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

Jian-Ya Ding,[1] Chao Zhang,[1] Lei Shen,[1] Shengyin Li,[1] Bing Wang,[1]* Yinghui Xu,[1] Le Song[2,3]
[1]Artificial Intelligence Department, Zhejiang Cainiao Supply Chain Management Co., Ltd, Hangzhou, China
[2]Ant Financial Services Group, Hangzhou, China
[3]Georgia Institute of Technology, GA, USA
{jianya.djy, chao.zc, kenny.sl, shengyin.lsy, lingfeng.wb}@cainiao.com, renji.xyh@alibaba-inc.com, le.song@antfin.com

Abstract

Mixed Integer Programming (MIP) is one of the most widely used modeling techniques for combinatorial optimization problems. In many applications, a similar 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, we propose to represent a MIP instance using a tripartite graph, based on which a Graph Convolutional Network (GCN) is constructed to predict solution values for binary variables. The predicted solutions are used to generate a local branching type cut which can be either treated as a global (invalid) inequality in the formulation resulting in a heuristic approach to solve the MIP, or as a root branching rule resulting in an exact approach. Computational evaluations on 8 distinct types of MIP problems show that the proposed framework improves the primal solution finding performance significantly on a state-of-the-art open-source MIP solver.

Introduction

Mixed Integer Programming (MIP) is widely used to solve *combinatorial optimization* (CO) problems in the field of *Operations Research* (OR). The existence of integer variables endows MIP formulations with the ability to capture the discrete nature of many real-world decisions. Applications of MIP include production scheduling (Chen 2010), vehicle routing (Laporte 2009), facility location (Farahani and Hekmatfar 2009), to mention only a few. In many real-world settings, homogeneous MIP instances with similar scales and combinatorial structures are optimized repeatedly but treated as completely new tasks. These instances share remarkable similarities in model structures and solution appearances, which motivates us to integrate *Machine Learning* (ML) methods to explore correlations between a MIP model's structure and its solution values to improve the solver's performance.

Identifying correlations between problem structures and solution values is not new, and is widely used as guidelines for heuristics design for CO problems. These heuristic methods are usually human-designed priority rules to guide the search directions to promising regions in solution space. For example, the nearest neighbor algorithm (Cook 2011) for the traveling salesman problem (TSP) constructs a solution by choosing the nearest unvisited node as the salesman's next move, based on the observation that two distantly distributed nodes are unlikely to appear consecutively in the optimal route. Similar examples include the shortest processing time first heuristic for flow shop scheduling (Pinedo 2012), the first fit algorithm for bin packing (Dósa and Sgall 2013), among many others. A major drawback of human-designed heuristics is the lack of generalization to other problems, where new domain knowledge has to be re-identified.

MIP models can describe CO problems of various types using a standard formulation strategy $z = \min_{\boldsymbol{A}\boldsymbol{x}\leq\boldsymbol{b},\boldsymbol{x}\in\mathcal{X}} \boldsymbol{c}^T\boldsymbol{x}$, differing only in model coefficients $\boldsymbol{A}, \boldsymbol{b}$ and $\boldsymbol{c}$ and integrality constraints $\mathcal{X}$. This makes it possible to explore connections between problem structures and the resulting solution values without prior domain knowledge. Notice that most MIP models are binary variables intensive[1], a natural way to explore hidden information in the model is to treat solution value prediction of binary variables as binary classification tasks. Major challenges in solution prediction lie in the implicit correlations among decision variables, since a feasible solution $\boldsymbol{x}$ is restricted by constraints in MIP, i.e., $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$. Rather than predicting each decision variable value isolatedly, we propose a tripartite graph representation of MIP, where each MIP instance is encoded as an undirected graph with three types of nodes: decision variables nodes, constraints nodes and the objective node. Correlations among variables are reflected in embeddings of the trigraph where each vertex maintains aggregate feature information from its neighbors.

Incorporating solution prediction results in MIP solving process is not trivial. Fixing a single false-predicted decision variable can sometimes lead to the infeasibility of the entire problem. Instead of utilizing the predicted solutions

*Corresponding author

[1]Take the benchmark set of MIPLIB 2017 (miplib2017 2018) as an example, among all 240 MIP benchmark instances, 164 of them are Binary Integer Linear Programming (BILP) problems, and 44 out of the 76 remainings are imbalanced in the sense that binary variables account for more than 90% of all integer variables.

directly, we identify "predictable" decision variables and use this information to guide the Branch and Bound (B&B) tree search to focus on "unpredictable" ones to accelerate convergence. This is achieved by a novel labeling mechanism on the training instances, where a sequence of feasible solutions is generated by an iterated proximity search method (Fischetti and Monaci 2014). *Stable* decision variables, of which the value remain unchanged across these solutions, are recorded. It is noticeable that although obtaining optimal solutions is sometimes a difficult task, the stable variables can be viewed as an easy-to-predict part that reflects the MIP's local optimality structure. This labeling mechanism is very helpful especially for difficult MIP instances when solving them to optimality is almost impossible.

The overall framework of solution prediction based MIP solving can be summarized as follows:

Training data generation: For a certain type of CO problem, generate a set of p MIP instances $\mathcal{I} = \{I_1, \ldots, I_p\}$ of similar scale from the same distribution $\mathbb{D}$. For each $I_i \in \mathcal{I}$, collect variable features, constraint features, and edge features, and use the iterated proximity search method to generate solution labels for each binary variable in I_i.

GCN model training: For each $I_i \in \mathcal{I}$, generate a tripartite graph from its MIP formulation. Train a Graph Convolutional Network (GCN) for binary variable solution prediction based on the collected features, labels and trigraphs.

Application of solution prediction: For a new MIP instance I from $\mathbb{D}$, collect features, build the trigraph and use the GCN model to make solution value predictions. The predicted solutions are used to generate a local branching type cut which is treated as a global (invalid) inequality in the formulation resulting in a heuristic approach, or as a root branching rule resulting in an exact approach to solve I.

Related work

With similar motivation, there are some recent attempts that consider integrating ML and OR to solve CO problems. Dai et al. combined reinforcement learning and graph embedding to learn greedy heuristics for several optimization problems over graphs. Li, Chen, and Koltun trained a graph convolutional network to estimate the likelihood that a vertex in a graph appears in the optimal solution. Selsam et al. proposed a message passing neural network to solve SAT problems via a supervised learning framework. Pointer Networks (or its variants) with recurrent neural network (RNN) decoder are designed to solve permutation related optimization problems such as TSP and Vehicle Routing Problem (VRP) (Vinyals, Fortunato, and Jaitly 2015), (Kool, van Hoof, and Welling 2018; Kool and Welling 2018) and (Nazari et al. 2018). Different from their settings, the proposed framework does not restrict to certain graph-based problems but can adapt to a variety of CO problems using a standard MIP formulation.

Quite related to our work, there is an increasing concern in using ML techniques to enhance MIP solving performance. Alvarez, Louveaux, and Wehenkel, Alvarez, Wehenkel, and Louveaux, Khalil et al. used learning-based approaches to imitate the behavior of the so-called strong branching method, a node-efficient but time-consuming branching variable selection method in the B&B search tree. In a very recent work by Gasse et al., a GCN model is trained to imitate the strong branching rule. Our model is different from theirs in terms of both the graph and network structure as well as the application scenario of the prediction results. Tang, Agrawal, and Faenza designed a deep reinforcement learning framework for intelligent selection of cutting planes. He, Daume III, and Eisner used imitation learning to train a node selection and a node pruning policy to speed up the tree search in the B&B process. Khalil et al. used binary classification to predict whether a primal heuristic will succeed at a given node and then decide whether to run a heuristic at that node. Kruber, Lübbecke, and Parmentier proposed a supervised learning method to decide whether a Danzig-Wolfe reformulation should be applied and which decomposition to choose among all possibles. Interested readers can refer to Bengio, Lodi, and Prouvost for a comprehensive survey on the use of machine learning methods in CO.

The proposed MIP solving framework is different from previous work in two aspects:

Generalization: Previous solution generation method for CO usually focus on problems with certain solution structures. For example, applications of Pointer Networks (Vinyals, Fortunato, and Jaitly 2015; Kool and Welling 2018) are only suited for sequence-based solution encoding, and reinforcement learning (Dai et al. 2017; Li, Chen, and Koltun 2018) type decision making is based on the assumption that a feasible solution can be obtained by sequential decisions. In contrast, the proposed framework does not limit to problems of certain types but applies to most CO problems that can be modeled as MIPs. This greatly enlarges the applicable area of the proposed framework.

Representation: Previous applications of ML techniques to enhance MIP solving performance mainly use hand-crafted features, and make predictions on each variable independently. Notice that the solution value of a variable is strongly correlated to the objective function and the constraints it participates in, we build a tripartite graph representation for MIP, based on which graph embedding technique is applied to extract correlations among variables, constraints and the objective function without human intervention.

The Solution Framework

Consider a MIP problem instance I of the general form:

$$\min \quad \boldsymbol{c}^T \boldsymbol{x} \tag{1}$$

$$\text{s.t.} \quad \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \tag{2}$$

$$x_j \in \{0,1\},\ \forall j \in \mathcal{B}, \tag{3}$$

$$x_j \in \mathbb{Z},\ \forall j \in \mathcal{Q}, \quad x_j \geq 0,\ \forall j \in \mathcal{P}, \tag{4}$$

where the index set of decision variables $\mathcal{U} := \{1, \ldots, n\}$ is partitioned into $(\mathcal{B}, \mathcal{Q}, \mathcal{P})$, and $\mathcal{B}, \mathcal{Q}, \mathcal{P}$ are the index set of binary, general integer and continuous variables, respectively. The main task here is to predict the probability that a binary variable $x_j,\ j \in \mathcal{B}$ takes value 1 (or zero) in the optimal solution. Next, we describe in detail the tripartite graph representation of MIP, the GCN model structure, and how solution prediction results are incorporated to accelerate MIP solving.

Graph Representation for MIP

Our main idea is to use a tripartite graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ to represent an input MIP instance I. In particular, objective function coefficients $\boldsymbol{c}$, constraint right-hand-side (RHS) coefficients $\boldsymbol{b}$ and coefficient matrix $\boldsymbol{A}$ information is extracted from I to build the graph. Vertices and edges in the graph are detailed as follows and graphically illustrated in Fig 1.

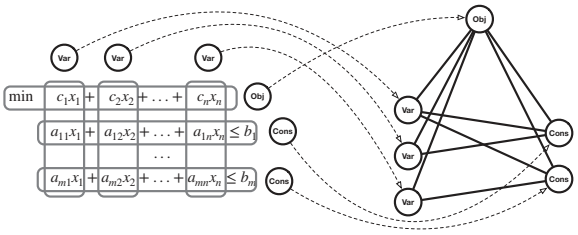

Figure 1: Transforming a MIP instance to a tripartiete graph

Vertices:

1) the set of decision variable vertices $\mathcal{V}_V$, each of which corresponds to a binary variable in I.

2) the set of constraint vertices $\mathcal{V}_C$, each of which corresponds to a constraint in I.

3) an objective function vertex o.

Edges:

1) v-c edge: there exists an edge between $v \in \mathcal{V}_V$ and $c \in \mathcal{V}_C$ if the corresponding variable of v has a non-zero coefficient in the corresponding constraint of c in the MIP formulation.

2) v-o edge: for each $v \in \mathcal{V}_V$, there exists an edge between v and o.

3) c-o edge: for each $c \in \mathcal{V}_C$, there exists an edge between c and o.

Remark. The presented trigraph representation not only captures connections among the variables, constraints and objective functions but maintains the detailed coefficients numerics in its structure as well. In particular, non-zero entries in coefficient matrix $\boldsymbol{A}$ are included as features of v-c edges, entries in objective coefficients $\boldsymbol{c}$ as features of v-o edges, and entries in $\boldsymbol{b}$ as features of c-o edges. Note that the constraint RHS coefficients $\boldsymbol{b}$ are correlated to the objective function by viewing LP relaxation of I from a dual perspective.

Solution Prediction for MIP

We describe in Algorithm 1 the overall forward propagation prediction procedure based on the trigraph. The procedure consists of three stages: 1) a fully-connected "EMBEDDING" layer with 64 dimension output for each node so that the node representations are of the same dimension (lines 1-3 in the algorithm). 2) a graph attention network to transform node information among connected nodes (lines 4-12). 3) two fully-connected layers between variable nodes and the output layer (line 13). The sigmoid activation function is used for output so that the output value can be regarded as the probability that the corresponding binary variable takes value 1 in the MIP solution. The overall GCN is trained by minimizing the binary cross-entropy loss.

Algorithm 1 Graph Convolutional Network (forward propagation)

Input: Graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$; Input features $\{\mathbf{x}_j, \forall j \in \mathcal{V}\}$; Number of iterations T; Weight matrices $\mathbf{W}^t, \forall t \in \{1, \ldots, T\}$ for graph embedding; Output layer weight matrix $\mathbf{W}^{\text{out}}$; Non-linearity σ (the relu function); Non-linearity σ_s (the sigmoid function); Neighborhood function $\mathcal{N}$; Attention coefficients $\boldsymbol{\alpha}$.

Output: Predicted value of binary variables: $z_v, \forall v \in \mathcal{V}_V$.

1: $\boldsymbol{h}_v^0 \leftarrow \text{EMBEDDING}(\mathbf{x}_v), \forall v \in \mathcal{V}_V$

2: $\boldsymbol{h}_c^0 \leftarrow \text{EMBEDDING}(\mathbf{x}_c), \forall c \in \mathcal{V}_C$

3: $\boldsymbol{h}_o^0 \leftarrow \text{EMBEDDING}(\mathbf{x}_o)$

4: **for** $t = 1, \ldots, T$ **do**

5: $\quad \boldsymbol{h}_o^t \leftarrow \sigma\Big(\mathbf{W}_{Vo}^t \cdot \text{CONCAT}\big(\boldsymbol{h}_o^{t-1}, \sum_{v \in \mathcal{V}_V \cap \mathcal{N}(o)} \alpha_{vo} \boldsymbol{h}_v^{t-1}\big)\Big)$

6: $\quad$ **for** c in $\mathcal{C}$ **do** :

7: $\quad\quad \boldsymbol{h}_o^t \leftarrow \sigma\Big(\mathbf{W}_{oC}^t \cdot \text{CONCAT}\left(\boldsymbol{h}_o^t, \boldsymbol{h}_c^{t-1}\right)\Big)$

8: $\quad\quad \boldsymbol{h}_c^t \leftarrow \sigma\Big(\mathbf{W}_{Vc}^t \cdot \text{CONCAT}\big(\boldsymbol{h}_o^t, \sum_{v \in \mathcal{V}_V \cap \mathcal{N}(c)} \alpha_{vc} \boldsymbol{h}_v^{t-1}\big)\Big)$

9: $\quad \boldsymbol{h}_o^t \leftarrow \sigma\Big(\mathbf{W}_{Co}^t \cdot \text{CONCAT}\big(\boldsymbol{h}_o^t, \sum_{c \in \mathcal{V}_C \cap \mathcal{N}(o)} \alpha_{co} \boldsymbol{h}_c^t\big)\Big)$

10: $\quad$ **for** v in $\mathcal{V}$ **do** :

11: $\quad\quad \boldsymbol{h}_o^t \leftarrow \sigma\Big(\mathbf{W}_{oV}^t \cdot \text{CONCAT}\left(\boldsymbol{h}_o^t, \boldsymbol{h}_v^{t-1}\right)\Big)$

12: $\quad\quad \boldsymbol{h}_v^t \leftarrow \sigma\Big(\mathbf{W}_{Cv}^t \cdot \text{CONCAT}\big(\boldsymbol{h}_o^t, \sum_{c \in \mathcal{V}_C \cap \mathcal{N}(v)} \alpha_{cv} \boldsymbol{h}_c^t\big)\Big)$

13: $z_v \leftarrow \sigma_s\left(\mathbf{W}^{\text{out}} \cdot \text{CONCAT}\left(\boldsymbol{h}_v^0, \boldsymbol{h}_v^T\right)\right), \forall v \in \mathcal{V}_V$

Nodes' representations in the tripartite graph are updated via a 4-step procedure. In the first step (line 5 in Algorithm 1), the objective node o aggregates the representations of all variable nodes $\{\boldsymbol{h}_v, v \in \mathcal{V}_V\}$ to update its representation $\boldsymbol{h}_o$. The "CONCAT" operation represents the CONCATENATE function that joins two arrays. In the second step (lines 6-8), $\{\boldsymbol{h}_v, v \in \mathcal{V}_V\}$ and $\boldsymbol{h}_o$ are used to update the representations of their neighboring constraint node $c \in \mathcal{V}_C$. In the third step (line 9), representations of constraints $\{\boldsymbol{h}_c, c \in \mathcal{V}_C\}$ are aggregated to update $\boldsymbol{h}_o$, while in the fourth step (lines 10-12), $\{\boldsymbol{h}_c, c \in \mathcal{V}_C\}$ and $\boldsymbol{h}_o$ are combined to update $\{\boldsymbol{h}_v, v \in \mathcal{V}_V\}$. See Fig. 2 for an illustration of information transition flow in the trigraph. After T transitions, two fully-connected layers coupled with a sigmoid activation function σ_s is used for solution value prediction of each $v \in \mathcal{V}_V$ (line 13).

The intuition behind Algorithm 1 is that at each iteration, a variable node incrementally gathers more aggregation information from its neighboring nodes, which correspond to the related constraints and variables in the MIP formulation. It is worth mentioning that these transitions only extract connection relations among the nodes, ignoring the detailed coefficients numerics $\boldsymbol{A}$, $\boldsymbol{b}$ and $\boldsymbol{c}$. To enhance the representation ability of our model, we include an attention mechanism to import information from the coefficients' values.

Attention Mechanism: A distinct feature in the tripartite graph structure is the heterogeneities in nodes and arcs. Rather than using a shared linear transformation (i.e., weight matrix) for all nodes (Veličković et al. 2017), we consider different transformations in each step of graph embedding updates, reflecting the importance of feature of one type of nodes on the other. In particular, given node i of type T_i and

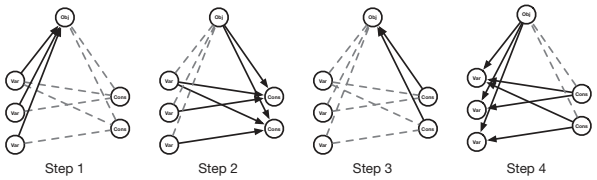

Figure 2: Information transition flow in the trigraph convolutional layer

The information transitions run consecutively as follows: Step 1, transform variable nodes information to the objective node; Step 2, transform the objective and variable nodes information to constraint nodes; Step 3, transform constraint nodes information to the objective node; Step.4, transform the objective node and constraint nodes information to variable nodes.

node j of type T_j, the attention coefficient which indicates the importance of node $i \in \mathcal{V}$ from its neighbor $j \in \mathcal{N}(i)$ is computed as:

$$\alpha_{ij} = \sigma_s \left(\mathbf{W}^{\text{att}}_{T_i,T_j} \cdot \text{CONCAT} \left(\boldsymbol{h}_i, \boldsymbol{h}_{e_{ij}}, \boldsymbol{h}_j \right) \right), \quad (5)$$

where $\boldsymbol{h}_i, \boldsymbol{h}_j, \boldsymbol{h}_{e_{ij}}$ are embeddings of node $i, j \in \mathcal{V}$ and edge $(i, j) \in \mathcal{E}$ respectively, σ_s is the sigmoid activation function, and $\mathbf{W}^{\text{att}}_{T_i,T_j}$ is the attention weight matrix between type T_i and T_j nodes. For each $i \in \mathcal{V}$, the attention coefficient is normalized cross over all neighbor nodes $j \in \mathcal{N}(i)$ using a softmax function. With this mechanism, coefficients information in $\boldsymbol{A}, \boldsymbol{b}$ and $\boldsymbol{c}$ (all of which contained in the feature vector of the edges) are incorporated to reflect edge connection importance in the graph.

Prediction-Based MIP Solving

Next, we introduce how the solution value prediction results are utilized to improve MIP solving performance. One approach is to add a local branching (Fischetti and Lodi 2003) type (invalid) global cut to the MIP model to reduce the search space of feasible solutions. This method aims to identify decision variables that are predictable and stable, and restrict the B&B tree search on unpredictable variables to accelerate primal solution-finding. An alternative approach is to perform an actual branching at the root node that maintains global optimality. These two methods are detailed as follows.

a) Approximate approach. Let $\hat{x}_j$ denote the predicted solution value of binary variable $x_j, j \in \mathcal{B}$, and let $\mathcal{S} \subseteq \mathcal{B}$ denote a subset of indices of binary variables. A local branching initial cut to the model is defined as:

$$\Delta(\boldsymbol{x}, \hat{\boldsymbol{x}}, \mathcal{S}) = \sum_{j \in \mathcal{S}: \hat{x}^k_j = 0} x_j + \sum_{j \in \mathcal{S}: \hat{x}^k_j = 1} (1 - x_j) \leq \phi, \quad (6)$$

where ϕ is a problem parameter that controls the maximum distance from a new solution $\boldsymbol{x}$ to the predicted solution $\hat{\boldsymbol{x}}$. Adding cuts with respect to subset $\mathcal{S}$ rather than $\mathcal{B}$ is due to the unpredictable nature of unstable variables in MIP solutions. Therefore, only those variables with high probability to take value 0 or 1 are included in $\mathcal{S}$. For the extreme case that ϕ equals 0, the initial cut is equivalent to fixing variables with indices in $\mathcal{S}$ at their predicted values. It is worth mentioning that the inclusion of global constraint (6) shrinks the feasible solution region of the original model, trading optimality for speed as an approximate approach.

b) Exact approach. The proposed local branching type cut can also be incorporated in an exact solver by branching on the root node. To do this, we create two child nodes from the root as follows:

$$\text{Left: } \Delta(\boldsymbol{x}, \hat{\boldsymbol{x}}, \mathcal{S}) \leq \phi, \quad \text{Right: } \Delta(\boldsymbol{x}, \hat{\boldsymbol{x}}, \mathcal{S}) \geq \phi + 1,$$

which preserves all feasible solution regions in the tree search. After the root node branching, we switch back to the solver's default setting and perform an exact B&B tree search process.

Data Collection

Features: An ideal feature collection procedure should capture sufficient information to describe the MIP solving process, and being of low computational complexity as well. A good trade-off between these two concerns is to collect features at the root node of the B&B tree, where the problem has been presolved to eliminate redundant variables and constraints and the LP relaxation is solved. In particular, we collect for each instance 3 types of features: variable features, constraint features, and edge features. Features descriptions are summarized in Table 1 in Appendix A.

As presented in the feature table, features of variables and constraints can be divided into three categories: basic features, LP features, and structure features. The structure features (most of which can be found in (Alvarez, Louveaux, and Wehenkel 2017; Khalil et al. 2016)) are usually hand-crafted statistics to reflect correlations between variables and constraints. It is noticeable that our tripartite graph neural network model can naturally capture these correlations without human expertise and can generate more advanced structure information to improve prediction accuracy. This will be verified in the computational evaluations section.

Labels: To make predictions on solution values of binary variables, an intuitive labeling scheme for the variables is to label them with the optimal solution values. Note, however, obtaining optimal solutions for medium or large scale MIP instances can be very time-consuming or even an impossible task. In the situation when optimal solutions are difficult to obtain, we propose to identify *stable* and *unstable* variables in solutions. This is motivated by the observation that solution values of the majority of binary decision variables remain unchanged across a series of different feasible solutions. To be specific, given a set of K solutions $\{\bar{\boldsymbol{x}}^1, \ldots, \bar{\boldsymbol{x}}^K\}$ to a MIP instance I, a binary variable x_j is defined as unstable if there exists some $k_1, k_2 \in \{1, \ldots, K\}$ such that $x^{k_1}_j \neq x^{k_2}_j$, and as stable otherwise. Although obtaining optimal solutions might be a difficult task, the stable variables can be viewed as an easy-to-predict part in the (near) optimal solutions. To generate a sequence of solutions to an instance, we use the proximity search method (Fischetti and Monaci 2014). Starting from some initial solution $\bar{\boldsymbol{x}}^k$ with objective

value $\boldsymbol{c}^T\bar{\boldsymbol{x}}^k$, a neighborhood solution with the objective value improvement being at least δ can be generated by solving the following optimization:

$$\min \sum_{j\in\mathcal{B}:\bar{x}_j^k=0} x_j + \sum_{j\in\mathcal{B}:\bar{x}_j^k=1} (1-x_j) \tag{7}$$

$$\text{s.t. } \boldsymbol{c}^T\boldsymbol{x} \le \boldsymbol{c}^T\bar{\boldsymbol{x}}^k - \delta, \tag{8}$$

$$\boldsymbol{A}\boldsymbol{x} \le \boldsymbol{b}, \tag{9}$$

$$x_j \in \{0,1\}, \forall j \in \mathcal{B}, \tag{10}$$

$$x_j \in \mathbb{Z}, \forall j \in \mathcal{G};\ \ x_j \ge 0, \forall j \in \mathcal{C}, \tag{11}$$

where the objective function represents the distance between $\bar{\boldsymbol{x}}^k$ and a new solution $\boldsymbol{x}$. Note that the above optimization is computationally tractable since solving process can terminate as soon as a feasible solution is found. By iteratively applying this method, we obtain a series of improving feasible solutions to the original problem. Stable binary variables are labeled with their solution values while the unstable variables are marked as unstable and discarded from the training set. A limitation of this labeling method is the inability of handling the case when the initial feasible solution $\bar{\boldsymbol{x}}^0$ is hard to obtain.

Remark. The logic behind the stable variable labeling scheme is to explore local optimality patterns when global optimality is not accessible. In each iteration of proximity search, a neighboring better solution is found, with a few flips on solution values of the binary variables. Performing this local search step for many rounds can identify local minimum patterns which reflect domain knowledge of the CO problem. Take the Traveling Salesman Problem (TSP) as an example. Let z_{jl} define whether node l is visited immediately after node j. If j and l are geometrically far away from each other, z_{jl} is likely to be zero in all the solutions generated by proximity search and being recorded by our labeling scheme, reflecting the underlying local optimality knowledge for TSP.

Experimental Evaluations

Setup. To evaluate the proposed framework, we modify the state-of-the-art open-source MIP solver SCIP (version 6.0.1) for data collection and solution quality comparison. The GCN model is built using the Tensorflow API. All experiments were conducted on a cluster of three 4-core machines with Intel 2.2 GHz processors and 16 GB RAM.

Instances. To test the effectiveness and generality of the prediction-based solution framework, we generate MIP instances of 8 distinct types: *Fixed Charge Network Flow* (FCNF), *Capacitated Facility Location* (CFL), *Generalized Assignment* (GA), *Maximal Independent Set* (MIS), *Multidimensional Knapsack* (MK), *Set Covering* (SC), *Traveling Salesman Problem* (TSP) and *Vehicle Routing Problem* (VRP). These problems are the most commonly encountered NP-hard combinatorial optimizations in OR and are quite general because they differ significantly in MIP structures and solution structures. For each problem type, 200 MIP instances of similar scales are generated. The number of instances used for training, validation, and testing is 140, 20, 40 respectively. Parameter calibrations are performed on the validation set, while prediction accuracy and solution quality comparisons are evaluated on the test instances. Detailed MIP formulation and instance parameters of each type are included in Appendix B.

Data collection. In terms of feature collection, we implemented a feature extraction plugin embedded in the branching procedure of SCIP. In particular, variable features, constraint features, and edge features are collected right before the first branching decision is made at the root node, where the presolving process, root LP relaxation, and root cutting plane have completed. No further exploration of the B&B search tree is needed and thus the feature collection process terminates at the root node. Construction of the tripartite graph is also completed at the root node where SCIP is working with a transformed MIP model such that redundant variables and constraints have been removed. In terms of label collection, since optimal solutions to all problem types can not be obtained within a 10000 seconds time limit, we applied the proximity search method to label stable binary variables. The initial solution $\bar{\boldsymbol{x}}^0$ for proximity search is obtained under SCIP's default setting with a 300 seconds execution time limit. If SCIP fails to obtain an initial feasible solution within the time limit, the time limit doubles until a feasible initial solution can be found. Parameter δ for the proximity search is set as $0.01 \cdot (\boldsymbol{c}^T\bar{\boldsymbol{x}}^0 - \text{LB})$ where LB is the lower bound. Each proximity search iteration terminates as soon as a feasible solution is found. This process generally converges within 20 to 40 iterations.

Parameter calibration. We applied the orthogonal initialization for $\mathbf{W}^t$, and updated $\mathbf{W}^t$ by ADAM with a 0.001 learning rate and a mini-batch size of 8. We decrease the learning rate by a factor of 5 if the validation loss does not decrease for 10 consecutive epochs and stop training for 50.

Table 1: Hyper-parameter selections for each problem type

	FCNF	CFL	GA	MIS	MK	SC	TSP	VRP
ϕ	0	0	5	10	10	0	0	5
η	0.80	0.95	0.99	0.90	0.80	0.90	0.90	0.95

The performance of the proposed framework benefits from a proper selection of hyper-parameters ϕ and $\mathcal{S}$. Let z_j denote the prediction probability that binary variable $x_j, j \in \mathcal{B}$ takes value 1 in the MIP solution. We sort x_j in non-decreasing order of $\min(z_j, 1-z_j)$ and choose the first $\eta \cdot |\mathcal{B}|$ variables as $\mathcal{S}$. The strategy of tuning $\phi \in \{0, 5, 10, 15, 20\}$ and $\eta \in \{0.8, 0.9, 0.95, 0.99, 1\}$ is grid search, where the combination of ϕ and η that results in best solution quality on the validation set is selected. Table 1 summarizes ϕ and η selections for each problem type.

Results of Solution Prediction

We demonstrate the effectiveness of the proposed GCN model on prediction accuracy against the XGBoost (XGB) classifier (Chen and Guestrin 2016). For XGB, only variable features are used for prediction since it can not process the trigraph

information. Noting that solution values of binary variables are usually highly imbalanced, we use the *average precision* (AP) metric (Zhu 2004) to evaluate the performance of the classifiers. In particular, the AP value is defined as:

$$\text{AP} = \sum_{k=1}^{n} P(k)\Delta r(k), \tag{12}$$

where k is the rank in the sequence of predicted variables, $P(k)$ is the precision at cut-off k in the list, and $\Delta r(k)$ is the difference in recall from $k-1$ to k.

Table 2: Comparisons on the average precision metric

	Basic		Basic&structure		All	
Instances	XGB	GCN	XGB	GCN	XGB	GCN
FCNF	0.099	**0.261**	0.275	**0.317**	0.787	**0.788**
CFL	0.449	**0.590**	0.567	**0.629**	0.846	**0.850**
GA	0.499	**0.744**	0.750	**0.797**	0.936	**0.937**
MIS	0.282	**0.355**	0.289	**0.337**	0.297	**0.325**
MK	0.524	**0.840**	0.808	**0.843**	0.924	**0.927**
SC	0.747	**0.748**	0.748	**0.753**	**0.959**	**0.959**
TSP	0.327	**0.358**	0.349	**0.353**	0.401	**0.413**
VRP	0.391	**0.403**	0.420	**0.424**	0.437	**0.459**
Average	0.415	**0.537**	0.526	**0.556**	0.698	**0.707**

Table 2 describes the AP value comparison results for the two classifiers under three settings: using only basic features, using basic&structure features, and using all features respectively. It is observed that the proposed GCN model outperforms the baseline classifier in all settings. The performance advantage is particularly significant in the basic feature columns (0.537 by GCN against 0.415 by XGB), where only raw coefficient numerics in MIP are used for prediction. The other notable statistic in the table is that the GCN model with only basic features is on average superior to the XGB classifier with basic&structure features, indicating that the proposed embedding framework can extract more information compared to hand-crafted structure features used in the literature (Alvarez, Louveaux, and Wehenkel 2017; Khalil et al. 2016). For comparisons in the all features column, the advantage of GCN is less significant due to the reason that high-level MIP structure information is also captured in its LP relaxations.

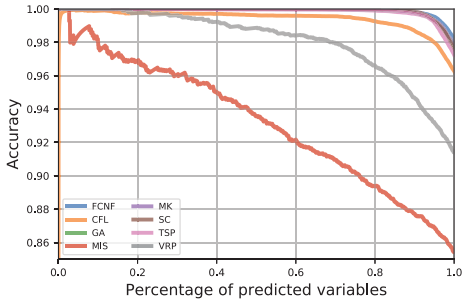

Figure 3: Accuracy under different prediction percentage

To help illustrate the predictability in solution values for problems of different types, we present in Fig.3 the detailed accuracy curve for each of the 8 problem types using the GCN model with all features. The figure depicts the prediction accuracy if we only predict a certain percentage of the most predictable binary variables[2]. It can be observed from the figure that solution values of most considered MIP problems (such as FCNF, GA, MK, SC,TSP) are fairly predictable, with an almost 100% accuracy if we make solution value prediction on the top 0-80% most predictable variables. Among the tested problem types, solution values to MIS and VRP problem instances are fairly unpredictable, which is consistent with the hyper-parameter calibration outcomes that ϕ takes value greater than 0.

Comparisons of solution quality

a) Approximate approach: To evaluate the value of incorporating the (invalid) global cut in MIP solving, we compare the performance of prediction-based approximate approach against that of the solver's aggressive heuristics setting[3]. To be specific, we modified SCIP's setting to "set heuristics emphasis aggressive" to make SCIP focus on solution finding rather than proving optimality. For each problem type, 10 MIP instances are randomly selected from the 40 testing instances for solution quality comparisons.

Notice that the proposed approximate approach does not guarantee global optimality (i.e., does not provide a valid lower bound), we use the primal gap metric (Khalil et al. 2017) to capture solver's performance on primal solution finding. In particular, the *primal gap* metric P-Gap($\tilde{\boldsymbol{x}}$) reports the relative gap in the objective value of a feasible solution $\tilde{\boldsymbol{x}}$ to that of the optimal (or best-known) solution $\tilde{\boldsymbol{x}}^*$:

$$\text{P-Gap}(\tilde{\boldsymbol{x}}) = \frac{|\boldsymbol{c}^T\tilde{\boldsymbol{x}} - \boldsymbol{c}^T\tilde{\boldsymbol{x}}^*|}{\max\{|\boldsymbol{c}^T\tilde{\boldsymbol{x}}|, |\boldsymbol{c}^T\tilde{\boldsymbol{x}}^*|\} + \epsilon} \times 100\%, \tag{13}$$

where $\epsilon = 10^{-10}$ is a small constant to avoid numerical error when $\max\{|\boldsymbol{c}^T\tilde{\boldsymbol{x}}|, |\boldsymbol{c}^T\tilde{\boldsymbol{x}}^*|\} = 0$. Since all problem types are not solvable within a 10000 seconds time limit (under the SCIP's default setting without the global cut), $\tilde{\boldsymbol{x}}^*$ is selected as the best-known solution found by all tested methods.

Table 3 is a collection of solution quality results by the proposed method with a 1000 seconds execution time limit. To demonstrate the significance of performance improvement, we allow the solver to run for 2-10 times the execution time (i.e., 2000-10000 seconds) in aggressive heuristics setting. It is revealed from the table that the proposed approximate method (the GCN-A columns) gains remarkable advantages in terms of the P-Gap metric to the SCIP's aggressive heuristics setting under the same time limit (the AGG1 column) on all testing problems. Compared to the setting with 10000 seconds (the AGG10 column), the proposed framework still maintains an average better P-Gap performance, indicating

[2]The predictability of a binary variable x_j is measured by $\max(z_j, 1-z_j)$.

[3]As far as we know, there are hardly any stable approximate solvers for MIP and "set heuristics emphasis aggressive" is the most relevant setting we find to accelerate primal solution-finding in the SCIP's documentation. Therefore we use this setting as a benchmark for comparison.

Table 3: Primal gap comparisons for the approximate approach

	AGG1*		AGG2		AGG5		AGG10		GCN-A	
	P-Gap(%)	Heur-T(s)	P-Gap	Heur-T	P-Gap	Heur-T	P-Gap	Heur-T	P-Gap	Heur-T
FCNF	1.733	303.9	1.733	538.1	1.678	821.7	0.380	1182.8	**0.000**	1000.0
CFL	2.334	284.8	2.112	683.6	0.868	1594.0	0.206	2818.4	**0.092**	1000.0
GA	0.451	499.4	0.430	958.8	0.430	1992.1	0.430	3619.0	**0.000**	1000.0
MIS	10.292	352.5	6.125	480.0	5.083	575.0	5.083	696.2	**1.042**	1000.0
MK	0.003	233.1	0.003	378.4	**0.000**	665.5	**0.000**	1006.6	0.003	1000.0
SC	1.509	707.4	0.529	1120.5	0.383	1796.5	**0.000**	2851.1	0.164	1000.0
TSP	10.387	475.4	6.286	972.3	2.752	1896.0	1.981	2968.7	**0.332**	1000.0
VRP	3.096	381.6	3.096	756.9	1.177	1869.0	1.131	3800.5	**0.896**	1000.0
Average	3.726	404.8	2.539	736.1	1.546	1401.2	1.151	2367.9	**0.316**	1000.0

* AGG1 represents SCIP's aggressive heuristics setting with 1×1000 seconds execution time limit. Similarly, AGG2, AGG5 and AGG10 correspond to aggressive heuristics setting with 2×1000, 5×1000 and 10×1000 seconds time limit respectively.

a 10 times acceleration in solution-finding, with the trade of optimality guarantees. Note that SCIP with AGG setting still spends quite some time proving optimality, we report the average running time of heuristics for AGG (the Heur-T columns). It is revealed from table 3 that AGG10 needs an average higher running time (2367.9s) in heuristics compared to the total running time (1000s) by GCN-A.

b) Exact approach To evaluate the value of performing an actual branching at the root, we compare the performance of the prediction-based exact approach against that of the solver's default setting with a 1000 seconds time limit. Because the exact approach provides a valid lower bound to the original problem, we use the well-known *optimality gap* metric O-Gap($\tilde{\boldsymbol{x}}$) to measure the MIP solving performance:

$$\text{O-Gap}(\tilde{\boldsymbol{x}}) = \frac{|\boldsymbol{c}^T\tilde{\boldsymbol{x}} - \text{LB}|}{|\boldsymbol{c}^T\tilde{\boldsymbol{x}}| + \epsilon} \times 100\%, \tag{14}$$

where $\tilde{\boldsymbol{x}}$ and LB denote respectively the primal solution and best lower bound obtained by a specific method.

Table 4: O-Gap comparisons for the exact approach (%)

	FCNF	CFL	GA	MIS	MK	SC	TSP	VRP
DEF	7.06	3.96	5.30	62.92	2.01	12.09	**12.13**	68.69
GCN-E	**4.78**	**3.64**	**4.31**	**61.75**	**2.00**	**11.19**	12.84	**68.13**

We conclude the experimental results in Table 4. The DEF row corresponds to SCIP's default setting and GCN-E corresponds to the exact approach using the new root branching rule. It is revealed from the table that GCN-E outperforms DEF in terms of the optimality gap within the time limit. This provides empirical evidence that the proposed method is potentially useful for accelerating MIP to global optimality.

To help understand how the new branching rule accelerates the MIP solving process, we plot the B&B tree of a small-scale "CFL" instance in Fig. 4. It is observed from the tree search process that the solver finds a good solution quickly on the left tree, and goes to the right tree to get the optimal solution, and finally prove optimality by exploring the remaining nodes on both sides.

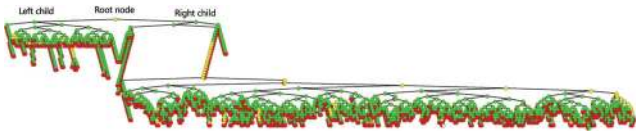

Figure 4: Visualization of the B&B tree after performing a root branching based on GCN prediction.

Generalization to larger instances

The graph embedding framework endows the model to train and test on MIP instances of different scales. This is important for MIP solving since there is hardly any good strategy to handle large-scale NP-hard MIP problems. To investigate this, we generate 200 small-scale MIP instances for each problem type and train our GCN model on these instances and test its applicability in large-scale ones. Detailed statistics of small and large instances are reported in Appendix B.

Table 5: Generalization ability of the proposed framework

	Average precision		Primal gap (%)		
	GCN	GCNG*	AGG1	GCN-A	GCNG-A
FCNF	0.653	0.675	1.733	0.000	2.119
CFL	0.837	0.801	2.341	0.100	0.410
GA	0.963	0.873	0.661	0.211	0.016
MIS	0.091	0.104	2.223	1.136	1.087
MK	0.789	0.786	0.000	0.000	0.000
SC	0.878	0.843	1.349	0.000	0.215
TSP	0.396	0.343	10.061	0.000	4.802
VRP	0.358	0.321	4.408	1.254	1.239
Average	0.621	0.593	2.847	0.338	1.236

* GCNG is the GCN model trained on small-scale MIP instances. GCNG-A is the approximate solving approach based on the GCNG prediction model.

It is revealed from table 5 that the GCN model maintains an acceptable prediction accuracy degradation when the problem scale differs in the training and testing phase. Besides, the prediction result is still useful to improve solver's primal solution finding performance.

Conclusions

We presented a supervised solution prediction framework to explore the correlations between the MIP formulation structure and its local optimality patterns. The key feature of the model is a tripartite graph representation for MIP, based on which graph embedding is used to extract connection information among variables, constraints and the objective function. Through experimental evaluations on 8 types of MIP problems, we demonstrate the effectiveness and generality of the GCN model in prediction accuracy. Incorporated in a global cut to the MIP model, the prediction results help to accelerate SCIP's solution-finding process by 10 times on similar problems with a sacrifice in proving global optimality. This result is inspiring to practitioners facing routinely large-scale MIPs on which the solver's execution time is unacceptably long and tedious, while global optimality is not a major concern.

Limitations of the proposed framework are two-fold. First, this method is better being applied to binary variable intensive MIP problems due to the difficulties in solution value prediction for general integer variables. Second, the prediction performance degrades for problems without local optimality structure where correlations among variables from the global view can not be obtained from the neighborhood information reflected in the MIP's trigraph representation.

References

Alvarez, A. M.; Louveaux, Q.; and Wehenkel, L. 2017. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing* 29(1):185–195.

Alvarez, A. M.; Wehenkel, L.; and Louveaux, Q. 2016. Online learning for strong branching approximation in branch-and-bound.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2018. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint arXiv:1811.06128*.

Chen, T., and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794. ACM.

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

Cook, W. J. 2011. *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton University Press.

Dai, H.; Khalil, E.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, 6348–6358.

Dósa, G., and Sgall, J. 2013. First fit bin packing: A tight analysis. In *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Farahani, R. Z., and Hekmatfar, M. 2009. *Facility location: concepts, models, algorithms and case studies*. Springer.

Fischetti, M., and Lodi, A. 2003. Local branching. *Mathematical programming* 98(1-3):23–47.

Fischetti, M., and Monaci, M. 2014. Proximity search for 0-1 mixed-integer convex programming. *Journal of Heuristics* 20(6):709–731.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*.

He, H.; Daume III, H.; and Eisner, J. M. 2014. Learning to search in branch and bound algorithms. In *Advances in neural information processing systems*, 3293–3301.

Khalil, E. B.; Le Bodic, P.; Song, L.; Nemhauser, G. L.; and Dilkina, B. N. 2016. Learning to branch in mixed integer programming. In *AAAI*, 724–731.

Khalil, E. B.; Dilkina, B.; Nemhauser, G. L.; Ahmed, S.; and Shao, Y. 2017. Learning to run heuristics in tree search. In *26th International Joint Conference on Artificial Intelligence (IJCAI)*.

Kool, W., and Welling, M. 2018. Attention solves your tsp. *arXiv preprint arXiv:1803.08475*.

Kool, W.; van Hoof, H.; and Welling, M. 2018. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*.

Kruber, M.; Lübbecke, M. E.; and Parmentier, A. 2017. Learning when to use a decomposition. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 202–210. Springer.

Laporte, G. 2009. Fifty years of vehicle routing. *Transportation Science* 43(4):408–416.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, 537–546.

2018. MIPLIB 2017. http://miplib.zib.de.

Nazari, M.; Oroojlooy, A.; Snyder, L.; and Takác, M. 2018. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, 9839–9849.

Pinedo, M. 2012. *Scheduling*, volume 29. Springer.

Selsam, D.; Lamm, M.; Bünz, B.; Liang, P.; de Moura, L.; and Dill, D. L. 2018. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*.

Tang, Y.; Agrawal, S.; and Faenza, Y. 2019. Reinforcement learning for integer programming: Learning to cut. *arXiv preprint arXiv:1906.04859*.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, 2692–2700.

Zhu, M. 2004. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo* 2:30.