A Review for GNN-Guided Mixed Integer Linear Programming

Bowen Xu
xubw1@shanghaitech.edu.cn

Abstract

The MILP (Mixed-Integer Linear Programming) problem is a widely emplyed problem in practical applications, but it is often difficult to solve efficiently using ordinary solvers because it is a NP-hard problem. Therefore, we model the MILP problem as a graph structure problem, and try to use GNN to predict the solution to improve the efficiency of the solution. In order to solve the problem that the prediction solution of GNN may be infeasible, we also use the trust region method to search for the optimal feasible solution. This predict-and-search method can not only speed up the solution rate compared with the commonly used solvers (gurobi, SCIP et al.), but also ensure that the solution is feasible in the simulation experiments. Based on the GNN-Guided MILP problem, we will also study some other algorithms related to this topic, analyze their advantages and disadvantages, and propose possible innovative improvements.

1 Introduction

MILP is mainly used to model combinatorial optimization problems. It has many application scenarios in reality, including resource allocation(Watson and Woodruff [2010]), production planning(Pochet and Wolsey [2006]), etc. Now there are some commonly used solvers can be used to solve the MILP problem, such as Gurobi, SCIP, etc.. These commonly used solvers apply algorithms including branch and cut, benders algorithm to solve the MILP problem. However, with the recent rise of ML algorithms and deep neural networks, many improvements to these solvers based on ML algorithms or Neural Networks have also emerged. Based on the Neural Network, The paper (Han et al. [2023]) consider modeling the MILP problem as a graph structure problem, and use the Graph Neural Network to assist the solver to solve the MILP problem. The specific method is that we use GNN to predict the optimal value of some variables to be solved. In this way, the variables to be solved in the problem are reduced to simplify the problem. Finally, the simplified problem is given to the solver. This method can reduce the computational consumption required by the solver, thereby reducing the time required to find the optimal solution.

Related Work. In the previous methods to optimize MILP problems, the paper of (Bengio et al. [2018]) conclude that there are mainly several ways as below: (i) end-to-end learning (Khalil et al. [2022]); (ii) learning to configuring algorithms Kruber et al. [2017] (iii) learning alongside optimization (Khalil et al. [2016]).

However, the previous ML algorithm, especially for end-to-end algorithms, to solve MILP problems also has the following challenges:

1. **High sample collection cost:** In order to obtain a network that can accurately predict the solution of the MILP problem, a large number of optimal problem solutions are often required for training. In the process of obtaining these training data, the common method is to use a solver to obtain them, but this method will bring a large computational cost.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

2. **Feasibility:** For the solution predicted by GNN, the possible problem is that this result does not satisfy all the constraints of the problem.

Based on the above challenges, some papers give some corresponding solutions. Thereby improving the performance of GNN-Guided MILP solver.

In the following content, we will firstly introduce the basic principles and algorithm design of GNN-Guided MILP solver based on the paper Han et al. [2023]. The paper uses Graph Neural Network (GNN) to guide the MILP solver to predict the solution and applies trust region algorithm to ensure the feasibility of the prediction solutions.

2 Preliminaries

2.1 Mixed Integer Linear Programming

The basic form of MILP (Mixed Integer Linear Programming) problem is in Equation 1 as follows:

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b, \\ & l \leq x \leq u, \\ & x \in \mathbb{Z}^q \times \mathbb{R}^{n-q}, \end{aligned} \tag{1}$$

in this equation, the variable $x = (x_1, \cdots, x_n)$ denotes the vector of variable which we want to solve and the vector x contains both real numbers and integers. The vector $c \in \mathbb{R}^n$ denotes the objective coefficient vector, and the constraint matrix $A \in \mathbb{R}^{m \times n}$, the constraint bounds $b \in \mathbb{R}^m$, lower bound $l \in (\mathbb{R} \cup \{-\infty\})^n$, upper bound $u \in (\mathcal{R} \cup \{\infty\})^n$ are all give data. The objective of MILP problem is to minimize the linear objective function $c^T x$ under the constraint conditions. The specific problem obtained by fixing the parameters is called an instance of the problem.

2.2 Graph Neural Network

Graph Neural Network (Graph Neural Network) is a class of neural network that can process the graph data. We denote the input graph as $G(V, E)$, where V is the set of vertices and E is the edges set. Denote $|V|$ as the number of vertices and $|E|$ as the number of edges. Here we use a more commonly used graph classification problem as an application example of GNN. The graph classification problem can be stated as: given a specific graph, classify it into its corresponding class. For example, we have two classes of molecular: high energy and low energy.

To achieve the goal of classification, we need to extract the feature of a graph, which facilitates us to use a special kinds of neural network: GNN. Therefore we introduce the defition of GNN.

We have the network architecture of GNN in Equation 2.2 as follows,

$$\begin{aligned} f^{(1)}\left(\bar{\boldsymbol{h}}_{G,j}\right) &= \mathbf{W}^{(1)} \bar{\boldsymbol{h}}_{G,j} \\ f^{(l)}\left(\bar{\boldsymbol{h}}_{G,j}\right) &= \sqrt{\frac{2}{m}} \mathbf{W}^{(l)} \sigma_{relu}\left(f^{(l-1)}\left(\bar{\boldsymbol{h}}_{G,j}\right)\right) \in \mathbb{R}^m, 1 < l \leq L \\ f_{GNN}\left(G; \boldsymbol{\theta}\right) &= \frac{1}{N} \sum_{j=1}^{N} \sqrt{2} \mathbf{W}^{(L+1)} \sigma_{relu}\left(f^{(L)}\left(\bar{\boldsymbol{h}}_{G,j}\right)\right) \end{aligned} \tag{2}$$

where $\bar{\boldsymbol{h}}_{G,j} = \sum_{i \in \mathcal{N}(j)} \frac{\boldsymbol{h}_{G,j}}{||\boldsymbol{h}_{G,i}||_2}$ is the aggregated feature of node j in the graph G, where $\boldsymbol{h}_{G,j}$ is the node features of j in Graph G. This is designed to enable the GNN to learn knowledge from neighbours of nodes. Due to the permutation invariance of GNN, we can solve the graph structured problem by GNN more efficiently than general Neural Network. This is actually an instance of the graph convolutional network(GCN) (Kipf and Welling [2016]).

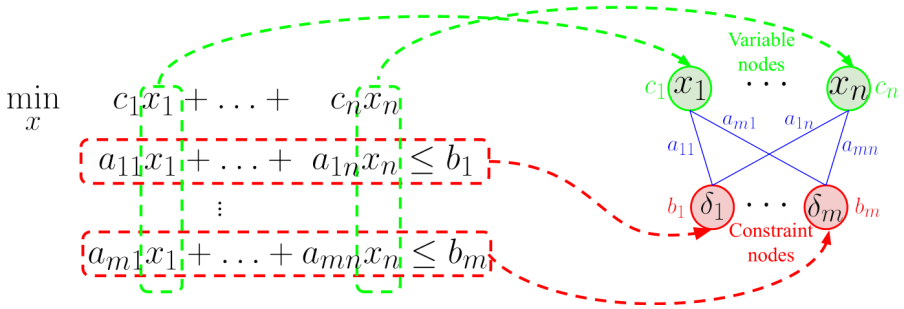


Figure 1: Biparitite graph representation of a MILP problem

3 Problem Formulation

3.1 Graph Modeling

On how to model an MILP problem as a graph structure, we can model it as a bipartite graph in Figure 1:

In the bipartite graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{A})$, where $\mathcal{V}$ is the node set, $\mathcal{E}$ is the edge set, $\mathcal{A}$ denotes the adjacency matrix. The node set $\mathcal{V}$ of the graph can be separated as two subsets: the n nodes above corresponding to the n variables being optimized and the other set of m nodes below corresponding to the m constraint in the equation (1). The edge in the graph denotes the connection between the variables and the constraint, which means if a variable is in that constraint, then there exists an edge between the corresponding variable node and the constraint node. The objective coefficients $\{c_1, \cdots, c_n\}$ and the constraint bound $\{b_1, \cdots, b_m\}$ and the non-zero coefficients of the constraint matrix A are respectively encoded as features into the corresponding variable nodes, constraint nodes and the edges. And the adjacency matrix is shown in Equation 3 as below:

$$\mathcal{A} \equiv \begin{bmatrix} 0 & C^T \\ C & 0 \end{bmatrix} \quad \textbf{where } C_{i,j} = \mathbb{I}_{A_{i,j} \neq 0} \tag{3}$$

4 Algorithm Analysis

4.1 GNN Prediction Model

After we model the original problem into a graph-structured problem and input the instance into GNN, the GNN Prediction Model will output the probability distribution of the values of each variable. Through the probability distribution of the optimal value of the variable output by GNN, we can use it to assist the solver in solving the problem.

4.1.1 Distribution Probability of Variables

As mentioned above, the purpose of the GNN Prediction model is to predict the conditional probability distribution of the variable x in the entire solution space to obtain the maximum objective function value, given an instance M of a MILP problem, so we can calculate the probability $p(x; M)$ through the value objective function $c^T x$ as follows:

$$p(x; M) \equiv \frac{\exp(-E(x; M))}{\sum_{x'} \exp(-E(x'; M))}, \quad \text{where } E(x; M) \equiv \begin{cases} c^T x & \text{if x is feasible,} \\ +\infty & \text{otherwise.} \end{cases} \tag{4}$$

This implies that an infeasible solution has a probability of 0. And the optimal solution has the largest probability, which satisfy the purpose of the probability setting.

4.1.2 Loss Function of GNN Prediction Model

To train the Graph Neural Network model, we should firstly collect some instances of a MILP problem: $\{(M^i, L^i)\}_{i=1}^N$, where $L \equiv \{x^{i,j}\}_{j=1}^{N_i}$ denotes the set of N_i feasible solutions of instance

M^i. And based on the probability form given above in Equation 4, we use the KL divergence to calculate the loss function of the GNN Prediction Model in Equation 5:

$$L(\theta) \equiv -\sum_{i=1}^{N}\sum_{j=1}^{N} w^{i,j} \log P_\theta(x^{i,j}; M^i), \quad \text{where } w^{i,j} \equiv \frac{\exp(-(c^i)^T x^{i,j})}{\sum_{k=1}^{N_i} \exp(-(c^i)^T x^{i,k})}, \tag{5}$$

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

4.1.3 Marginal Probability Learning

But according to the settings in Equation 5, we will face another challenge during the sampling: since we are learning the multi-dimensional variable x as a whole, if the dimension of x is high, then the process of sampling the high-dimensional variable x of instances will have a very large computational complexity. To solve the challenge, A common technique is to decompose the high-dimensional distribution into lower-dimensional distribution ones. Given the instance M, we denote x_d as the dth element of a solution vector x. In Nair et al. [2020b], it gives the assumption that the elements of variable x can be independent of each other, which means the probability of a solution can be shown as the form of Equation 6:

$$P_\theta(x; M) = \prod_{d=1}^{n} p_\theta(x_d; M) \tag{6}$$

With this assumption, the high-dimensional sampling problem can be decomposed in to n 1-dimensional sampling problem for each x_d according to their marginal probabilities $p_\theta(x_d; M)$.

Without loss of generality, we can set the integer elements of the variable of the original problem as binary, that is, x_i can only take 0 or 1, for $i \in \{1, \cdots, n\}$ and $x_i \in \mathbb{Z}$. In this way, since $p_\theta(x_d = 1; M) = 1 - p_\theta(x_d = 0; M)$, we only need the marginal probability of one dimension: $p_\theta(x_d = 1; M)$, for $d \in \{1, \cdots, n\}$. Similarly to the process of sampling, the process of prediction can only use the setting above, so the GNN Prediction Model of instance M can be represented as $F_\theta(M) \equiv (\hat{p}_1, \cdots, \hat{p}_n)$, where $\hat{p}_d = p_\theta(x_d = 1; M)$ for $d \in \{1, \cdots, n\}$.

According to the assumption above and Equation 6, we can have the loss function based on the marginal probability in Equation 7:

$$\begin{aligned} L(\theta) &= -\sum_{i=1}^{N}\sum_{d=1}^{n}\sum_{j=1}^{N_i} w^{i,j} \log p_\theta\left(x_d^{i,j}; M^i\right) \\ &= -\sum_{i=1}^{N}\sum_{d=1}^{n}\left\{\sum_{j\in S_d^i} w^{i,j} \log p_\theta\left(x_d^{i,j}; M^i\right) + \sum_{j\notin S_d^i} w^{i,j} \log p_\theta\left(x_d^{i,j}; M^i\right)\right\} \\ &= -\sum_{i=1}^{N}\sum_{d=1}^{n}\left\{p_d^i \log\left(\hat{p}_d^i\right) + \left(1 - p_d^i\right) \log\left(1 - \hat{p}_d^i\right)\right\}. \end{aligned} \tag{7}$$

This loss function indicates that the multi-dimensional distribution learning loss $L(\theta)$ becomes a summation of each dimension's marginal probability learnning loss. The converting of marginal probability can largely decrease the computational cost.

4.1.4 Multi-Layer Perceptron

In practical applications, we will split the GNN Prediction Model into two parts: Graph Neural Network (GNN) module and Multi-Layer Perceptron (MLP) module, The main function of the GNN module is to extract the main embedded node features converted from the graph structure information of MILP problem instances. While the main function of the MLP module is to classify the features output by the GNN module, and output the marginal probability of each dimension of the variable x: $F_\theta(M) \equiv (\hat{p}_1, \cdots, \hat{p}_n)$, where $\hat{p}_d = p_\theta(x_d = 1; M)$, for $d \in \{1, \cdots, n\}$.

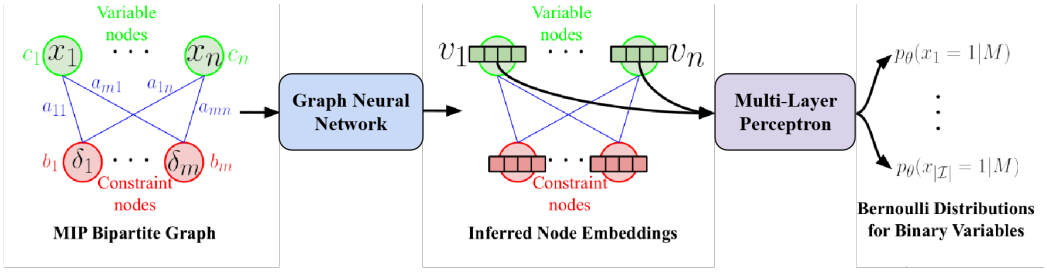


Figure 2: Pipeline of Graph Neural Network Prediction

And the network architecture of MLP is shown as below in Equation 8:

$$\begin{aligned} Z^{(0)} &= U, \\ Z^{l+1} &= \mathcal{A} f_{\theta(l)}(Z^l), \end{aligned} \tag{8}$$

where $f_{\theta(l)} : \mathbb{R}^D \to \mathbb{R}^H$ is the lth layer of Multi-Layer Perceptron, $\mathcal{A}$ denotes the adjacency matrix of graph G, matrix $U \in \mathbb{R}^{N \times D}$ denotes the output matrix containing feature vectors of GNN module.

4.1.5 Pipeline of GNN Prediction Model

To sum up, in practice, the entire GNN Prediction Model can be divided into two main steps: Graph Neural Network (GNN) module and Multi-Layer Perceptron (MLP) module. Combining with the two modules above, as shown in the Figure 2, the GNN Prediction Model can realize the function of predicting the probability distribution of variable in one instance.

4.2 Trust Region Search

Through the above analysis, we can predict the probability distribution of the variable x through the GNN Prediction model. But we still face a challenge: the value of the variable x predicted by the GNN Prediction model cannot satisfy the constraints of the original problem in many cases. Therefore, we need to make further adjustments on the basis of this probability distribution, so that the solution of the problem can satisfy the constraints. To address this challenge, we introduce the trust region method to solve it.

4.2.1 Fixed Variables Strategy

First, based on the predicted probability distribution of the variables output by the GNN Prediction Model, we select the values of variable x's elements $(x_1, \cdots x_n)$ with higher confidence to fix. The method is as follows:

1. Firstly, **sort** the predicted probability vectors $(\hat{p}_1, \cdots, \hat{p}_n)$ output by the GNN Prediction Model from large to small.
2. Secondly, **select** the k_1 elements with the highest probability, and assign them a value of 1; select the k_0 smallest elements, and assign them to 0. k_1 and k_0 are hyperparameters denotes the number of partial fixed solutions.
3. Thirdly, **output** the predicted prediction vector with partially fixed values $\hat{x}$.

In this way, we get a prediction variable $\hat{x}$ with some elements assigned. The pipeline of Fixed Variables Strategy is shown in Figure 3.

4.2.2 Search Within Trust Region

After that, we will conduct a further search based on the variable $\hat{x}$ output by the Fixed Variables Strategy to avoid the infeasibility of the solution. Let x^* denotes the optimal feasible solution of

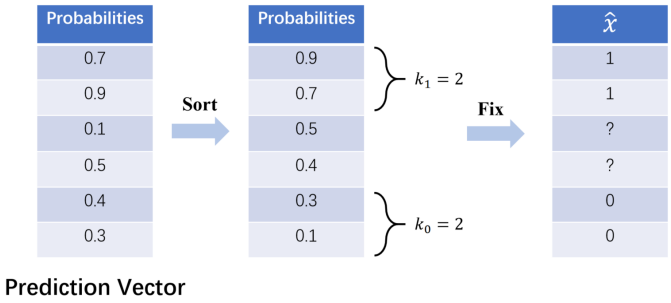


Figure 3: Fixed Variables Strategy

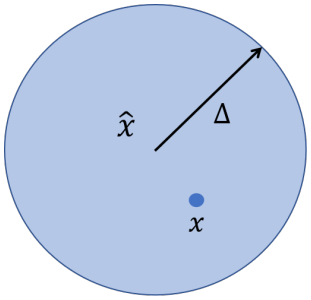


Figure 4: Trust Region of $\hat{x}$

instance M, according to Han et al. [2023], we know that $\hat{x}$ is close to x^*. Based on the assumption, it is reasonable to search the optimal feasible solution x^* in the trust region of $\hat{x}$.

Specifically, we use l_1 norm to characterize the distance between $\hat{x}$ and x^* and set a small hyperparameter Δ as the scope of $\hat{x}$'s trust region. Therefore, the trust region of $\hat{x}$ is $\{x \in \mathbb{R}^n | \ ||x - \hat{x}||_1 \leq \Delta\}$ as shown in Figure 4, which we denote as $B(\hat{x}, \Delta)$.

We create the scope of trust region as a new constraint δ'. After that, we set a new instance $M^{'}$, whose variable is partial fixed solution $\hat{x}$, and its constraint is the constraint δ' which we set through the trust region. Through this fixed partial decomposition method, we greatly reduce the variable dimension and computational complexity of the new instance $M^{'}$. After this, we can input such a new problem $M^{'}$ into the solver to solve it. The complete pseudocode of predict-and-search algorithm is shown in the Figure 5.

What's more, according to Theorem 9, we can know that after the Search within trust region step, the obtained objective function value must be better than the result obtained only after the Fixed Variables Strategy operation.

Theorem 1 *Note that* $B(\hat{x}, \Delta) \subset B(\hat{x}, 0)$

$$\min_{x \in D \cap B(\hat{x}, \Delta)} c^T x \leq \min_{x \in D \cap B(\hat{x}, 0)} c^T x, \tag{9}$$

where D is the solution space of the original instance M.

Compared with directly inputting the original instance M into the solver, this predict-and-search method shown in Figure 6 can reduce a lot of computational consumption.

5 Overview of Existing Work

In this section, we will mainly summarize some other important work related to GNN-Guided MILP Solver.

Algorithm 1 Predict-and-search Algorithm

Parameter: Size $\{k_0, k_1\}$, radius of the neighborhood: 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' denote the instance M with new constraints and variables
Let $x = SOLVE(M')$
return x

Figure 5: Predict-and-Search Algorithm

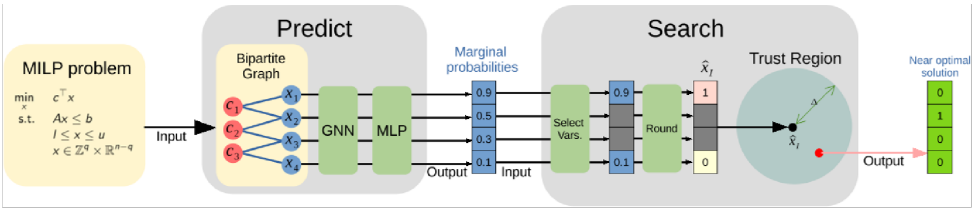


Figure 6: Predict-and-Search pipeline

5.1 A GNN-Guided Predict-and-Search Framework for Mixed-Integer Linear Programming

The detailed review of the paper Han et al. [2023] is in the content above. This paper uses Graph Neural Network (GNN) to guide the MILP solver to predict the solution and applies trust region algorithm to ensure the feasibility of the prediction solutions.

5.2 Solving Mixed Integer Programs Using Neural Networks

The paper Nair et al. [2020a] constructs two corresponding neural network-based components, Neural Diving and Neural Branching, to use in a base MIP solver such as SCIP.

5.2.1 Introdution

Mixed integer programming (MIP) solvers generally use a series of heuristic algorithms to solve. We find that machine learning algorithms can construct better heuristic algorithms by exploiting the shared structure among data instances.

The paper applies the learning algorithm to the two subtasks of Diving and Branching, namely Neural Diving and Neural Branching. Among them, Neural Diving uses the neural network to generate partial solutions for integer variables, and uses the SCIP solver to solve the remaining variables (personal understanding is to prevent the problem of too many decision variables); Neural Branching is used to bound the gap between the objective value of the best assignment and an optimal one.

5.2.2 Method Details

The method includes two main parts: Neural Diving and Neural Branching. Their main implementation methods are as follows:

1. Neural Diving: train a deep neural network to produce multiple partial assignments of the integer variables of the input MIP. The remaining unassigned variables define smaller 'sub-MIPs', which are solved using an off-the-shelf MIP solver (e.g., SCIP) to complete the assignments.
2. Neural Branching: train a deep neural network policy to imitate choices made by an expert policy. The imitation target is a well-known heuristic called Full Strong Branching (FSB), which has been empirically shown to produce small search trees.

5.3 Exact Combinatorial Optimization with Graph Convolutional Neural Networks

This paper Gasse et al. [2019] mainly introduce a new method of using GNN to assist branch-and-bound algorithm to solve combinatorial optimization problems.

5.3.1 Introduction

Combinatorial optimization problems are usually solved using the branch-and-bound paradigm. The authors of the paper propose a new Graph Convolutional Neural (GCN) Network model for learning branch-and-bound variable selection policies that exploits natural variable-constrained bipartite graph representations of mixed-integer linear programs (MILPs). The authors train the model via strong branch expert rules and demonstrate on a range of NP-hard problems that the resulting policies improve state-of-the-art machine learning methods for branching and generalize to larger on many instances.

5.3.2 Motivation

In reality, most combinatorial optimization problems can be formulated as mixed integer linear programs, and many of these problems are solved using Branch-and-Bound algorithm.

Branch-and-Bound recursively partitions the solution space into search trees and computes slack bounds to prune subtrees that cannot prove to contain optimal solutions. This iterative process requires successive decisions, such as node selection (choosing the next node to evaluate), variable selection (choosing variables to partition the search space of nodes). The Branch-and-Bound decision process traditionally follows a series of hard-coded heuristics, carefully designed by experts to minimize the average solution time for a representative set of MILP instances. However, in many cases it is common to repeatedly solve similar combinatorial optimization problems, which may differ significantly from the set of instances on which Branch-and-Bound algorithms are usually evaluated.

Therefore, it is necessary to use statistical learning to automatically tune Branch-and-Bound algorithms to solve the desired class of problems. However, this also faces two challenges:

1. How to encode the state of the MILP Branch-and-Bound decision-making process;
2. How to get rules that generalize, at least to similar instances, and ideally to larger instances than those seen in training.

The author proposes to use Graph Convolutional Neural (GCN) network to solve the above challenges. The author's contributions include:

1. Encoding branching strategies into graph convolutional neural networks, which allows exploiting the natural bipartite graph representation of MILP problems, thereby reducing manual feature engineering;
2. Approximate strong branch decisions by using behavioral cloning with cross-entropy loss, an easier task than predicting strong branch scores or rankings.

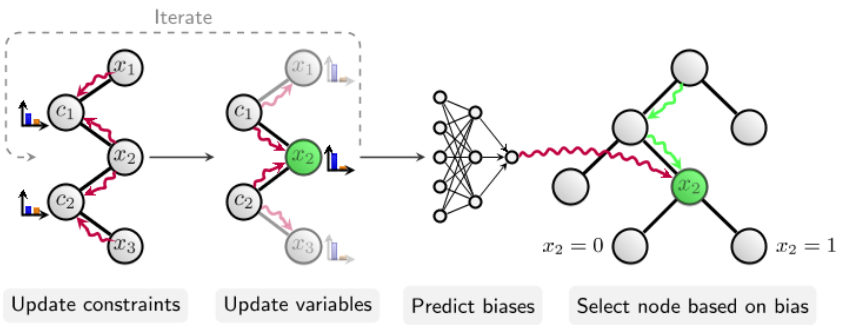


Figure 7: MIP-GNN

5.3.3 Algorithm Details

Since the Branch-and-Bound variable selection problem can be expressed as a Markov decision process, the training strategy naturally comes to mind with reinforcement learning. In this work, the authors learn directly from expert branching rules, an approach often referred to as imitation learning.

Imitation Learning The authors train by behavioral cloning using strong branching rules, an approach that is computationally expensive but typically produces the smallest Branch-and-Bound trees. The authors first run the expert algorithm on the set of training instances of interest, record the expert state-action pairs: $\mathcal{D} = \{(s_i, a_i^*)\}_{i=1}^N$, and then learn the policy by minimizing the cross-entropy loss in Equation 10:

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{(s,a^*)\in\mathcal{D}} \log \pi_\theta(a^*|s) \tag{10}$$

5.4 MIP-GNN: A Data-Driven Framework for Guiding Combinatorial Solvers

This paper Khalil et al. [2022] proposes MIP-GNN solver, a general framework for enhancing MIP (Mixed Integer Programming) solvers with data-driven insights.

5.4.1 Introduction

This paper introduces MIP-GNN, which is a generic GNN-based architecture to guide heuristic components within common MIP solvers (e.g. CPLEX). By leveraging the structural information within the MILP's constraint-variable interaction, the authors trained MIP-GNN in a supervised way to predict variable biases, i.e., the likelihood of a variable taking a value of 1 (in the problem of Binary Linear Programming) in near-optimal solutions.

5.4.2 Method Details

Given an encoded Binary Linear Programming (BLP), the MIP-GNN aims to learn an embedding, i.e., a vectorial representation of each variable, which is subsequently fed into a multi-layer perceptron (MLP) for predicting the corresponding bias between the prediction and the optimal solution. To learn meaningful variable embeddings that are relevant to bias prediction, the MIP-GNN consists of two passes, the variable-to-constraint and the constraint-to-variable pass. The pipeline of MIP-GNN is shown in Figure 7.

5.5 Confidence Threshold Neural Diving

This paper Yoon presents a post-hoc method based on Neural Diving to build heuristics more flexibly. In this paper, authors hypothesize that variables with higher confidence scores are more definite to be included in the optimal solution. For this hypothesis, authors provide empirical evidence that confidence threshold technique produces partial solutions leading to final solutions with better primal objective values. The illustration of Confidence Threshold Neural Diving is in Figure 8.

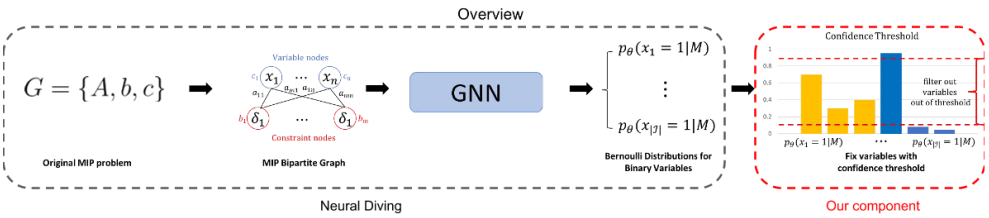


Figure 8: The Overview of Confidence Threshold Neural Diving

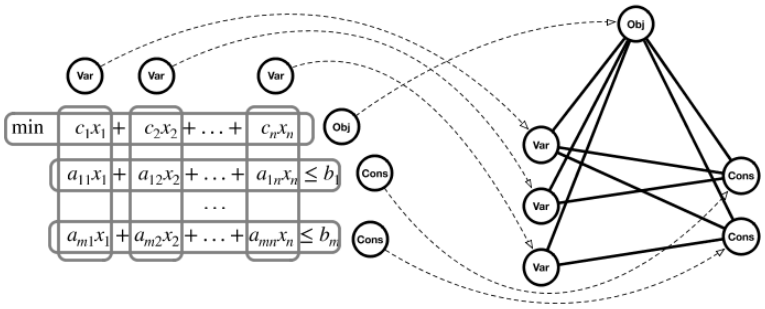


Figure 9: Transforming an MIP instance to a tripartiete graph

5.6 Hybrid Models for Learning to Branch

In the practical application, GNN relies on a GPU for inference, while MILP solvers are purely CPU-based. In fact, GPU severely limits the GNN-Guided MILP Solver application as many practitioners may not have access to high-end GPUs. This paper Gupta et al. [2020] works to devise an alternate computationally inexpensive model that retains the predictive power of the GNN architecture based on CPU machines. And it proposed an architecture that combines the expressive power of GNNs with computationally inexpensive multi-layer perceptrons (MLP) for branching.

5.7 Neural Combinatorial Optimization with Reinforcement Learning

This paper Bello et al. [2016] presents a framework to tackle combinatorial optimization problems using neural networks and reinforcement learning. And focus on the traveling salesman problem (TSP) and present a set of results for each variation of the framework. The RL algorithm this paper used is Actor-Critic algorithm and use this algorithm to solve combinatorial optimization problems.

5.8 Accelerating Primal Solution Findings for Mixed Integer Programs Based on Solution Prediction

In many applications, a similar Mixed Integer Programming (MIP) model is solved on a regular basis, maintaining remarkable similarities in model structures and solution appearances but differing in formulation coefficients. This offers the opportunity for machine learning methods to explore the correlations between model structures and the resulting solution values. To address this issue, this paper Ding et al. [2019] proposes to represent an MIP instance using a tripartite graph shown in Figure 9, based on which a Graph Convolutional Network (GCN) is constructed to predict solution values for binary variables.

6 Criticism of the Existing Work

Through the research on the existing GNN-Guided MILP Solver, it can be found that there are still some aspects that can be improved. Below we will summarize some problems of these existing works that can be critically considered and improved:

6.1 Instance Structure Limitation

In the method proposed in the paper, there are restrictions on the structure of the instance: since the network is constructed in the MLP module of the GNN Prediction Model as follows:

$$\begin{aligned} Z^{(0)} &= U, \\ Z^{l+1} &= \mathcal{A} f_{\theta(l)}(Z^l), \end{aligned} \tag{11}$$

where $\mathcal{A}$ denotes the adjacency matrix of graph G,

We can find that the structure of the MLP network is limited by the adjacency matrix $\mathcal{A}$, so if the structural information of the graph (such as the size or value of the adjacency matrix $\mathcal{A}$ changes), then our MLP network cannot process this information or obtain prediction results There will be a big gap. But in practical application, we found that for a similar problem, if there is not much difference between different instances, then the solutions of these two instances will not be very different, and the solution process will be similar.

Therefore, we consider that we can use the transfer learning method to improve the MLP module, so that when the instances are different but have similar structures, the GNN prediction model can also be used to make predictions. In this way, we can greatly improve the versatility of GNN-Guided MILP Solver.

6.2 Extensions for Different Objective Functions

The current research in related fields mainly focuses on the research of MILP problems. The objective function of the MILP problem is just a simple linear function $c^T x$. Therefore, we consider whether such a model can be extended to more forms of objective functions, such as using GNN-Guided Solver to solve Quadratic Programming (QP) problems.

6.3 Branching and Bounding Applied in Trust Region Search

Through analysis, we can find that in the part of Trust Region Search, the method adopted in the original paper Han et al. [2023] is to traverse all points in the trust region of $\hat{x}$ to find the optimal solution. But this traversal method will undoubtedly bring about very huge calculation consumption. We observed that during the search process, since the variable x is a binary vector (that is, the elements of the variable x can only be 0 or 1). For binary vector search problems, the branch-and-bound algorithm is a very common method for faster and more efficient searches. Therefore, we consider using the branch-and-bound algorithm to assist trust region search.

6.4 Binary Solution Variable

We found that in the original paper Han et al. [2023], it was assumed that the integer elements in the solution variable x are binary, which is a relatively strong assumption. Therefore, we consider whether it can be extended to the integer field.

7 Numerical Results

In the section of the simulation experiment, we first train the GNN Prediction Model by 1000 instances of the problem, these instances including 50 variables and $180 + n$ constraints, where n satisfy a Gaussian distribution and the variance of the distribution is small (The number of constraints is about 175-180). This is not a small scale problem.

And we run the GNN-Guided Predict-and-Search Algorithm to predict 100 random independent instances of the problem, we obtain the solutions in Figure 10:

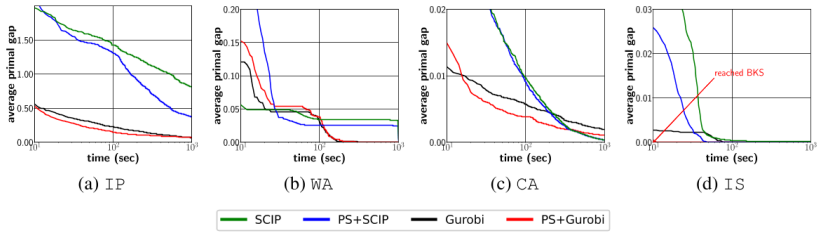
From the solution output by the algorithm, we can find that the algorithm successfully obtain almost all the optimal solution of the 100 instances in an extremely short time.

Also, if we compare the performance of GNN-Guided MILP Solver with the common MILP Solver, we can have the results in Figure 11.

This result proves the effectiveness of GNN-Guided Predict-and-Search Algorithm.

```
Set parameter LogFile to value "./logs/IS/IS_GRB_Predect&Search/IS-0.lp.log"
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (linux64)
Thread count: 40 physical cores, 80 logical processors, using up to 1 threads
Optimize a model with 282 rows, 100 columns and 608 nonzeros
Model fingerprint: 0xfb3bd22a
Variable types: 50 continuous, 50 integer (50 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+01]
Found heuristic solution: objective -0.0000000
Presolve removed 183 rows and 52 columns
Presolve time: 0.00s
Presolved: 99 rows, 48 columns, 296 nonzeros
Variable types: 0 continuous, 48 integer (48 binary)
Found heuristic solution: objective 14.0000000

Root relaxation: objective 1.500000e+01, 38 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

H    0     0                    15.0000000   48.00000   220%     -    0s
     0     0          -    0        15.00000   15.00000  0.00%     -    0s

Explored 1 nodes (60 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 1 (of 80 available processors)

Solution count 3: 15 14 -0

Optimal solution found (tolerance 1.00e-04)
Best objective 1.500000000000e+01, best bound 1.500000000000e+01, gap 0.0000%
```

Figure 10: Simulation Results of GNN-Guided Predict-and-Search Algorithm



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

Figure 2: Performance comparisons between PS, Gurobi and SCIP, where the y-axis is relative

Figure 11: Comparison of Performance

8 New Contributions

8.1 Solution to Instance Structure Limitation

Based on the problem of instance structure limitation described in 5.1, we found that if the graph structures of instances are different, then the GNN Prediction Model cannot process the input or predict the accurate probability. From 5.1, we know that it results from the existence of adjacency matrix $\mathcal{A}$ in the architecture of MLP module that limit the graph structure of input. To solve this problem, we try to modify the network structure using some techniques of matrix slacking which we learned in the class. Specifically, we plan to solve this problem by introducing GraphSAGE technique into GNN.

Firstly we will introduce the GraphSAGE: it is an inductive learning framework that can efficiently generate unknown node embedding by using the feature information of nodes. The core idea of GraphSAGE is to generate the embedding vector of the target nodes by learning a function that aggregates the neighbor nodes.

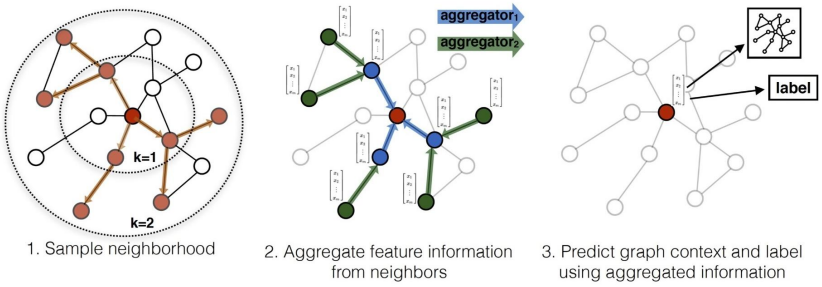


Figure 12: Visual Illustration of the GraphSAGE

The operation process of GraphSAGE is shown in the figure above 12, which can be divided into three steps:

1. Sampling the neighbor nodes of each node in the graph.
2. According to the aggregation function to aggregate the information contained in the neighbor nodes.
3. The vector representation of each node in the graph is obtained for downstream tasks.

With GraphSAGE technology, we can make changes in the input of the GNN network. The naive node features generated by aggregated function f_1 used in the original paper are replaced by the embedded node features generated by GraphSAGE. Through this technology, we can embed the complete graph structure information (include the adjacency information) into the node features of GNN, so we no longer need to add the adjacency matrix $\mathcal{A}$ in MLP.

So the new network architecture of MLP module is shown in Equation 12 as below:

$$\begin{aligned} Z^{(0)} &= U, \\ Z^{l+1} &= f_{\theta(l)}(Z^l), \end{aligned} \tag{12}$$

Through this GraphSAGE technique, we also avoid the occurrence of adjacency matrix $\mathcal{A}$ in MLP, and solve the problems caused by different graph structure information of different instances.

By the GraphSAGE technique, our GNN-Guided MILP Solver can solve the instance of similar problems and is no longer limited by the constraint structure information of problems. This has been verified in our simulation experiments that the accuracy and the universality of solving similar problems can be greatly improved.

9 Conclusion

This review paper mainly focuses on GNN-Guided MILP Solver, and introduces the method of solving MILP problem by Graph Neural Network (GNN). The innovation points and technical details in paper Han et al. [2023], It mainly includes two parts: GNN Prediction Model and Trust Region Search. The Predict-and-Search Framework is composed of the above two parts to assist Solver to predict the solution. In addition, the algorithms using other ML, RL and other algorithms, as well as different network architectures (NN / GCN, etc.) to assist the solution are summarized. Finally, aiming at the problems existing in the existing algorithms, we also propose some corresponding improvement schemes, such as introducing GraphSAGE to improve the universality of the algorithm. In conclusion, the use of GNN-Guided MILP Solver will greatly improve the performance of traditional Solver and make this field more widely used.

References

I. Bello, H. Pham, Q. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning, Nov 2016.

Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour dhorizon, Nov 2018.

J.-Y. Ding, C. Zhang, S. Lei, S. Li, B. Wang, Y. Xu, and L. Song. Accelerating primal solution findings for mixed integer programs based on solution prediction, Jun 2019.

M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks, 2019.

P. Gupta, M. Gasse, E. Khalil, P. Mudigonda, A. Lodi, and Y. Bengio. Hybrid models for learning to branch, Jan 2020.

Q. Han, L. Yang, Q. Chen, X. Zhou, D. Zhang, A. Wang, R. Sun, and X. Luo. A gnn-guided predict-and-search framework for mixed-integer linear programming, Feb 2023.

E. Khalil, P. Bodic, L. Song, G. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming, Feb 2016.

E. Khalil, C. Morris, and A. Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers, May 2022.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

M. Kruber, M. E. Lübbecke, and A. Parmentier. *Learning When to Use a Decomposition*, page 202210. May 2017. doi: 10.1007/978-3-319-59776-8_16. URL http://dx.doi.org/10.1007/978-3-319-59776-8_16.

V. Nair, S. Bartunov, F. Gimeno, I. Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, R. Addanki, T. Hapuarachchi, T. Keck, J. Keeling, P. Kohli, I. Ktena, Y. Li, O. Vinyals, and Y. Zwols. Solving mixed integer programs using neural networks, Dec 2020a.

V. Nair, S. Bartunov, F. Gimeno, I. Glehn, P. Lichocki, I. Lobov, B. ODonoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, R. Addanki, T. Hapuarachchi, T. Keck, J. Keeling, P. Kohli, I. Ktena, Y. Li, O. Vinyals, and Y. Zwols. Solving mixed integer programs using neural networks, Dec 2020b.

Y. Pochet and L. Wolsey. Production planning by mixed integer programming, Apr 2006.

J.-P. Watson and D. L. Woodruff. Progressive hedging innovations for a class of stochastic mixed-integer resource allocation problems. *Computational Management Science*, 8(4): 355370, Jul 2010. doi: 10.1007/s10287-010-0125-4. URL http://dx.doi.org/10.1007/s10287-010-0125-4.

T. Yoon. Confidence threshold neural diving.