half-convolution is performed to promote each constraint node aggregating information from its relevant variable nodes; after that, another one is performed on each variable node to aggregate information from its relevant constraint nodes and variable nodes.

2.4 TRUST REGION METHOD

The trust region method is designed for solving non-linear optimization problems as follows:

$$\min_{x \in H} f(x), \tag{3}$$

where $H \subseteq \mathbb{R}^n$ denotes a set of feasible solutions. The main idea is to convert the originally difficult optimization problem into a series of simple local search problems. Specifically, it iteratively searches for trial steps within the neighborhood of the current iterate point Yuan (2015). The trial step d_k is obtained by solving the following trust region problem:

$$\begin{aligned} \min_{d \in H_k} \quad & \widetilde{f}_k(d), \\ \text{s.t.} \quad & \|d\|_{W_k} \leq \Delta_k \end{aligned} \tag{4}$$

where $\widetilde{f}_k(d)$ is an approximation of the objective function $f(x_k + d)$, H_k denotes a shifted $H - x_k$ of the set H, $\|.\|_{W_k}$ is a norm of $\mathbb{R}^n$, and Δ_k denotes the radius of the trust region. After solving the problem in the k^{th} iteration, x_{k+1} is updated to $x_k + d_k$ or x_k accordingly. Then, new $\|.\|_{W_{k+1}}$ and Δ_{k+1} are selected along with a new approximation $\widetilde{f}_{k+1}(d)$.

3 PROPOSED FRAMEWORK

The supervised learning task in end-to-end approaches maps the instance-wise information to a high-dimensional vector. Ding et al. (2020) predicts optimal solutions based on a GNN that learns the values of variables that stay unchanged across collected solutions. However, such a group of variables does not necessarily exist for many combinatorial optimization problems. Another approach is to learn distributions rather than directly learning solution mappings; Li et al. (2018) predict a set of probability maps for variables, and then utilize such maps to conduct tree search for maximum independent set problems. This work, while obtained remarkable results, is designed for specific tree search algorithm, while we aim at developing a general method that is suitable for most of the MILP problems. A more general case is raised by Nair et al. (2020) to learn the conditional probability distribution in the solution space of an MILP instance; guided by a neural network, a subset of variables is fixed to reduce the problem size. The fixing strategy can potentially accelerate the solving process, however, it assigns values to variables without explicitly considering constraints, such that the resulting sub-problem becomes infeasible.

To alleviate these issues, we adopt the idea of trust region method and design a novel approach that searches for near-optimal solutions within a properly defined region. Specifically, we propose a predict-and-search framework that: (i) predicts weighted conditional marginal probabilities of all binary variables in a given MILP instance by a trained GNN model; (ii) searches for near-optimal solutions within the trust region constructed from the prediction.

3.1 PREDICT

In this part, we aim at training a GNN by supervised learning to predict weighted conditional marginal probability distribution for MILP instances. To this end, we define the conditional probability distribution learning. On this basis, we present the training label in the form of a vector, i.e. weighted conditional marginal probabilities.

3.1.1 DISTRIBUTION LEARNING

A probability distribution learning model outputs the conditional probability distribution on the entire solution space of an MILP instance M. A higher conditional probability is expected when the corresponding solution is more likely to be optimal. Nair et al. (2020) proposed a method to construct the probability distribution with objective values, and for a solution x, the conditional probability $p(x|M)$ can be calculated by:

$$op(x;M) \equiv \frac{\exp(-E(x;M))}{\sum_{x'}\exp(-E(x';M))}, \quad \text{where } E(x;M) \equiv \begin{cases} c^\top x & \text{if } x \text{ is feasible,} \\ +\infty & \text{otherwise.} \end{cases} \tag{5}$$

This implies that an infeasible solution leads to a probability of 0, while the optimal solution induces the highest probability value. Note that each instance corresponds to only one distribution, hence, the training task transforms the one-to-many learning into one-to-one learning.

For the collected dataset $\{(M^i, L^i)\}_{i=1}^N$, $L^i \equiv \{x^{i,j}\}_{j=1}^{N_i}$ denotes the set of N_i feasible solutions to instance M^i. The probability of each solution in the dataset can be calculated by Equation (5). In general, the distance between two distributions can be measured by Kullback-Leible divergence. Thus, the loss function for the supervised learning task is defined as:

$$L(\theta) \equiv -\sum_{i=1}^{N}\sum_{j=1}^{N_i} w^{i,j}\log P_\theta(x^{i,j};M^i), \quad \text{where } w^{i,j} \equiv \frac{\exp(-{c^i}^\top x^{i,j})}{\sum_{k=1}^{N_i}\exp(-{c^i}^\top x^{i,k})}. \tag{6}$$

$P_\theta(x^{i,j};M^i)$ is the prediction from the GNN denoted as F_θ with learnable parameters θ. The conditional probability distribution of an MILP problem can be approximated by a part of the entire solution space. Consequently, The number of samples to be collected for training is remarkably reduced.

3.1.2 WEIGHT-BASED SAMPLING

To further investigate the learning target and align labels with outputs, we propose a vector form of the label. With the learning task specified in Equation (6), a new challenge arises that high-

dimensional sampling for solution acquiring is computationally prohibitive. A common technique is to reduce the dimension by decomposing the high-dimensional distribution into low-dimensional distributions. Given an instance M, let x_d denote the d^{th} dimension of a solution x, Nair et al. (2020) assume that variables are independent of each other, i.e.,

$$P_\theta(x; M) = \prod_{d=1}^{n} p_\theta(x_d; M). \tag{7}$$

With this assumption, the high-dimensional sampling problem is decomposed into n 1-dimensional sampling problems for each x_i according to their probabilities $p_\theta(x_i|M)$. Since $p_\theta(x_i = 1; M) = 1 - p_\theta(x_i = 0; M)$, we only need $p_\theta(x_i = 1; M)$ for $i \in \{1, 2, ..., n\}$ to represent the conditional probability $P_\theta(x; M)$. Then the conditional probability distribution mapping outputs a $n-$dimension vector $(p_\theta(x_1 = 1; M), ..., p_\theta(x_n = 1; M))$. Hence, the prediction of the GNN model can be represented as $F_\theta(M) \equiv (\hat{p}_1, \hat{p}_2, ..., \hat{p}_n)$, where $\hat{p}_d^i \equiv p_\theta(x_d = 1; M)$ for $d \in \{1, 2, ..., n\}$.

Let $S_d^i \subseteq \{1, 2, ..., N^i\}$ denote the set of indices in L^i with their d^{th} component being 1.

$$p_d^i \equiv \sum_{j \in S_d^i} w^{i,j}, \tag{8}$$

where p_d^i is normalized by $|L_i|$ to a value between 0 and 1. Given an MILP instance M, we can calculate a corresponding learning target in the form of vector, i.e, $P \equiv (p_1, p_2, ..., p_q)$, where each component is calculated by Equation (8). This equation calculates the weighted conditional marginal probability, where the weight is 1 if the variable holds a value of 1 in the corresponding solution and vice versa. We define such a learning target in the form of a vector as *weighted conditional marginal probability distribution*. When the weighting coefficients w^{ij} are the same, each individual solution contributes equally to the loss function; when larger weighting coefficients are given to high-quality feasible solutions, this means they play more important roles in the loss function.

For the loss function shown in Equation (6), based on the assumption in Equation (7) and the calculation of probabilities in Equation (6), we have:

$$\begin{aligned} L(\theta) &= -\sum_{i=1}^{N}\sum_{d=1}^{n}\sum_{j=1}^{N_i} w^{i,j} \log p_\theta(x_d^{i,j}; M^i) \\ &= -\sum_{i=1}^{N}\sum_{d=1}^{n}\left\{\sum_{j \in S_d^i} w^{i,j} \log p_\theta(x_d^{i,j}; M^i) + \sum_{j \notin S_d^i} w^{i,j} \log p_\theta(x_d^{i,j}; M^i)\right\} \\ &= -\sum_{i=1}^{N}\sum_{d=1}^{n}\{p_d^i \log(\hat{p}_d^i) + (1 - p_d^i)\log(1 - \hat{p}_d^i)\}. \end{aligned}$$

This indicates that the loss function can be derived as approximating the weighted conditional marginal probability of each component in a cross-entropy format. Thus, with assumption (7), the conditional probability distribution learning is converted to a weighted conditional marginal probabilities learning.

3.2 SEARCH

With weighted conditional marginal probabilities as inputs, we adopted a trust region like method to carry out a search algorithm. In this section, we first introduce our observation that the distance between the solution obtained from a rounded learning target and the optimal solution can be very insignificant. Finally, we adopt a trust region like method to address such an issue and present a proposition to manifest the superiority of our framework. The complete framework can be found in Algorithm 1.

3.2.1 OBSERVATION

A variable's weighted conditional marginal probability is closer to 1 if it's more likely to be 1 in the optimal solution, and vice versa. Given an MILP instance M and its learning target P, we set the

partial solution size parameter (k_0, k_1) to represent the numbers of 0's and 1's in a partial solution. Let I_0 denote the set of indices of the k_0 smallest components of P, and I_1 denote the set of indices of the k_1 largest components of P. If $d \in I \equiv I_1 \cup I_0$, we get a partial solution $\overline{x}_I$ by:

$$\overline{x}_d \equiv \begin{cases} 0 & \text{if } d \in I_0, \\ 1 & \text{if } d \in I_1. \end{cases} \tag{9}$$

Let x^* denote an optimal solution to M, empirically, we found $\overline{x}_I$ is close to x_I^* as discussed in Section 5.2. Explicitly, we define the distance by ℓ_1 norm, and there still exists a small $\Delta > 0$, such that $\|\overline{x}_I - x_I^*\|_1 < \Delta$ while $(k_0 + k_1)$ is a large number. We speculate that, since the optimal solution has the largest weight as shown in equation (8), it is closer to the learning target than all other solutions. As a result, we hypothesize that similar phenomena can be observed with a slightly larger Δ and the same (k_0, k_1) when obtaining the set of indices I based on prediction $F_\theta(M)$.

With this observation, it is reasonable to accelerate the solving process for MILP problems by fixing variables in the partial solution. Specifically, the sub-problem of an instance M using the fixing strategy with the partial solution $\overline{x}_I$ can be formulated as:

$$\min_{x \in D \cap S(\overline{x}_I)} c^\top x, \tag{10}$$

where $S(\overline{x}_I) \equiv \{x \in \mathbb{R}^n : x_I = \overline{x}_I\}$. However, once $\overline{x}_I \neq x_I^*$, the fixing strategy may well lead the above sub-problem to sub-optimal and even infeasible, which can also be observed in Figure **??**.

3.2.2 Search within a trust region

Under the above observation and analysis, we design a more practicable method. Inspired by the trust region method, we use the partial solution as the starting point to establish a trust region and search for a trial step, and then the trial step can be applied to the starting point to generate a new solution.

Specifically, given instance M, we acquire the set I via the prediction $F_\theta(M)$ from a trained GNN, and get a partial solution $\hat{x}_I$ by equation (9) to construct the trust region problem for a trail step d^* similar to problem (3), (4). At last, output the point updated by trail step. Such a trust region problem is equivalent to following:

$$\min_{x \in D \cap B(\hat{x}_I, \Delta)} c^\top x, \tag{11}$$

where $B(\hat{x}, \Delta) \equiv \{x \in \mathbb{R}^n : \|\hat{x}_I - x_I\|_1 \leq \Delta\}$. In order to reduce computational costs, we solve this problem only once to find a near-optimal solution. The pseudo-implementation of such an algorithm can be found in the appendix. We show that our proposed method always outperforms fixing-based methods under a fixed method of variable selection in Proposition 1.

Proposition 1. *let z^{Fix} and z^{Search} denote optimal values to Equation (10) and (11) respectively. $z^{Search} \leq z^{Fix}$, if they have same partial solution $\hat{x}_I$.*

Proof. *Note that $S(\hat{x}_I) = B(\hat{x}_I, 0) \subset B(\hat{x}_I, \Delta)$, it is obvious*

$$\min_{x \in D \cap B(\hat{x}_I, \Delta)} c^\top x \leq \min_{x \in D \cap S(\hat{x}_I)} c^\top x,$$

i.e. $z^{Search} \leq z^{Fix}$

□

This proposition demonstrates the advantage of our proposed search strategy over fixing strategy; that is, when fixing strategies lose optimality or feasibility as a consequence of inappropriate fixing, applying such a trust region search approach can always add flexibility to the sub-problem. Empirically, results given in computational studies also support this proposition.

Algorithm 1 details the search algorithm of the proposed framework. It takes the prediction $F_\theta(M)$ as input, and acquires a partial solution x_d and neighbor constraints. A complete solution x is then attained by solving the modified instance M' with neighbor constraints added. The solving process is denoted by $SOLVE(M')$, i.e. using SCIP/Gurobi to solve instance M'.

Algorithm 1 Search Algorithm

Parameter: size $\{k_0, k_1\}$, Radius of the Area: Δ
Input: Instance M, Probability prediction $F_\theta(M)$
Output: Solution $x \in \mathbb{R}^n$

Sort the components in $F_\theta(M)$ from smallest to largest to obtain sets I_0 and I_1.
for $d = 1 : n$ **do**
  if $d \in I_0 \cup I_1$ **then**
    create binary variable δ_d
    if $d \in I_0$ **then**
      create constraint $x_d \leq \delta_d$
    else
      create constraint $1 - x_d \leq \delta_d$
    end if
  end if
end for
create constraint $\sum_{d \in I_0 \cup I_1} \delta_d \leq \Delta$
Let M' be M with added constraints and variables
Let $x = SOLVE(M')$
return x

4 COMPUTATIONAL STUDIES

We conduct extensive experiments on four public datasets with fixed testing environments to ensure the fairness of comparisons. Detailed model structures are also introduced in this section.

Benchmark Problems Four MILP benchmark datasets are considered in our computational studies. Two minimizing problems come from the ML4CO 2021 competition Gasse et al. (2022), including the *Balanced Item Placement* (denoted by `IP`) dataset and the *Workload Appointment* (denoted by `WA`) dataset (Gasse et al., 2022). Then we generated two maxmization problems: *Independent Set* (`IS`) and *Combinatorial Auction* (`CA`) using Ecole library (Prouvost et al., 2020) referencing instances used by Gasse et al. (2019). *Set Covering* and *Capacitated Facility Location* are not chosen since they can be easily solved by Gurobi and SCIP. Detailed dimensions of selected datasets can be found in the appendix.

Graph neural networks A single layer perceptron embeds feature vectors so they have the same dimension of 64, and the layer normalization Ba et al. (2016) is applied for a better performance of the network. Then we adopted 2 half-convolution layers from Gasse et al. (2019) to conduct information aggregation between nodes. Finally, weighted conditional marginal probabilities are obtained by feeding the aggreated variable nodes into a 2-layer perceptron followed by a sigmoid activation function.

Training protocol Each dataset contains 400 instances, including 240 instances in the training set, 60 instances in the validation set, and 100 instances in the test set. All reported numerical results are running in the test set. Our model is constructed with the PyTorch framework, and the training process runs on GPUs. The loss function is specified in Equation (6) and (7) with a batch size of 8. The ADAM optimizer Kingma & Ba (2014) is used for optimizing the loss function with a start learning rate of 0.003.

Evaluation metrics We use the primal objective values obtained by running single-thread default Gurobi for $3,600$ seconds as the performance baseline. To measure the performance of a given approach, we calculate the average objective values (OBJ) obtained by the approach within a certain amount of time across instances of different datasets. For each instance, we compare objective values obtained by different approaches, and choose the best one as the best known solution (BKS). With that, the performance gap can be formulated as: $\mathrm{gap}_{\mathrm{avg}} \equiv |\mathrm{OBJ} - \mathrm{BKS}| \,/\, \left(|\mathrm{BKS}| + 10^{-10}\right)$. We compare our approach with SCIP and Gurobi by measuring their performance gaps from the BKS, and compute the improvement ratio. Similarly, we can assess the performance gap $\mathrm{gap}_{\mathrm{p}}$

between any two approaches by replacing BKS and OBJ with their average primal gaps, i.e. $\mathrm{gap_p} \equiv |\mathrm{OBJ_a} - \mathrm{OBJ_b}| \,/\, \left(|\mathrm{OBJ_a}| + 10^{-10}\right)$ for approach a and b.
Evaluation Configurations For detailed configurations, please check the appendix.
Data collection Details will be provided in the appendix.

5 Results and Discussion

In order to investigate the benefits of applying such a predict-and-search framework, we carried out comprehensive computational studies that: (1) compare our framework with SCIP and Gurobi; (2) measure the distance between rounded label and the optimal solution to support Proposition 1; (3) demonstrate the advantages of our approach over fixing based methods. The rest of the comparison experiments (e.g. against Neural Diving (Nair et al., 2020)), and implementation details can be found in the appendix. For the sake of simplicity, we denote the predict-and-search framework as PS in Figures and Tables.

Table 1: Average objective values given by different approaches at 1,000 seconds.

dataset	BKS	SCIP		PS+SCIP		$\mathrm{gap_p}$	Gurobi		PS+Gurobi		$\mathrm{gap_p}$
		obj.	gap	obj.	gap		obj.	gap	obj.	gap	
IP	12.02	19.43	7.41	15.46	3.45	**0.53**	12.65	0.64	12.71	0.69	-0.07
WA	700.94	704.23	3.29	702.35	1.41	**0.57**	701.24	0.30	701.22	0.28	**0.01**
IS	685.04	684.94	0.10	685.03	0.01	**0.90**	685.04	0.00	685.04	0.00	0.00
CA	23,680.01	23,671.66	8.39	23,671.95	8.11	**0.03**	23,635.07	44.98	23,654.47	25.59	**0.43**
AVG						50.8%					9.3%

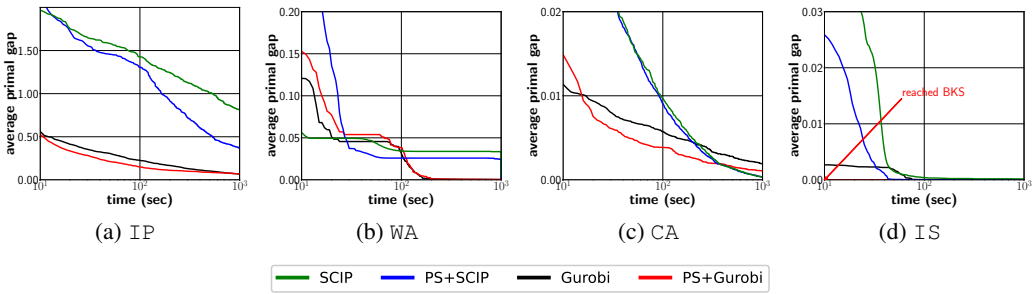
5.1 Comparing against state-of-the-art solvers

We implement the predict-and-search approach with both SCIP and Gurobi to ensure the fairness. Figure 2 exhibit in line plots the tendency of average primal gap as the solving process goes on. In Figure 2a, we observed a huge performance improvement of our framework (blue) comparing to the default SCIP (green), and such an improvement also exists for WA, CA and IS datasets as shown in 2b, 2c and 2d. Changing the under-laying solver does not dramatically diminish the performance; in Figure 2a, 2c and 2d, our approach (red) also outperforms Gurobi (black), and it is remarkable that, in Figure 2d, the predict-and-search framework obtained optimal solutions to IS problems within 10 seconds. Our framework exposed a performance drawback in the early solving stage as shown in Figure 2b, but the performance is quickly restored within 100 seconds. We present detailed objective values for a time limit of $1,000$ seconds in Table 1, and the objective value gaps are calculated, along with improvement ratios. Larger gap means better performance. Our framework outperforms SCIP and Gurobi with improvements of 50.8% and 9.3%, respectively. We noticed that the enhancement for Gurobi is not as significant as for SCIP; it is due to the fact that Gurobi can quickly obtain high-quality solutions.

5.2 Comparing to fixing strategies

In this part, we compare our framework with fixing strategies to show the advantages. We directly use the learning target to define confidence scores, and we obtain partial solutions by setting different size parameters (k_0, k_1) as stated in Section 3.2.1. ℓ_1 norms are computed to measure such distances. We selected $(700, 0)$, $(750, 0)$, $(800, 0)$, $(850, 0)$, and $(900, 0)$ as size parameters and obtained their corresponding ℓ_1 distances as 0.00, 0.04, 1.20, 35.26, and 85.01. We observed that such distances can be insignificant if only a small portion of variables are taken into consideration. However, as we enlarge the number of involved variables, the distance increases dramatically. Hence, the predict-and-search approach can involve larger set of variables than fixing strategy does while still retaining the optimality.

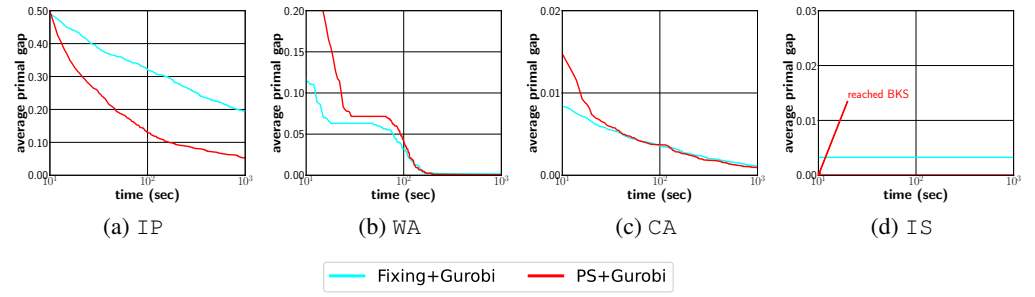
To testify this argument, we directly compare the performance of the fixing strategy and our approach. Figure 3 shows the average primal gap achieved by different approaches, where the numbers



(a) IP (b) WA (c) CA (d) IS

Figure 2: Performance comparisons between PS, Gurobi and SCIP, where the APG is averaged across 100 instances; each plot represents one benchmark dataset. PS implemented with different solvers are denoted as PS+Solver. The result shows a generally better performance of the proposed framework.

of selected variables are fixed for both approaches. One can spot the dominance of our approach over fixing strategy from Figure 3a; this implies that fixing strategy only leads to mediocre solutions, while the search method (red) achieves much better qualities. In Figure 3d, both approaches are capable of finding optimal solutions instantly, but the ones obtained by fixing method is far from optimal solutions to original problems. We notice that in Figure 3b and 3c, our framework struggles to find good solution in the early solving phases, but produces equivalent or better solutions within $1,000$ seconds comparing to the fixing method. A possible reason is that our solution results in larger feasible regions comparing to the fixing strategy. Conclusively, our framework always outperforms the fixing strategy.



(a) IP (b) WA (c) CA (d) IS

Figure 3: This figure shows average primal gaps achieved by the search method and the fixing method under the same partial solution. The primal gap is averaged across 100 instances, and each plot represents one dataset. The proposed framework shows a constantly dominant performance over the fixing-based method. Detailed parameters and settings can be found in the appendix.

6 Conclusions

We propose a predict and search framework for tackling difficult MILP problems that are routinely solved. A GNNs model was trained under a supervised learning setting to map from bipartite graph representations of MILP problems to weighted conditional marginal probabilities. The key contribution of this work is that we design a trust region based algorithm to search for high-quality feasible solutions with the guidance of such a mapping. Both theoretical and empirical supports are provided to illustrate the superiority of this framework over fixing-based strategies. With an extensive set of publicly available MILP datasets, we demonstrate the effectiveness of our proposed framework in quickly recognizing high-quality feasible solutions. Overall, our proposed framework achieved 51% and 9% better primal gaps comparing to SCIP and Gurobi, respectively.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Irwan Bello*, Hieu Pham*, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning, 2017. URL https://openreview.net/forum?id=rJY3vK9eg.

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.

Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, et al. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016.

Zhi-Long Chen. Integrated production and outbound distribution scheduling: review and extensions. *Operations research*, 58(1):130–148, 2010.

Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 1452–1459, 2020.

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.

Maxime Gasse, Simon Bowly, Quentin Cappart, Jonas Charfreitag, Laurent Charlin, Didier Chételat, Antonia Chmiela, Justin Dumouchelle, Ambros Gleixner, Aleksandr M Kazachkov, et al. The machine learning for combinatorial optimization competition (ml4co): Results and insights. In *NeurIPS 2021 Competitions and Demonstrations Track*, pp. 220–231. PMLR, 2022.

Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33: 18087–18097, 2020.

Humayun Kabir, Ying Wang, Ming Yu, and Qi-Jun Zhang. High-dimensional neural-network technique and applications to microwave filter modeling. *IEEE Transactions on Microwave Theory and Techniques*, 58(1):145–156, 2009.

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.

Elias B Khalil, Christopher Morris, and Andrea Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. *Update*, 2:x3, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Markus Kruber, Marco E Lübbecke, and Axel Parmentier. Learning when to use a decomposition. In *International conference on AI and OR techniques in constraint programming for combinatorial optimization problems*, pp. 202–210. Springer, 2017.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

Li Liu and Qi Fan. Resource allocation optimization based on mixed integer linear programming in the multi-cloudlet environment. *IEEE Access*, 6:24533–24542, 2018.

Paramet Luathep, Agachai Sumalee, William HK Lam, Zhi-Chun Li, and Hong K Lo. Global optimization method for mixed transportation network design problem: a mixed-integer linear programming approach. *Transportation Research Part B: Methodological*, 45(5):808–827, 2011.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks, 2020.

Yves Pochet and Laurence A Wolsey. *Production planning by mixed integer programming*, volume 149. Springer, 2006.

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. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020. URL https://openreview.net/forum?id=IVc9hqgibyB.

Anita Schöbel. A model for the delay management problem based on mixed-integer-programming. *Electronic notes in theoretical computer science*, 50(1):1–10, 2001.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

Jean-Paul Watson and David L Woodruff. Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8(4):355–370, 2011.

Taehyun Yoon. Confidence threshold neural diving, 2022.

Ya-xiang Yuan. Recent advances in trust region algorithms. *Mathematical Programming*, 151(1): 249–281, 2015.