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, [15], [25], [8] [10] [12], paper by Bistra Dilkina:
- Papers on SAT and CSP solving (Hsu, LeSong, Aachen, Loukas ...) [22]

GNN: Recently, graph neural networks (GNNs) [9, 21] emerged as a flexible framework for machine learning on graphs and relational data. Notable instances of this architecture include, e.g., [7, 11, 24], and the spectral approaches proposed in, e.g., [3, 6, 13, 19]—all of which descend from early work in [14, 18, 23, 21]. A survey of recent advancements in GNN techniques can be found, e.g., in [4, 26, 28]. The limits of GNNs have been studied in [2, 5, 16, 17, 20, 27].

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] [CM: Background on bb, branching etc]

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

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

where $f_{\mathrm{aggr}}^{\mathbf{W_1}}$ aggregates over the set of neighborhood features and $f_{\mathrm{merge}}^{\mathbf{W_2}}$ merges the vertex's representations from step $(t-1)$ with the computed neighborhood features. Both $f_{\mathrm{aggr}}^{\mathbf{W_1}}$ and $f_{\mathrm{merge}}^{\mathbf{W_2}}$ 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.]

[**CM:** exploration/exploitation η] Same algorithm as above interpreted as a message passing algorithm:

Algorithm 1 MWU (Message passing version) for the LP feasibility problem.

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

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

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

    Perform gradient descent by $\mathbf{w}_i \leftarrow (1 - \eta \mathbf{e}_i)\mathbf{w}_i$
    Normalize weights $\mathbf{p}_j \leftarrow \mathbf{w}_j / \boldsymbol{\Gamma}(t)$ for i in $[m]$, where $\boldsymbol{\Gamma}(t) = \sum_{i \in [m]} \mathbf{w}_i$
    Update solution $\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} + \mathbf{x}$
end for
return Average over solution $\bar{\mathbf{x}}/T$

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 2 Data-driven MWU for variable bias prediction.

Input: Training set $\mathcal{S}$, $\varepsilon > 0$, number of epochs E, 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 $\mathbf{v}_i^{(0)}$, $\mathbf{c}_i^{(0)}$
Initialize parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\mathrm{V}}, \boldsymbol{\theta}_{\mathrm{C}}, \boldsymbol{\theta}_{\mathrm{X}})$
Set weights $\mathbf{w}_j^{(0)} \leftarrow 1$, $\mathbf{p}_j^{(0)} \leftarrow 1/m$ for j in $[m]$
for e in $[E]$ **do**
    Sample $I = (\mathbf{c}, \mathbf{A}, \mathbf{b})$ with variable bias $\bar{\mathbf{b}}$ uniformly and at random from $\mathcal{S}$, inducing the bipartite graph $B(I)$
    for t in $[T]$ **do**
        $\mathbf{v}_i^{(t)} \leftarrow f_{\mathrm{V}}^{(t)}(\{\!\{(\mathbf{v}_i^{(t-1)}, \mathbf{c}_j^{(t-1)}, \mathbf{p}_j^{(t-1)}, \mathbf{A}_{ji}) \colon c_j \in N(v_i)\}\!\})$ for all i in $[n]$
        $\mathbf{x}_i^{(t)} \leftarrow f_{\mathrm{X}}^{(t)}\left(\mathbf{v}_i^{(t-1)}\right)$ for all i in $[n]$
        $\mathbf{c}_j^{(t)} \leftarrow f_{\mathrm{C}}^{(t)}\left(\left\{\!\!\left\{\left(\mathbf{c}_j^{(t-1)}, \mathbf{v}_i^{(t)}, \mathbf{x}_i^{(t)}, \mathbf{A}_{ji}, \mathbf{b}_j\right) \colon i \in N(c_j)\right\}\!\!\right\}\right)$ for all j in $[m]$
        $\mathbf{e}_j^{(t)} \leftarrow 1/\rho \left(\sum_{i \in N(j)} \mathbf{A}_{ji} \mathbf{x}_i\right) - \mathbf{b}_j$
        $\mathbf{w}_j^{(t)} \leftarrow (1 - \eta \mathbf{e}_j^{(t)}) \mathbf{w}_j^{(t-1)}$
        $\mathbf{p}_j \leftarrow w_j^{(t)} / \boldsymbol{\Gamma}(t)$, where $\boldsymbol{\Gamma}(t) = \sum_{j \in [m]} \mathbf{w}_j^{(t)}$
        Update solution $\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} + \mathbf{x}$
    end for
    $\bar{\mathbf{x}} \leftarrow 1/T \sum_{t \in [T]} \mathbf{x}(t)$
    $L \leftarrow L + \ell(\bar{\mathbf{x}}, \bar{\mathbf{b}})$
    Gradient descent on $1/T L$ with regard to $\boldsymbol{\theta}$
end for
return $\boldsymbol{\theta}$

[**CM:** overloading of c] [**CM:** how to incoperate data information]

Algorithm 3 MIP-GNN for variable bias prediction.

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

Initialize $\mathbf{v}^{(0)}$ and $\mathbf{c}^{(0)}$
Initialize $\mathbf{p}_i^{(0)} \leftarrow 1/m$
for e in $[E]$ **do**
  Sample $I = (\mathbf{c}, \mathbf{A}, \mathbf{b})$ from $U(S)$
  for t in $[T]$ **do**
    $\mathbf{V}_j^{(t)} \leftarrow f_{\mathrm{V}}^{(t)}(\{\!\{(\mathbf{V}_j^{(t-1)}, \mathbf{C}_j^{(t-1)}, \mathbf{p}_j^{(t-1)}, \mathbf{A}_{ij}) \mid c_i \in N(v_j)\}\!\})$ for all j in $[n]$
    $\mathbf{X}_j^{(t)} \leftarrow f_{\mathrm{X}}^{(t)}(\mathbf{V}_i^{(t-1)})$ for all j in $[n]$

    $\mathbf{C}_j^{(t)} \leftarrow f_{\mathrm{C}}^{(t)}(\{\!\{(\mathbf{C}_j^{(t-1)}, \mathbf{V}_i^{(t)}, \mathbf{X}_i^{(t)}, \mathbf{A}_{ij}, \mathbf{b}_j) \mid v_i \in N(c_j)\}\!\})$ for all i in $[m]$
    $\mathbf{p}_j^{(t)} \leftarrow \mathrm{softmax}(f_{\mathrm{P}}(\mathbf{e}^{(t)}))_j$
  end for
  $\bar{\mathbf{x}} \leftarrow f(\|_{t\in[T]}\mathbf{x}^{(t)})$
  $L \leftarrow L + \ell(\bar{\mathbf{x}}, \bar{\mathbf{b}})$
  Gradient descent on L with regard to $\boldsymbol{\theta}$
end for
return $\boldsymbol{\theta}$

Initialize parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\mathrm{V}}, \boldsymbol{\theta}_{\mathrm{C}}, \boldsymbol{\theta}_{\mathrm{X}}, \boldsymbol{\theta}_{\mathrm{P}})$

3.1 Connnection to boosting

joint MWU and boosting

3.2 Unsupervised setting

random matrices, recover MWU, also test in experiments

3.3 Discussion: Road blocks and the road ahead

discuss limitation of GNNs approaches (bipartite GNN), 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.

[2] P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. L. Reutter, and J. Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2020.

[3] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and deep locally connected networks on graphs. In *International Conference on Learning Representation*, 2014.

[4] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *CoRR*, abs/2005.03675, 2020.

[5] Z. Chen, S. Villar, L. Chen, and J. Bruna. On the equivalence between graph isomorphism testing and function approximation with GNNs. In *Advances in Neural Information Processing Systems*, pages 15868–15876, 2019.

[6] M. Defferrard, Bresson X., and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[7] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, pages 2224–2232, 2015.

[8] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 15554–15566, 2019.

[9] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

[10] P. Gupta, M. Gasse, E. B. Khalil, M. P. Kumar, A. Lodi, and Y. Bengio. Hybrid models for learning to branch. *CoRR*, abs/2006.15212, 2020. URL `https://arxiv.org/abs/2006.15212`.

[11] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1025–1035, 2017.

[12] E. B. Khalil, P. Le Bodic, L. Song, G. L. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming. In *AAAI Conference on Artificial Intelligence*, pages 724–731, 2016.

[13] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representation*, 2017.

[14] D. B. Kireev. Chemnet: A novel neural network based method for graph/property mapping. *Journal of Chemical Information and Computer Sciences*, 35(2):175–180, 1995.

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

[16] T. Maehara and H. NT. A simple proof of the universality of invariant/equivariant graph neural networks. *CoRR*, abs/1910.03802, 2019.

[17] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2153–2164, 2019.

[18] C. Merkwirth and T. Lengauer. Automatic generation of complementary descriptors with molecular graph networks. *Journal of Chemical Information and Modeling*, 45(5): 1159–1168, 2005.

[19] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5425–5434, 2017.

[20] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, Jan Eric Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI Conference on Artificial Intelligence*, pages 4602–4609, 2019.

[21] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[22] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations*, 2019.

[23] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(2):714–35, 1997.

[24] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[25] P. Velickovic, R. Ying, M. Padovano, R> Hadsell, and C. Blundell. Neural execution of graph algorithms. In *International Conference on Learning Representations*, 2020.

[26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019.

[27] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[28] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.