MIP-GNN: A data-driven tool for guiding combinatorial solvers

Elias B. Khalil[1] and Christopher Morris[2]

[1]CERC in Data Science for Real-Time Decision-Making, Polytechnique Montréal
[2]Department of Mechanical & Industrial Engineering, University of Toronto

Write abstract. Theme: general data-driven tool for replacing heuristic components of MIP solvers.

1 Introduction

Write introduction, main points (MIP-GNN is a general purpose tool for guiding decision throughout the solving process of MIPs in a data-driven way):

- MIPs solvers use many heuristics throughout solving process, need for data-driven heuristics that adapt to the problem distribution
- Graphs as a natural way to represent BIPs/ILPs/MIPs and other constraint convex optimization problems, graphs are a natural inductive bias
- Arguments of Fischetti

Present work Contribution.

- General approach, many different applications within the MIP/B&C solving framework, list examples
- Connection to MWU problem (theoretically principled, MWU can be recovered)
- Experimental evaluation on real-world instances, speed-up of X

Related work Discuss related work.

- Most important GNN papers,
- GNNs to solve MIPs/comb. optimization problems,
- Papers on SAT and CSP solving (Hsu, LeSong, Aachen, Dilkina, Khalil, ...)

2 Preliminaries

In the following, we fix notation and give a short introduction to GNNs, binary integer programs, and setup the learning program.

2.1 Notation

[CM: shorten] A *graph* G is a pair (V, E) with a *finite* set of *vertices* V and a set of *edges* $E \subseteq \{\{u, v\} \subseteq V \mid u \neq v\}$. We denote the set of vertices and the set of edges of G by $V(G)$ and $E(G)$, respectively. For ease of notation, we denote the edge $\{u, v\}$ in $E(G)$ by (u, v) or (v, u). In the case of *directed graphs* $E \subseteq \{(u, v) \in V \times V \mid u \neq v\}$. A *labeled graph* G is a triple (V, E, l) with a label function $l \colon V(G) \cup E(G) \to \Sigma$, where Σ is some finite alphabet. Then $l(v)$ is a *label* of v for v in $V(G) \cup E(G)$. The *neighborhood* of v in $V(G)$ is denoted by $\delta(v) = N(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$. Moreover, its complement $\overline{\delta}(v) = \{u \in V(G) \mid (v, u) \notin E(G)\}$. Let $S \subseteq V(G)$ then $G[S] = (S, E_S)$ is the *subgraph induced* by S with $E_S = \{(u, v) \in E(G) \mid u, v \in S\}$. A *tree* is a connected graph without cycles. A *rooted tree* is a tree with a designated vertex called *root* in which the edges are directed in such a way that they point away from the root. [CM: biparite graph] Let p be a vertex in a directed tree then we call its out-neighbors *children* with parent p. [CM: Add matrix notation, all-row vectors] We say that two graphs G and H are *isomorphic* if there exists an edge preserving bijection $\varphi \colon V(G) \to V(H)$, i.e., (u, v) is in $E(G)$ if and only if $(\varphi(u), \varphi(v))$ is in $E(H)$. If G and H are isomorphic, we write $G \simeq H$ and call φ an *isomorphism* between G and H. Moreover, we call the equivalence classes induced by $\simeq$ *isomorphism types*, and denote the isomorphism type of G by τ_G. In the case of labeled graphs, we additionally require that $l(v) = l(\varphi(v))$ for v in $V(G)$ and $l((u, v)) = l((\varphi(u), \varphi(v)))$ for (u, v) in $E(G)$. Moreover, let $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ for $n \geq 1$, and let $\{\!\{\ldots\}\!\}$ denote a multiset.

2.2 Binary integer programs

An *instance* I of binary integer program (BIP) is a 3-tuple $(\mathbf{c}, \mathbf{A}, \mathbf{b})$, with a vector $\mathbf{c}$ in a matrix $\mathbf{A}$ in $\mathbb{R}^{n \times m}$, and $\mathbf{b}$ in $\mathbb{R}^m$. We interpret the former as a (linear) *objective* or *cost function*, and the latter two are interpreted as a system of linear equations, i.e., $\mathbf{Ax} \geq \mathbf{b}$, which induces a set of *feasible solutions* $F(I) = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \geq 0\}$. We call each component of $\mathbf{x}$ a *variable*, and the set $\mathcal{V}(I) = \{x_i\}_{i \in [n]}$ the (set of) variables.

The *optimal integral solution* $\mathbf{x}^*$ is equal to $\arg\max_{\mathbf{x} \in \{0,1\}^n} \{\mathbf{c}^\mathsf{T}\mathbf{x} \mid \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \geq 0\}$, i.e., we maximize the linear functional $\mathbf{c}$ over $F(I)_{0,1} = F(I) \cap \{0, 1\}^n$. The *optimal solution* $\bar{\mathbf{x}}$ of the *relaxed BIP* is obtained by dropping the integrality contraints, i.e., $\bar{\mathbf{x}} = \arg\max_{\mathbf{x} \in [0,1]^n} \{\mathbf{c}^\mathsf{T}\mathbf{x} \mid \mathbf{Ax} \geq \mathbf{b}, \mathbf{x} \geq 0\}$. In other words, we maximize the linear functional $\mathbf{c}$ over $F(I)$.

[CM: State connection to comb. optimization]

2.3 Graph neural networks

Let G be a labeled graph (V, E, l) with an initial vertex coloring $f^{(0)} \colon V(G) \to \mathbb{R}^{1 \times d}$ that is *consistent* with l. That is, each vertex v is annotated with a feature $\mathbf{f}^{(0)}(v)$ in $\mathbb{R}^{1 \times d}$ such that $\mathbf{f}^{(0)}(u) = \mathbf{f}^{(0)}(v)$ if and only if $l(u) = l(v)$. Alternatively, $\mathbf{f}^{(0)}(v)$ can be an arbitrary real-valued feature vector associated with v. A GNN architecture consists of a stack of neural network layers, where each layer aggregates local neighborhood information, i.e., features of neighbors, and then passes this aggregated information on to the next layer. In each round or layer t a new feature $\mathbf{f}^{(t)}(v)$ is computed as

$$f_{\mathrm{UP}}^{\mathbf{W_2}}\Big(\mathbf{f}^{(t-1)}(v), f_{\mathrm{AGG}}^{\mathbf{W_1}}\big(\{\!\{\mathbf{f}^{(t-1)}(w) \mid w \in N(v)\}\!\}\big)\Big), \tag{1}$$

where $f_{\text{aggr}}^{\mathbf{W_1}}$ aggregates over the set of neighborhood features and $f_{\text{merge}}^{\mathbf{W_2}}$ merges the vertex's representations from step $(t-1)$ with the computed neighborhood features. Both $f_{\text{aggr}}^{\mathbf{W_1}}$ and $f_{\text{merge}}^{\mathbf{W_2}}$ may be arbitrary differentiable, (permutation-invariant) functions (e.g., neural networks), and $\mathbf{W_1}$ and $\mathbf{W_2}$, respectively, denote sets of parameters.

2.4 Setup of the learning problem

Let $\mathcal{C}$ be the set of all instances of a combinatorial optimization problem that can be formulated as a BIP. Let I be an instance in $\mathcal{C}$, then let $X_\varepsilon^*(I) = \{\mathbf{x} \in F_{0,1}(I) \mid \mathbf{c}^\mathsf{T}\mathbf{x}^* - \mathbf{c}^\mathsf{T}\mathbf{x} \leq \varepsilon\}$ be a set of integral solutions that are „close“ to the optimum of the instance I. That is, their solution value is equal to the optimal value up to a prespecified ε. The *variable bias* $\bar{\mathbf{b}}(I)$ of I with regard to $X_\varepsilon^*(I)$ then is the component-wise average over all elements, i.e., the variable bias $\bar{\mathbf{b}}(I) = 1/|X_\varepsilon^*| \sum_{\mathbf{x} \in X_\varepsilon^*(I)} \mathbf{x}$ in $\mathbb{R}^n$.

The aim here is to devise a neural architecture and train a corresponding model in a supervised fashion to predict the variable bias $\bar{\mathbf{b}}(I)$ for unseen instances. Hence, let D be a distribution over $\mathcal{C}$ and let S be a finite subset (training set) sampled uniformly and at random from D. We want to learn a function $f_\theta \colon \mathcal{V}(I) \to \mathbb{R}$, where θ represents a set of parameters in the set Θ, that predicts the variable biases of out-of-sample instances. Hence, we optimize the empirical error

$$\min_{\theta \in \Theta} 1/|S| \sum_{I \in S} \ell(f_\theta(\mathcal{V}(I)), \bar{\mathbf{b}}(I)),$$

with some loss function $\ell \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ over the set of parameters Θ.

[**CM:** Combination of GNN and MWU]

2.5 Multiplicative weights update algorithm for linear programs

The *multiplicative weights update algorihm* (MWU) is a framework to solve various computational problems that can be solved by iterativly adapting a discrete probability distribution over a certain set [1]. Here, we describe the variant of the MWU to determine the feasibility of a system of a system of linear equations.

Given a system of linear equations $\mathbf{Ax} \geq \mathbf{b}$, see Section 2.2, the MWU tries to determine if the above system is feasible up to a prespecified $\varepsilon > 0$. That is, it determines if there exists $\bar{\mathbf{x}}$ such that

$$\mathbf{A}_i \bar{\mathbf{x}} \geq b_i - \varepsilon, \tag{2}$$

for i in $[m]$.[**CM:** say sth. about optimal solution, give intution of the MWU, discrete prob., oracle for LP, Farkas' lemma.]

Algorithm 1 MWU for the LP feasibility problem.

Input: System of linear equations $\mathbf{Ax} \geq \mathbf{b}$, $\varepsilon > 0$, stepsize $\eta > 0$, scaling constant ρ, failure probability δ.
Output: $\bar{\mathbf{x}}$ satisfying Equation (2) or `non-feasible`

1: Initialize constraint weights $w_i \leftarrow 1$ for i in $[m]$
2: Set T to according Equation (3)
3: **for** t in $[T]$ **do**
4: Normalize weights $p_j \leftarrow \frac{w_j(t)}{\Phi(t)}$ for i in $[m]$, where $\Phi(t) = \sum_{i\in[m]} w_i$
5: Call oracle: $\mathbf{x}(t) \leftarrow \mathcal{O}(\mathbf{p}^\mathsf{T}\mathbf{Ax} \geq \mathbf{p}^\mathsf{T}\mathbf{b})$ if $\mathbf{x}(t) = \emptyset$ **return** `non-feasible`
6: Compute error signal $\mathbf{m}$ with

$$m_i \leftarrow {}^1\!/\!_\rho \, (\mathbf{A}_i^\mathsf{T}\mathbf{x}(t) - b_i)$$

7: Perform gradient descent by $w_i(t+1) \leftarrow (1 - \eta m_i) w_i(t)$
8: **end for**
9: **return** $\bar{\mathbf{x}} \leftarrow {}^1\!/\!_T \sum_{t\in[T]} \mathbf{x}(t)$

Algorithm 2 MWU (Message Passing version) for the LP feasibility problem.

Input: $B(I)$, $\varepsilon > 0$, stepsize $\eta > 0$, scaling constant ρ, failure probability δ.
Output: $\bar{\mathbf{x}}$ satisfying Equation (2) or `non-feasible`

1: Initialize constraint weights $\mathbf{p}_j \leftarrow {}^1\!/\!_m$ for each constraint node
2: Set T to according Equation (3)
3: **for** t in $[T]$ **do**
4: For each constraint node $\mathbf{c}_j$ send $\mathbf{p}_j$ to adjacent node variable
5: Update each variable node $\mathbf{v}_i$ based on output $\mathbf{x}_i$ of oracle using $\mathbf{p}$
6: Send $\mathbf{x}_i$ to each adjacent constraint, compute error signal $\mathbf{m}_i$ for each constraint $\mathbf{c}_j$

$$\mathbf{m}_j \leftarrow {}^1\!/\!_\rho \Big(\sum_{i\in N(j)} \mathbf{A}_{ji}\mathbf{x}_i \Big) - \mathbf{b}_j$$

7: Perform gradient descent by $\mathbf{w}_i(t+1) \leftarrow (1 - \eta \mathbf{m}_i)\mathbf{w}_i(t)$
8: Normalize weights $p_j \leftarrow \frac{w_j(t)}{\Phi(t)}$ for each constraint node, where $\Phi(t) = \sum_{i\in[m]} w_i$
9: **end for**
10: **return** $\bar{\mathbf{x}} \leftarrow {}^1\!/\!_T \sum_{t\in[T]} \mathbf{x}(t)$

We get the following bound of the running time.

Theorem 1 (E.g., [1])**.** [CM: fill in details]

$$T = \left\lceil \frac{8l\rho \ln(m)}{\varepsilon^2} \right\rceil \tag{3}$$

3 The MIP-GNN architecture

Let $I = (\mathbf{c}, \mathbf{A}, \mathbf{b})$ be a BIP. The bipartite graph $B(I)$ of I is three tuple $(V(I), C(I), E(I))$. Here, the vertex set $V(I) = \{v_i \mid x_i \in \mathcal{V}(I)\}$ represent the variables, and the set $C(I) = \{c_i \mid i \in [m]\}$ represent the constraints of I. The edge set $E(I) = \{\{v_i, c_j\} \mid A_{ij} \neq 0\}$. Further, we define (node) label function $c\colon V(I) \to \mathbb{R}$, defined by $v_i \mapsto c_i$, and the (edge) label function $a\colon E(I) \to \mathbb{R}$, defined by $(v_i, v_j) \mapsto A_{ij}$. [CM: add more label functions]

Intuitively, each round $t \geq 0$ of message-passing consists of two *passes*, the *variable to constraint* $f^{(t)}_{\mathrm{V}\to\mathrm{C}}\colon C(I) \to \mathbb{R}^{1\times d}$ pass, followed by the constraint to variable pass $f^{(t)}_{\mathrm{C}\to\mathrm{V}}\colon V(I) \to \mathbb{R}^{1\times d}$. [CM: Say sth. about the initialiazation, gaussian?] [CM: Say sth. about cost] In the following, we gives details about the two passes.

Variable to constraints.

Constraint to variable: sdfdsf

Algorithm 3 Data-driven MWU for variable bias prediction.

Input: Training set $\mathcal{S}$, $\varepsilon > 0$, number of rounds T, stepsize $\eta > 0$, scaling constant ρ.
Output: A trained model $f_{\boldsymbol{\theta}}\colon \mathcal{I} \to [0,1]^n$

Initialize $\mathbf{v}_i^{(0)}$, $\mathbf{c}_i^{(0)}$, and $\boldsymbol{\theta}$
Set weights $\mathbf{w}_j^{(0)} \leftarrow 1$, $\mathbf{p}_j^{(0)} \leftarrow 1/m$ for j in $[m]$

for e in $[E]$ **do**
  [CM: Make p a parameter?]
  Sample $I = (\mathbf{c}, \mathbf{A}, \mathbf{b})$ with bias $\bar{\mathbf{b}}$ uniformly and at random from $\mathcal{S}$ [CM: define Ib]
  for t in $[T]$ **do**
    $\mathbf{v}_i^{(t)} \leftarrow \mathbf{f}_{\mathrm{V}}^{(t)}\left(\left\{\!\!\left\{\left(\mathbf{v}_i^{(t-1)}, \mathbf{c}_j, \mathbf{A}_{ij}, \mathbf{p}_j\right)\colon j \in N(\mathbf{v}_i)\right\}\!\!\right\}\right) \quad \mathbf{x}_i^{(t)} \leftarrow \mathbf{f}_{\mathrm{V}}^{(t)}\left(\mathbf{v}_i^{(t-1)}\right)$
    $\mathbf{c}_j^{(t)} \leftarrow \mathbf{f}_{\mathrm{C}}^{(t)}\left(\left\{\!\!\left\{\left(\mathbf{c}_j^{(t-1)}, \mathbf{v}_i, \mathbf{x}_i, \mathbf{A}_{ij}, \mathbf{b}_j\right)\colon i \in N(\mathbf{c}_j)\right\}\!\!\right\}\right)$
    $\mathbf{e}_j^{(t)} \leftarrow 1/\rho \sum_{i\in N(j)} \mathbf{A}_{ji}\mathbf{x}_i^{(t)} - \mathbf{b}_j$
    $\mathbf{w}_j^{(t)} \leftarrow (1 - \eta\mathbf{e}_j)\mathbf{w}_j^{(t)}$
    $\mathbf{p}_j \leftarrow \frac{w_j^{(t)}}{\Phi(t)}$, where $\Phi(t) = \sum_{j\in[m]} \mathbf{w}_j^{(t)}$
  end for
  g
  $\bar{\mathbf{x}} \leftarrow 1/T \sum_{t\in[T]} \mathbf{x}(t)$
  $L \leftarrow L + \ell(\bar{\mathbf{x}}, \bar{\mathbf{b}})$
  Gradient descent on L
end for
return $\boldsymbol{\theta}$

[CM: overloading of c]

3.1 Unsupervised setting

random matrices, recover MWU, also test in experiments

3.2 Discussion: Road blocks and the road ahead

discuss limitation of GNNs approaches, unsupervised approaches
discuss problems of generating labeled training data

4 Experimental evaluation

Our intention here is to investigate the benefits of ... the compared to... More precisely, we address the following questions:

Q1 description
Q2 description
Q3 description

5 Conclusion

Write conclusion.

References

[1] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.