A Framework For Differentiable Discovery of Graph Algorithms

Hanjun Dai[‡], **Xinshi Chen**[◇], **Yu Li**[†], **Xin Gao**[†], **Le Song**[◇]
[‡]Google Research, hadai@google.com
[◇] Georgia Institute of Technology, xinshi.chen@gatech.edu, lsong@cc.gatech.edu
[†] King Abdullah University of Science and Technology, {yu.li, xin.gao}@kaust.edu.sa

Abstract

Recently there is a surge of interests in using graph neural networks (GNNs) to learn algorithms. However, these works focus more on imitating existing algorithms, and are limited in two important aspects: the search space for algorithms is too small and the learned GNN models are not interpretable. To address these issues, we propose a novel framework which enlarge the search space using cheap global information from tree decomposition of the graphs, and can explain the structures of the graph leading to the decision of learned algorithms. We apply our framework to three NP-complete problems on graphs and show that the framework is able to discover effective and explainable algorithms.

1 Introduction

Many graph problems such as minimum vertex cover (MVC), minimum dominant set (MDS) and maximum cut (Max-Cut) are NP-hard. The classical algorithm design paradigm often requires significant efforts from domain experts to understand and exploit problem structures, in order to come up with effective procedures. Thus there is a surge of interests in recent years to use learning and differentiable search to discover graph algorithms (see Appendix B for more related works).

In this context, GNNs have been widely used for representing and learning graph algorithms [1, 2]. However, directly using a GNN model to define the algorithm search space may not be enough for discovering an algorithm better than existing greedy ones [3, 4] as it only finds local algorithms that miss out the global information, which foundamentally restricts their expressiveness power.

Another important aspect which have largely been ignored in previous work is explaining the learned algorithm encoded in GNN. Many previous works focus on the ability of GNNs to imitate existing graph algorithms, without showing new algorithms are being learned. Therefore, there is an urgent need to develop explainable graph models to understand the learned algorithm.

In this paper, we propose a new framework for differentiable graph algorithm discovery (DAD), focusing on two important aspects of the discovery process: designing a larger search space, and an effective explainer model. More specifically, we design a search space for graph algorithms by *augmenting GNNs with cheap global graph features*. Our proposed global features are specially designed for learning graph algorithms. It reveals to be a simple yet effective way of enhancing the capacity of GNNs. After the GNN is sufficiently trained, we employ a novel *graph explainer model* to figure out the structural property of the graph leading to the algorithm decision. The graph explainer model allows us to transform the algorithm represented in continuous embedding space back to the interpretable discrete space, so that it can assist human experts in understanding the discovery.

We show that our DAD framework is able to discover graph algorithms that achieve better approximation ratios. Besides, we apply our explainer model to the learned algorithms and demonstrate some interesting and explainable patterns. Except for methodology, we also contribute in generating

1st Workshop on Learning Meets Combinatorial Algorithms @ NeurIPS 2020, Vancouver, Canada.

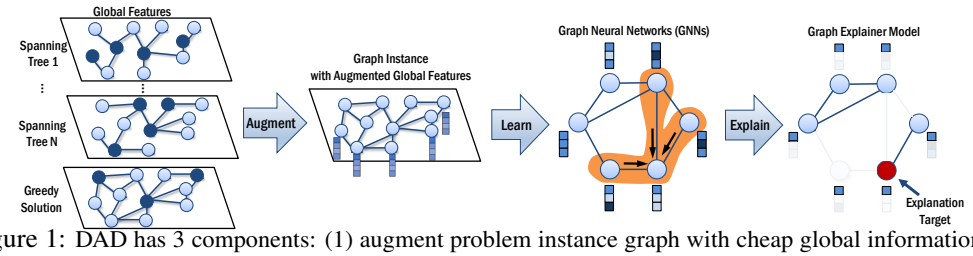


Figure 1: DAD has 3 components: (1) augment problem instance graph with cheap global information; (2) learn graph neural networks with augmented graphs; (3) explain the learned GNN with a graph explainer model.

a new dataset. We run `Gurobi` for around **12,000,000** core hours to generate a reasonably large set of (graph problem, solution) pairs, with varying graph sizes. It could be a very useful benchmark dataset for graph algorithm learning, which will help with future researches in this area.

2 Differeniable Graph Algorithm Discovery (DAD) Framework

We focus on three NP-hard problems in this paper: MVC, Max-Cut, and MDS. Let $G = (V, E, w)$ be a weighted graph, where V denotes the set of nodes, E the set of edges, and $w : E \mapsto \mathbb{R}$ the edge weight function. DAD innovates on three important aspects of the graph algorithm discovery process.

Search space design. we make use of cheap global information which can be obtained in sub-quadratic time in the number of nodes and edges of the graph. In particular, we will find spanning trees of the original graph, and use the solutions on these trees as augmented node/edge features.

Learning method. We can use either supervised or unsupervised approaches to train the GNNs.

Explainer model. We design a novel graph explainer model that uses subgraph pattern selection to find the most influential subgraph patterns in an information theoretic sense. This is trained by differentiating through a blackbox combinatorial solver.

2.1 Cheap Solutions As Global Features

Inspired by Kelner et al. [5], which speeds up the linear system solvers by 1) solving problem on the low-stretch spanning tree $T \subseteq G$; 2) refine the initial solution on the original graph G, we generalize the idea to solve a broader class of graph problems. In many graph problems, global optimal solutions can be found efficiently by algorithms such as dynamic programming if the graph is a tree. Therefore, we propose to use **(i) solutions on spanning trees** T and **(ii) approximate solutions of greedy algorithms on** G as cheap global features to augment the original graphs.

(i) Given a graph G, we first find its spanning tree $T \subseteq G$. There are multiple ways of constructing the trees, and we can obtain n spanning trees $T(1), \cdots, T(n)$ for each graph, using the same set of n algorithms. On each tree $T(k)$, we can apply dynamic programming (DP) based algorithms to quickly find an optimal solution on the tree, denoted by a binary vector $\boldsymbol{y}^{T(k)} \in \{0,1\}^{|V|}$. The tree solutions, together with the trees themselves, can be used as global features for the graph G.

$$T(1), \cdots, T(n) \xmapsto[\text{dynamic programming}]{\text{global optimal on spanning trees}} \{(T(1), \boldsymbol{y}^{T(1)}), \cdots, (T(n), \boldsymbol{y}^{T(n)})\} \tag{1}$$

(ii) We can also operate on the original graph G to quickly obtain an approximate solution by using some greedy algorithms. With a set of m greedy algorithms denoted by $\texttt{Gd}(1), \cdots, \texttt{Gd}(m)$, we can obtain a set of approximate solutions.

$$\texttt{Gd}(1), \cdots, \texttt{Gd}(m) \xmapsto[\text{different greedy algorithms}]{\text{approximate solution}} \{\boldsymbol{y}^{\texttt{Gd}(1)}, \cdots, \boldsymbol{y}^{\texttt{Gd}(m)}\} \tag{2}$$

The tree solutions and greedy solutions will be concatenated and used as node features. Besides, each spanning tree can be represented by a binary matrix $\boldsymbol{T}(k) \in \{0,1\}^{|V|\times|V|}$, where the (i,j)-th entry indicates whether the edge (i,j) is in the tree. These spanning trees will also be concatenated and used as edge features. Then the overall features can be denoted by

$$\textbf{node features:} \quad \boldsymbol{X} := [\boldsymbol{y}^{T(1)}, \cdots, \boldsymbol{y}^{T(n)}, \boldsymbol{y}^{\texttt{Gd}(1)}, \cdots, \boldsymbol{y}^{\texttt{Gd}(m)}] \in \{0,1\}^{|V|\times(n+m)}, \tag{3}$$

$$\textbf{edge features:} \quad \boldsymbol{Z} := [\boldsymbol{T}(1); \cdots; \boldsymbol{T}(n)] \in \{0,1\}^{|V|\times|V|\times n}. \tag{4}$$

In the rest of this paper, we denote $\boldsymbol{X}_i$ as the i-th row vector of $\boldsymbol{X}$ which corresponds to the features of node i, and $\boldsymbol{Z}_{ij}$ as the vector $\boldsymbol{Z}[i, j, :]$ which corresponds to the features of the edge (i, j).

2.2 Learning Distributed Local Graph Algorithm

A GNN computes the node embeddings by iteratively aggregating the messages from neighbours:

$$\boldsymbol{h}_i^{(t+1)} \leftarrow \boldsymbol{F}_\theta \left(\boldsymbol{h}_i^{(t)}, \boldsymbol{X}_i, \{\boldsymbol{h}_j^{(t)}, \boldsymbol{X}_j, \boldsymbol{Z}_{ij}\}_{j \in \mathcal{N}(i)} \right), t \in \{0, 1, 2, \ldots, T-1\} \tag{5}$$

where $\boldsymbol{h}_i^{(t)}$ is the d-dimentional feature embedding for node $i \in V$ at t-th layer, $\theta \in \Theta$ are the parameters in the GNN, $\boldsymbol{F}_\theta$ is a nonlinear function, and $\mathcal{N}(i)$ denotes the neighbours of node i. To learn the parameters θ in the GNN, we consider both the supervised and unsupervised setting:

Supervised learning with ILP solvers. In the supervised setting, for each graph G, a solution $\boldsymbol{y}^*$ is given by running an expensive solver. We use *hindsight loss* [6, 2] to alleviate the mode averaging:

$$\text{Hindsight loss: } \min_{k \in \{1, \cdots, K\}} \ell(y_i^*, p_{i,k}) \quad \text{where} \quad p_{i,k} = \texttt{OutputLayer}_k(\boldsymbol{h}_i^{(T)}), \tag{6}$$

K could be the number of modes, and ℓ is the binary cross entropy loss.

Unsupervised learning with continuous relaxation. The design of the unsupervised learning mainly follows [7] and more details and principles can be found there.

$$\mathcal{L}_U(\boldsymbol{p}, G) := \mathbb{E}[f(Y; G)] + \beta \cdot \textstyle\sum_{i=1}^{l} \mathbb{P}[g_i(Y; G) \leq 0], \quad \text{where} \quad Y \sim \text{Bernoulli}(\boldsymbol{p}), \tag{7}$$

Note that it is a relaxed version of the loss proposed by Karalias and Loukas [7]. Here we treat the l constraints seperately for efficient computations. See Appendix F for more details.

2.3 Explain The Learned GNN

We simply denoted the learned GNN as F_θ. To understand what algorithm is discovered in F_θ, we propose a novel method to explain its predictions $\{p_i\}_{i \in V}$. We want to find that, for each node i, which subset of nodes $V_S^i \subseteq V$ and edges $E_S^i \subseteq E$ are most influential to the prediction p_i. Besides, the global features X and Z are induced by the solutions of n spanning trees and m greedy algorithms, which we denote as $\mathcal{S} := \{T(1), \cdots, T(n), \texttt{Gd}(1), \cdots, \texttt{Gd}(m)\}$. We are also interested in finding which trees or algorithms provide more useful features for the prediction.

Since we focus on node-level predictions, our explainer is defined as a mapping that finds node-dependent subsets

$$\text{Explainer:} \quad \mathcal{E} : (i, V, E, \mathcal{S}) \mapsto \left(V_S^i, E_S^i, \mathcal{S}_S^i\right), \tag{8}$$

where V_S^i and E_S^i are selected subsets of k_V nodes and k_E edges respectively, and $\mathcal{S}_S^i \subseteq \mathcal{S}$ is a subset of $k_\mathcal{S}$ trees/algorithms. More precisely, we parameterize $\mathcal{E}$ as a second GNN along with a top-K operation, where the GNN is applied to score the set of structural elements in $(V, E, \mathcal{S})$, and the top-K operation selects the $(k_V, k_E, k_\mathcal{S})$ highest scoring elements. We here provide our optimization technique below. See Appendix A for more details on the parameterization.

Variational lower bound of mutual information. To learn the explainer $\mathcal{E}_\phi$, we employ the learning to explain (L2X) framework in [8]. In L2X, $\mathcal{E}_\phi$ is learned by maximizing the mutual information between the predicted label Y_i and the selected subsets $(V_S^i, E_S^i, \mathcal{S}_S^i)$. However, due to intractability, a variational distribution parameterized by a neural network Q_ψ will be learned jointly with the explainer $\mathcal{E}_\phi$ to maximize the variational lower bound of the mutual information

$$\max_{\phi, \psi} \ \mathbb{E}\left[\log Q_\psi\left(Y_i \mid V_S^i, E_S^i, \mathcal{S}_S^i\right)\right], \ \text{ s.t. } (V_S^i, E_S^i, \mathcal{S}_S^i) = \mathcal{E}_\phi(i, V, E, X, Z), \tag{9}$$

where the expectation is taken over the predictive probability p_i given by the learned GNN, over all node i in graph G and all graph G in the training set $\mathcal{G}_{train}$.

Differentiable top-K. The additional technical challenges for optimizing the variational objective in Eq. 9 is that the top-K operation in the parameterized explainer (Eq. 12) is not differentiable. More specifically, a top-K operation over the scores $c_1, \cdots, c_M$ for some M is equivalent to solving the following integer linear programming (ILP) problem:

$$\arg\max_{\boldsymbol{x} \in \{0,1\}^M} \textstyle\sum_{j=1}^{M} c_j x_j \quad \text{s.t. } \textstyle\sum_{j=1}^{M} x_j = K. \tag{10}$$

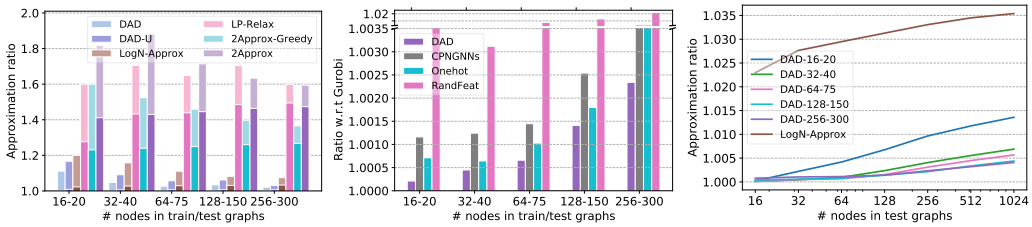
It is easy to see that the optimal solution $x_j^* = 1$ only if c_j is among the top-K scores. Given this equivalence, we can employ the blackbox differentiation techniques introduced by Vlastelica et al. [9] to compute the gradient of the ILP solver, and optimize the parameters ϕ in the explainer.

Our method is closely related but different from GNNExplainer [10]. See Appendix C for details.

3 Experiment

Below we present both numerical and explanation results. Please refer to Appendix D, H, J for details on experiment setup and complete set of experimental results.

3.1 Quantitative results



(a) Classical Algorithms (b) Global GNNs (c) Extrapolation

Figure 2: Test approximation ratio comparison on MVC Barabasi Albert graphs.

Synthetic graphs We first conduct experiments on MVC Barabasi Albert (BA) graphs. We can see in Figure 2(a) DAD leverages but goes beyond the cheap algorithms, with both supervised and unsupervised (DAD-U) learning strategies. The gain is significant regarding both the average (dark color) and maximum (light color) test ratio. Figure 2(b) compares against other GNNs with global information, where our design consistently outperforms alternatives. We also test the extrapolation ability of DAD, where we train the models on graphs up to 300 nodes, and evaluate them up to 1100 nodes. Figure 2(c) shows consistently better results than the best classical baseline algorithm.

Table 1: Real world MVC and MDS test results.

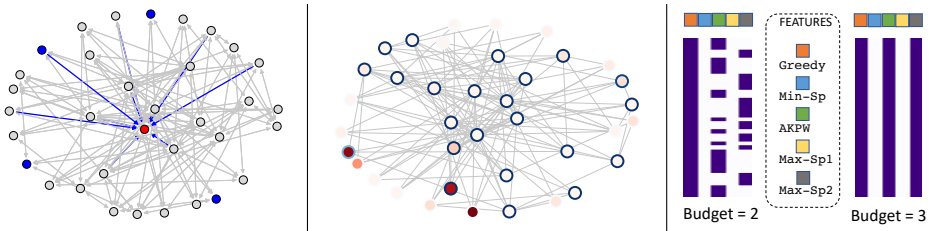
Data	MVC		MDS	
	2nd-best baseline	DAD	2nd-best baseline	DAD
Twitter	$1.68e^{-2}$	$\mathbf{1.93e^{-3}}$	$6.43e^{-2}$	$\mathbf{8.42e^{-3}}$
Github	$1.64e^{-3}$	$\mathbf{1.09e^{-5}}$	$3.89e^{-4}$	$\mathbf{4.14e^{-5}}$
Memetracker	$1.81e^{-2}$	$\mathbf{1.55e^{-5}}$	$1.30e^{-4}$	$\mathbf{7.67e^{-5}}$

Table 2: Real world MAXCUT.

	Optsicom-B	Optsicom-G
2Approx	$7.65e^{-2}$	$7.98e^{-2}$
MaxSpanning	$7.63e^{-2}$	$7.09e^{-2}$
SDP	$8.36e^{-1}$	$8.52e^{-1}$
DAD	$\mathbf{4.23e^{-3}}$	$\mathbf{1.16e^{-5}}$

Real-world graphs We run MVC and MDS tasks on the social networks, and run MAXCUT on the physics(`Optsicom`) graphs (see Appendix I). Table 1 and Table 2 show that the relative error of DAD can be several magnitudes smaller than the second best classical algorithms in some cases.

3.2 Explaining learned algorithms



(a) Greedy-like behavior (b) Anchor-node behavior (c) Feature selection

Figure 3: Explaining the learned MVC (a, b) and Max-Cut (c) algorithms.

In this section, we use our proposed explainer to explain the learned algorithm on BA graphs.

Explanation with edge selection: Here we limit the number of nodes to be 5, and edges to be 10 when explaining each target node of a graph. We found in Figure 3(a) that some nodes depict the greedy behavior, where the budget for edge explanations is mostly allocated to adjacent edges.

Explanation with node selection: We try to understand the correlation between selected explanation nodes and the node predictive probabilities in figure 3(b). We can see the existence of *anchor nodes* which are commonly selected for explanation (red colors). The anchors tend to distribute evenly among both high probability (with dark blue boundary) and low probability nodes. Such diversity driven anchoring behavior is quite similar to landmarks in kernel methods.

Explanation with feature selection: We further look at the contribution of different cheap global features to the GNN prediction for Max-Cut problem in Figure 3(c). We learn two principles in instantiating our framework: 1) better algorithm solution generally introduces better global information; 2) having diverse algorithmic solutions is better than multiple similar ones.

References

[1] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song. Learning steady-states of iterative algorithms over graphs. In *International conference on machine learning*, pages 1106–1114, 2018.

[2] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pages 539–548, 2018.

[3] Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Computing*, 28(1):31–53, 2015.

[4] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems. In *Advances in Neural Information Processing Systems*, pages 4081–4090, 2019.

[5] Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 911–920, 2013.

[6] Abner Guzman-Rivera, Dhruv Batra, and Pushmeet Kohli. Multiple choice learning: Learning to produce multiple structured outputs. In *Advances in Neural Information Processing Systems*, pages 1799–1807, 2012.

[7] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *arXiv preprint arXiv:2006.10643*, 2020.

[8] Jianbo Chen, Le Song, Martin J Wainwright, and Michael I Jordan. Learning to explain: An information-theoretic perspective on model interpretation. *arXiv preprint arXiv:1802.07814*, 2018.

[9] Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*, 2019.

[10] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In *Advances in neural information processing systems*, pages 9244–9255, 2019.

[11] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[12] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2156–2167, 2019.

[13] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

[14] Pablo Barceló, Egor V Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2019.

[15] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019.

[16] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. *arXiv preprint arXiv:1910.10593*, 2019.

[17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[18] Younjoo Seo, Andreas Loukas, and Nathanaël Perraudin. Discriminative structural graph classification. *arXiv preprint arXiv:1905.13422*, 2019.

[19] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks. *arXiv preprint arXiv:2002.03155*, 2020.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[21] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[22] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[23] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[24] Pieter-Jan Kindermans, Kristof Schütt, Klaus-Robert Müller, and Sven Dähne. Investigating the influence of noise and distractors on the interpretation of neural networks. *arXiv preprint arXiv:1611.07270*, 2016.

[25] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[26] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. pages 4768–4777, 2017.

[27] Noga Alon, Richard M Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995.

[28] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL www.graphlearning.io.

[29] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

Appendix

A Parameterization of the explainer model

Parameterization of $\mathcal{E}$. It is notable the prediction p_i is computed by F_θ based on the T-hop subgraph of the node i and is irrelavent to nodes of distrance larger than T. Therefore, when working on node i, the explainer only needs to select the influential nodes/edges from its T-hop subgraph, denoted by $G(i,T) := (V(i,T), E(i,T), w)$. Therefore, we define explainer GNN as

$$\text{Explainer GNN:} \quad F_\phi^{\mathcal{E}} : (V(i,T), E(i,T), X(i,T), Z(i,T)) \mapsto \{\boldsymbol{h}_j^{\mathcal{E}}\}_{j\in V(i,T)}, \tag{11}$$

where $X(i,T) := [X_j]_{j\in V(i,T)}$ and $Z(i,T) := [Z_{jk}]_{(j,k)\in E(i,T)}$ are features induced by the subgraph. Then multi-layer perceptrons (MLP) are applied to define the scoring functions:

(I) Node scores: $c_j = \texttt{MLP}_\phi^V(\boldsymbol{h}_j^{\mathcal{E}})$ for each node $j \in V(i,T)$.
(II) Edge scores: $c_{j,k} = \texttt{MLP}_\phi^E([\boldsymbol{h}_j^{\mathcal{E}}, \boldsymbol{h}_k^{\mathcal{E}}, E_{j,k}])$ for each edge $(j,k) \in E(i,T)$.
(III) Feature scores: $\boldsymbol{c} = \texttt{MLP}_\phi^{\mathcal{S}}\left(\texttt{Pooling}\left(\{\boldsymbol{h}_j^{\mathcal{E}}\}_{j\in V(i,T)}\right)\right)$, where the output $\boldsymbol{c}$ is a $(n+m)$-dimensional vector that gives the score for features induced by the n spanning trees and the m greedy algorithms.

With the scores, the subsets $(V_S^i, E_S^i, \mathcal{S}_S^i)$ can be selected as choosing the elements with scores in top-K, for $K = k_V, k_E$, and $k_{\mathcal{S}}$, respectively. In summary, the explainer model can be written as the composition of three operations

$$\text{Parameterized Explainer:} \quad \mathcal{E}_\phi = \text{top-}K \circ \texttt{MLP}_\phi \circ F_\phi^{\mathcal{E}}. \tag{12}$$

Parameterization of Q. Again we will parametrize the variational distribution Q_ψ by a third GNN $F_\psi^Q : (V_S^i, E_S^i, X_S^i, Z_S^i) \mapsto \{\boldsymbol{h}_j^Q\}_{j\in V(i,T)}$ followed by pooling and MLP:

$$\text{Variational Distribution:} \quad Q_\psi = \texttt{MLP}_\psi\left(\texttt{Pool}\left(F_\psi^Q(i, V_S^i, E_S^i, X_S^i, Z_S^i)\right)\right), \tag{13}$$

where the features $(X_S^i, Z_S^i) = H(V_S^i, E_S^i, \mathcal{S}_S^i, X, Z)$ are induced by the selected subsets via a differentiable mapping H.

B Related Work

Expressiveness of GNNs for representing algorithms. GNNs have been used to learn graph algorithms. Some recent works have discussed the representation power of GNNs for representing graph algorithms. Xu et al. [11] showed that GNNs are at most as powerful as the Weisfeiler-Lehman (WL) test in distinguishing graph structures. Maron et al. [12, 13] proposed k-GNN which can represent the results of k-WL test. Barceló et al. [14] showed that GNNs are too weak to represent logical Boolean classifiers - FOC_2, and proposed to add a global read-out functionto break this limitation. Focusing on combinatorial problems including MDS and MVC, Sato et al. [4] derived the approximation ratios of algorithms that GNNs can learn, and theorectically showed that adding the information of port numbering and weak-coloring can improve those approximation ratios.

Learning based graph algorithms. Empirically, several works have demonstrated the ability of GNNs for imitating existing graph algorithms. For instance, GNN is used to learn the convergent states of graph algorithms [1], imitate DP algorithms [15], and imitate individual steps and intermediate outputs of classical graph algorithms [16]. To learn a new and more effective graph algorithm, Khalil et al. [17] combined reinforcement learning with GNNs to learn greedy policies that can solve graph problems such as MVC and Max-cut. Karalias and Loukas [7] proposed an unsupervised framework for learning graph algorithms and demonstrated the results on maximum clique and local partitioning problems.

C Compare with GNNExplainer

Discussion. Our explainer model is very different from GNNExplainer [10], which may be of independent interests in term of explainer model for GNNs. For instance, GNNExplainer optimizes

the selection mask (i.e., the explainer) for an individual graph, while we parameterize the explainer as another GNN which can be applied to explain the predictions on a class of different graphs. Besides, the variational Q in GNNExplainer is defined to be the origianl GNN F_θ which is fixed, while we optimize Q over a parameteric family, so that our method is more principled and optimal. Furthermore, we formulate the top-K selection as a solution to an ILP and apply a blackbox differentiation techique to optimze the explainer model, which has not been attempted in the space of explainer models before. More discussions can be found in Appendix E.

D Experiment setup

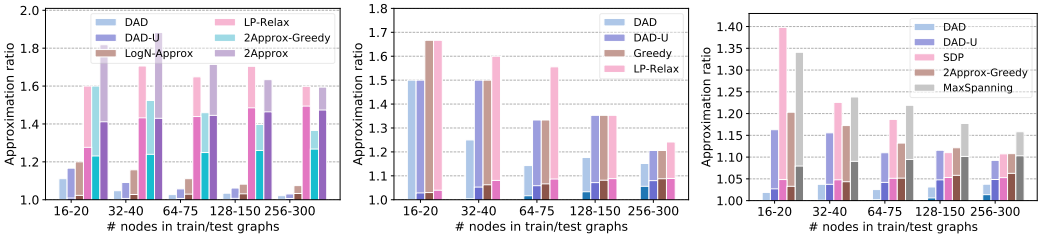
We leave the details of greedy algorithms and dynamic programming on trees in Appendix G.
We first compare against some of the cheap algorithms that are also served as features. As we use the continuous relaxation in Eq. 7, we also compare with the linear/semi-definite programming (LP/SDP) with randomized rounding for these problems:

- MVC: `LogN-Approx`(node-degree greedy), `2Approx-Greedy`(edge selection with degree greedy), `2Approx`(edge selection) and LP relaxation are included.
- MDS: `Greedy`(sequential coverage based greedy selection) and LP relaxation are included.
- Max-CUT: `2Approx-Greedy`(greedy local adjustment), SDP and `MaxSpanning`(solution directly obtained on max spanning tree, as any node assignment is a valid cut) are included.

We here include several recent works that extends GNNs with different forms of global information:

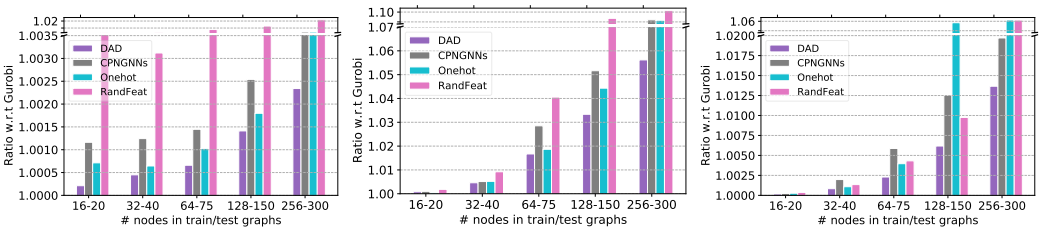
- `Onehot` [7, 18]: adds one-hot encoding to a random node.
- `RandFeat` [19]: uses random node features to break the tie.
- `CPNGNNs` [4]: leverages port numbering in GNN's message passing, together with weak 2-coloring node features. As CPNGNNs only works for graphs with bounded degree, we use the position encoding [20] for port number representation instead.

Synthetic data prepration We use two random graph models, namely Erdos Renyi (ER) with edge probability 0.15, and Barabasi Albert (BA) with edge attachment 4 to generate random graphs for training and test. We vary the graph sizes in $\{$16-20, 32-40, 64-75, 128-150, 256-300, 512-600, 1024-1100$\}$. In each $\{$problem, graph type, graph size$\}$ configuration, we generation $\{10k, 1k, 1k\}$ graph instances for training, validation and test, respectively. We run `Gurobi` on each instance to get the solution for evaluation or supervised training. See Appendix H for more details on setup and `Gurobi`.



MVC-Barabasi Albert MDS-Barabasi Albert Maxcut-Barabasi Albert

Figure 4: Test approximation ratio for different algorithms. Foreground color (darker) shows the average test ratio, while background color (lighter) shows the maximum test ratio.



MVC-Barabasi Albert MDS-Barabasi Albert Maxcut-Barabasi Albert

Figure 5: Test approximation ratio for GNNs with other types of global information.

E Other related works

Explainer models. Existing work on explainer models approach the problems from three directions. The first line of work use gradients of the outputs with respect to inputs to identify the salient features in the inputs [21, 22]; The second approximates the model with simple interpretable models, such as locally additive models [23–26]; the third defines input pattern selection operators, such that the outputs of the model based on the selected input patterns has large mutual informaton with the original model outputs [8, 10]. The difference of our explainer model compared to GNNExplainer [10] is discussed in Sec C.

F Probabilisitc relaxation of constraints

In main paper we mentioned that we are solving the following probabilistic relaxation of the integer programming:

$$\mathcal{L}_U(\boldsymbol{p}, G) := \mathbb{E}[f(Y; G)] + \beta \cdot \sum_{i=1}^{l} \mathbb{P}[g_i(Y; G) \leq 0], \quad \text{where} \quad Y \sim \text{Bernoulli}(\boldsymbol{p}),$$

This can be more general for many constrained optimization problems. For example:

MVC: MVC constraint requires that each edge $e = (i, j)$ has $p_i + p_j \geq 1$. In the probabilistic sense, the probability of violating this constraint is $(1 - p_i) * (1 - p_j)$. Thus the expected penalty would be:

$$\beta \sum_{e=(x,y)\in G} (1 - p_x) * (1 - p_y) \tag{14}$$

MDS: MDS constraints are placed on nodes. The ILP constraint says $p_i + \sum_{j\in\mathcal{N}(i)} p_j \geq 1$. Again, we write it in the probability sense, with the expected penalty be:

$$\beta \sum_{i\in V} (1 - p_i) * \prod_{j\in\mathcal{N}(i)} (1 - p_j) \tag{15}$$

In our experiment, we use above expected penalty to handle the hard constraints in unsupervised learning setting.

G Details of cheap algorithmic features

The cheap algorithm configurations are described in Table 3. For unweighted graphs the spanning

Table 3: Cheap algorithmic configurations for DAD on different problems.

	Tree solutions	Greedy Algorithms
MVC	DP on random spanning tree	Node/Edge based greedy
MDS	DP on random spanning tree	Node based greedy
Max-Cut	Two-coloring on {max,min,low-stretch} spanning tree	Greedy local adjustment

trees are usually not unique, so we can use multiple tree solutions as global features for DAD. We use AKPW [27] for computing the low-stretch spanning tree for Max-Cut problems.

G.1 Obtaining tree solutions

Note that although these combinatorial optimization on trees are easier than NP-hard, but they are still nontrivial. Here we cover the recursion of dynamic programming for these problems:

MVC: Firstly we pick a random node as root, and traverse the tree in DFS order. Suppose we use $f[n, 0]$ to denote the cover size of the subtree rooted at node n, when the node n is not chosen, and

$f[n,1]$ to denote the cover size when node n is chosen, then we have:

$$\begin{aligned} f[n,0] &= \sum_{c\in\text{child(n)}} f[c,1] \\ f[n,1] &= 1+\sum_{c\in\text{child(n)}} \min\{f[c,0],f[c,1]\} \end{aligned} \tag{16}$$

And the final cover size is obtained by $\min\{f[\text{root},0],f[\text{root},1]\}$.

MDS: Firstly we pick a random node as root, and traverse the tree in DFS order. Suppose we use $f[n,0]$ to denote the dominant set size of the subtree rooted at node n, when the node n is chosen; use $f[n,1]$ to denote the situation when n is not picked, but it has been dominated by other nodes; use $f[n,2]$ to denote that we don't care about node n's situation, but its subtree should satisfy the constraint. Then we have:

$$\begin{aligned} f[n,0] &= 1+\sum_{c\in\text{child(n)}} \min\{f[c,0],f[c,1],f[c,2]\} \\ f[n,2] &= \sum_{c\in\text{child(n)}} \min\{f[c,0],f[c,1]\} \\ f[n,1] &= \begin{cases} \infty, \text{child(n)}=\emptyset \\ f[n,2]+\min_{c\in\text{child(n)}}\left[f[c,0]-\min\{f[c,0],f[c,1]\}\right], \text{ otherwise} \end{cases} \end{aligned} \tag{17}$$

and finally the dominant set size equals to $\min\{f[\text{root},0],f[\text{root},1]\}$.

Max-Cut: Max-Cut on tree is easy when all the edge weights are non-negative, as every tree is a bipartite graph. We can perform two-coloring on this tree to obtain the solution. By flipping the colors afterwards, we can get another equivalent solution. Although this solution is simple, it still has nontrivial effect on obtaining the global information, as the 2-coloring solution on tree is the weak 2-coloring on the original graph. It is known that for some problems, such information can help boost the approximation ratio of distributed local algorithms [4].

H Experimental details

We run all the experiments on a cluster with 6149 nodes. In each node, there are 64 cores. To obtain the solution for evaluation or supervised training, for each graph per task, we run `Gurobi` on one node with the time limit as 1 hour. As we have three tasks, two graph models, seven graph sizes, and $12k$ graph instances in each category, the computational budget is 32 million core hours. In reality, we used around 12 million core hours because only the larger graph tasks can run up to 1 hour.

Compared to that of the exact solution, the core hour consumption of the relaxed LP is neglectable. However, the SDP for Max-CUT is computationally intensive. We used `cvxpy` to realize it. For the larger graphs, it can take up to 40 mins to resolve one graph with 64 cores. It takes around 0.2 million core hours to obtain the SDP solutions.

We tuned the hyperparameters of GNN models for each graph category, such as the number of layers and the number of spanning trees, using grid search. We run each job up to 24 hours on a 64-core node. It takes around 5 million core hours to run the jobs on the synthetic datasets and 0.3 million core hours for the real datasets.

In total, it takes us around 17.5 million core hours to run all the jobs.

I More real world data information

We collect a set of social network dataset Collab, Twitter, Github and Facebook from Morris et al. [28], following Karalias and Loukas [7]. Each dataset comes with a set of graphs, and we split them into training/validation/test with ratio 6:2:2. Additionally, we collect Memetracker graph from SNAP [29] and Optsicom used in Khalil et al. [17]. As these only have a few graphs, we use the same strategy from Khalil et al. [17] to generate graphs. Specifically, for Memetracker, we use

	Collab	Twitter	Github	Facebook	Memetracker	Optsicom
Average # nodes	74.49	131.76	83.83	81.66	315.36	125.0
Average # edges	2457.22	1709.32	117.32	80.58	335.68	375.0
# train	3,000	583	7,635	597	10,000	10,000
# valid	1,000	194	2,545	199	1,000	1,000
# test	1,000	196	2,545	199	1,000	1,000

Table 4: Real world data statistics.

Table 5: Real world MVC and MDS.

Data	MVC					MDS		
	LogN-Approx	2Approx-Greedy	2Approx	LP-Relax	DAD	Greedy	LP-Relax	DAD
Collab	$1.22e^{-3}$	$8.17e^{-2}$	$1.06e^{-1}$	$1.21e^{-1}$	$\mathbf{1.28e^{-4}}$	0	0	0
Facebook	0	0	0	0	0	$5.32e^{-3}$	**0**	**0**
Twitter	$1.68e^{-2}$	$1.62e^{-1}$	$2.88e^{-1}$	$3.84e^{-1}$	$\mathbf{1.93e^{-3}}$	$8.26e^{-2}$	$6.43e^{-2}$	$\mathbf{8.42e^{-3}}$
Github	$5.61e^{-3}$	$5.52e^{-1}$	$8.23e^{-1}$	$1.64e^{-3}$	$\mathbf{1.09e^{-5}}$	$6.24e^{-3}$	$3.89e^{-4}$	$\mathbf{4.14e^{-5}}$
Memetracker	$1.81e^{-2}$	$4.50e^{-1}$	$7.42e^{-1}$	$6.09e^{-1}$	$\mathbf{1.55e^{-5}}$	$2.38e^{-2}$	$1.30e^{-4}$	$\mathbf{7.67e^{-5}}$

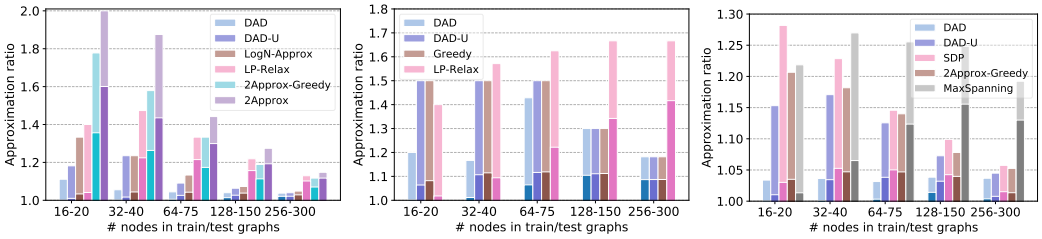
widely-adopted Independent Cascade model and sample a diffusion cascade from the full graph with constant set to 7, and consider the largest connected component in the graph as a single graph instance; for Optsicom we keep the graph structures, but perturb the edges weights using Bernoulli distribution (denoted as Optsicom-B) or Gaussian distribution with zero mean and 0.1 standard deviation (denoted as Optsicom-G).

See Table 4 for the final statistics about the datasets.

For the completeness, we additionally include MVC and MDS results on Collab and Facebook datasets in Table 5. These datasets are used in Karalias and Loukas [7] but not very suitable for evaluating MVC or MDS tasks, as it is easy to get exact solution with simple algorithms on thest two datasets.

J More experimental results

J.1 More in-distribution generalization results



MVC-Erdos Renyi MDS-Erdos Renyi Maxcut-Erdos Renyi

Figure 6: Test performance comparison with other classical algorithms on Erdos Renyi graphs.

In main paper we covered the comparison between DAD and other classical algorithms, as well as other types of global GNNs on Barabasi Albert graphs. Here we present the results on Erdos Renyi graphs, as well as the results with unsupervised learning in figure 6 and figure 7. We can see the proposed DAD consistently beats all other alternatives in all these settings.

J.2 More extrapolation results

Here we show the extrapolation results on Erdos Renyi and Barabasi Albert graphs, where DAD is trained with supervion on graphs with up to 300 nodes. We can see from figure 8, DAD still extrapolates well to large graphs.

The unsupervised learning extrapolates in a similar way, as seen in figure 9.

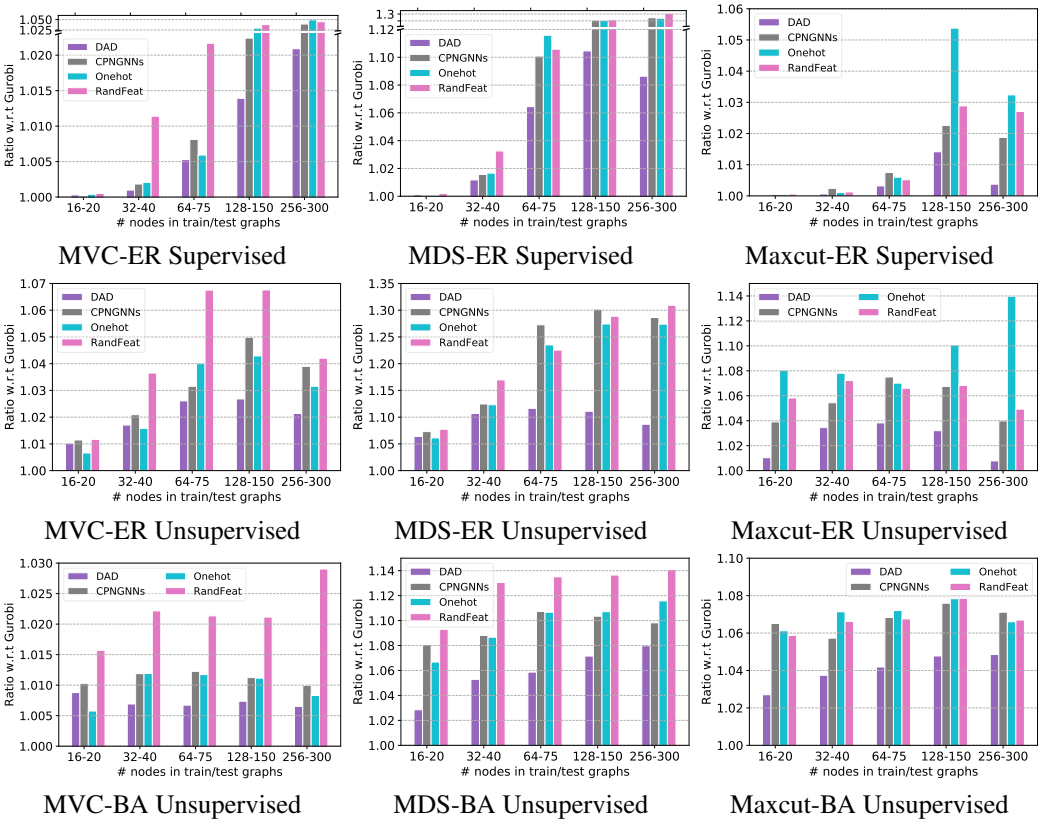


Figure 7: Test approximation ratio for GNNs with other types of global information.

One interesting finding is that, it achieves approximation ratio that is less than 1 on MDS large graphs. This indicates that the `Gurobi` is not capable of generating high quality solution on large graphs.

Also for ER graphs, it is harder to extraplate to smaller graphs. As we fix the edge probability for different sizes of the graphs, the node degree distribution would shift when graph size changes. This may prevent GNNs from extrapolating well.

J.3 More ablation results

Table 6: Ablation: Real world MVC and MDS relative error.

Data	MVC				MDS			
	Raw	Onehot	RandFeat	DAD	Raw	Onehot	RandFeat	DAD
Collab	$1.50e^{-4}$	$2.05e^{-4}$	$3.20e^{-4}$	$\mathbf{1.28e^{-4}}$	**0**	**0**	$6.00e^{-3}$	**0**
Twitter	$2.58e^{-3}$	$2.40e^{-3}$	$1.23e^{-2}$	$\mathbf{1.93e^{-3}}$	$8.67e^{-3}$	$1.19e^{-2}$	$3.37e^{-2}$	$\mathbf{8.42e^{-3}}$
Github	$3.15e^{-5}$	$2.86e^{-5}$	$1.39e^{-5}$	$\mathbf{1.09e^{-5}}$	$\mathbf{2.55e^{-6}}$	$1.22e^{-5}$	$2.25e^{-5}$	$4.14e^{-5}$
Facebook	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Memetracker	$\mathbf{7.94e^{-6}}$	$1.59e^{-5}$	$3.49e^{-4}$	$1.55e^{-5}$	$1.94e^{-5}$	$\mathbf{8.47e^{-6}}$	$5.21e^{-4}$	$7.67e^{-5}$

Table 7: Ablation: Real world MAXCUT.

	Raw	Onehot	RandFeat	DAD
Optsicom-B	$1.14e^{-2}$	$3.49e^{-5}$	$9.92e^{-3}$	$\mathbf{4.23e^{-3}}$
Optsicom-G	$1.15e^{-4}$	$8.79e^{-3}$	$2.93e^{-5}$	$\mathbf{1.16e^{-5}}$

Here we include more ablation results, on both real world datasets (see Table 7 and Table 6) and synthetic datasets (see figure 10). We can see the conclusion is still consistent with our claim in the main paper.

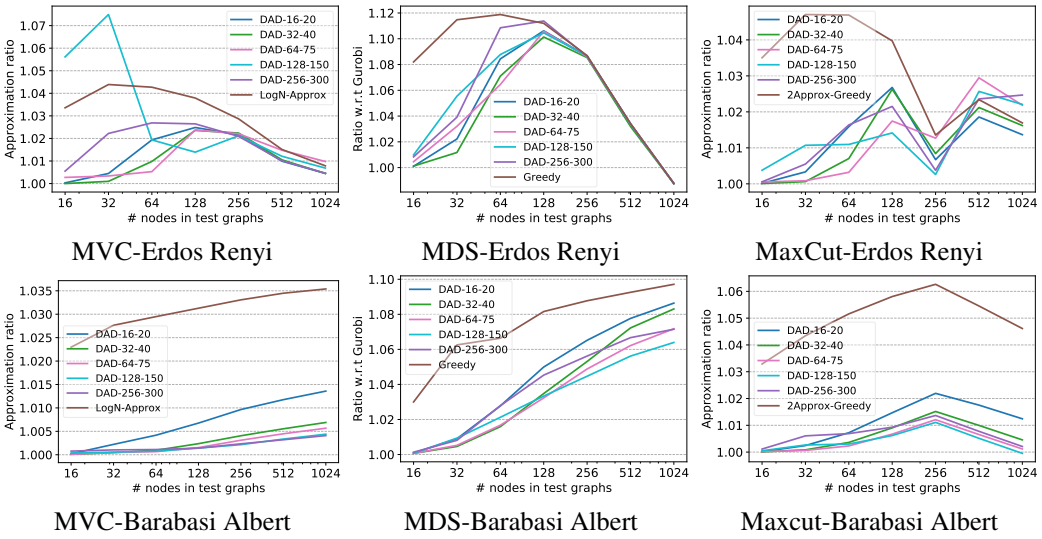


Figure 8: Extrapolation results with supervised learning, compared with the best polynomial baseline (greedy).

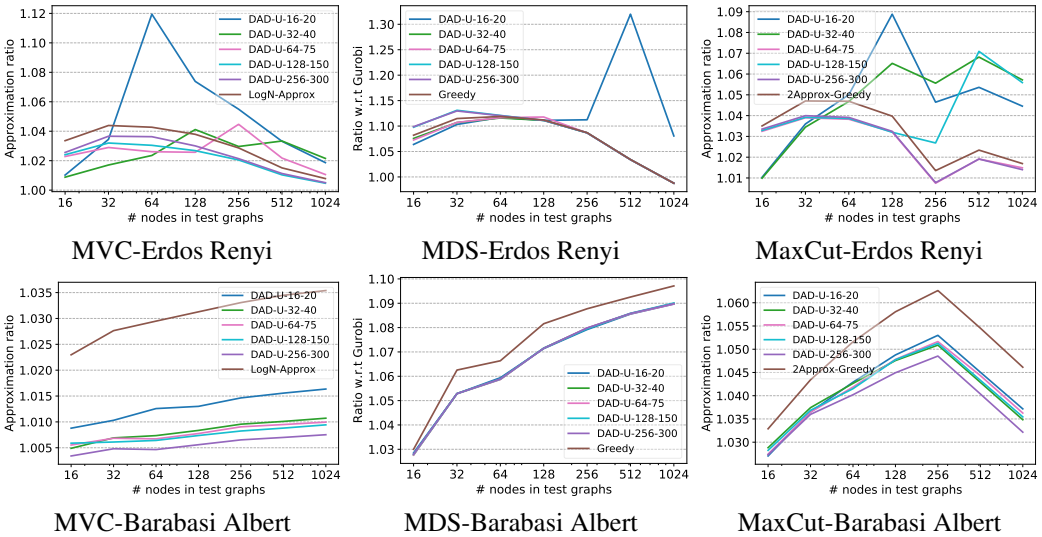


Figure 9: Extrapolation results with unsupervised learning, compared with the best polynomial baseline (greedy).

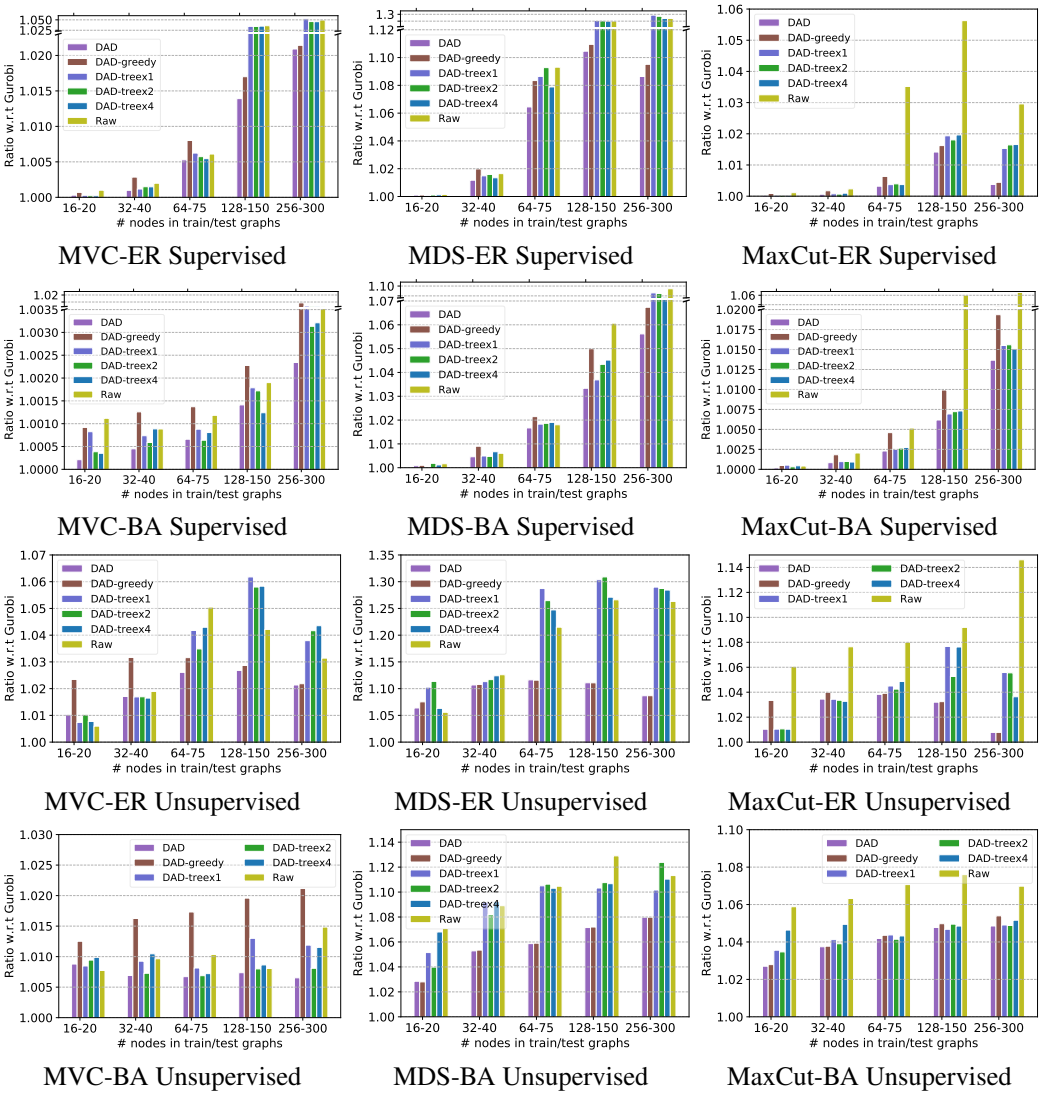


Figure 10: Ablation studies on synthetic graphs. ER stands for Erdos Renyi random graphs, and BA stands for Barabasi Albert graphs.