COMBHELPER: A Neural Approach to Reduce Search Space for Graph Combinatorial Problems

Hao Tian[1], **Sourav Medya**[2], **Wei Ye**[1,*]

[1]College of Electronic and Information Engineering, Tongji University, Shanghai, China
[2]Department of Computer Science, University of Illinois Chicago, IL, USA
{2133036, yew}@tongji.edu.cn, medya@uic.edu

Abstract

Combinatorial Optimization (CO) problems over graphs appear routinely in many applications such as in optimizing traffic, viral marketing in social networks, and matching for job allocation. Due to their combinatorial nature, these problems are often NP-hard. Existing approximation algorithms and heuristics rely on the search space to find the solutions and become time-consuming when this space is large. In this paper, we design a neural method called COMBHELPER to reduce this space and thus improve the efficiency of the traditional CO algorithms based on node selection. Specifically, it employs a Graph Neural Network (GNN) to identify promising nodes for the solution set. This pruned search space is then fed to the traditional CO algorithms. COMBHELPER also uses a Knowledge Distillation (KD) module and a problem-specific boosting module to bring further efficiency and efficacy. Our extensive experiments show that the traditional CO algorithms with COMBHELPER are at least 2 times faster than their original versions.

Introduction

CO problems over graphs arise in many applications such as social networks (Chaoji et al. 2012), transportation (James, Yu, and Gu 2019), health-care (Wilder et al. 2018), and biology (Hossain et al. 2020). Due to their combinatorial nature, these problems are often NP-hard, and thus the design of optimal polynomial-time algorithms is infeasible. Traditional CO algorithms such as approximation algorithms and heuristics (Hochba 1997; Papadimitriou and Steiglitz 1998; Vazirani 2001; Williamson and Shmoys 2011) are usually used to solve such problems in practice. Approximation algorithms take polynomial time and provide theoretical bounds on the quality of the solutions. On the other hand, the design of heuristics lacks provable guarantees and requires empirical expertise in specific CO problems. Although some of these algorithms (Andrade, Resende, and Werneck 2012; Mari 2017; Hoos and Stützle 2004) can find (nearly) optimal solutions or approximate high-quality solutions, they have one common bottleneck in terms of *efficiency*. Since the efficiency of generating solutions relies on the size of the search space, it becomes time-consuming to obtain the solution when the size of the search space is large.

*Corresponding author.

In recent years, machine learning (ML) methods, especially neural approaches, have been applied to design effective heuristics for CO problems (Khalil et al. 2017; Li, Chen, and Koltun 2018; Manchanda et al. 2020; Barrett et al. 2020). These neural approaches boost the performance of heuristics by exploiting the expressiveness of GNNs (Kipf and Welling 2016; Hamilton, Ying, and Leskovec 2017; Gilmer et al. 2020) and generate high-quality solutions in practice. However, they often consist of complex models with a large number of parameters. They predict the probability of each node being included in the final solution and add the best node (i.e., the node with the largest probability) to the current solution. Such process is repeated until the final solutions are obtained. Thus, they can still suffer from the *efficiency* issue of the search space being large.

To address the *efficiency* issue, an intuitive idea is to reduce the size of the search space for the CO problems. For example, (Grassia et al. 2019; Lauri et al. 2020; Fitzpatrick, Ajwani, and Carroll 2021) view the CO problem as a classification task. They train a linear classifier such as logistic regression (Cox 1958) to identify bad candidates (nodes or edges) unlikely to be included in the solution and prune them from the search space. In our work, we build upon such idea and propose a novel framework called COMBHELPER to address the *efficiency* issue of existing CO algorithms based on *node selection*. Instead of training a linear model for classification, we adopt a non-linear neural model, i.e., a GNN model, which fully exploits the graph structural information and has shown remarkable performance on *node classification* tasks. In particular, we train a GNN model to identify the *good candidates* (i.e., nodes) that are likely to be included in the solution and prune the *bad or unlikely ones* from the search space. Moreover, we apply two modules to further enhance the GNN model. The first one is a KD module used to compress the GNN model and reduce the inference time. The other one is a problem-specific boosting module, which is designed to improve the performance of node classification so that the search space for specific CO problems is pruned more precisely. Afterwards, we execute traditional CO algorithms on the reduced search space, which can accelerate the process of finding the solution set. We briefly summarize our contributions as follows:

- We propose a novel framework called COMBHELPER to improve the efficiency of existing graph CO algorithms based on node selection by reducing the search space with a GNN-based module.

- We adopt the KD framework and design a problem-specific boosting module for the GNN model to further improve the efficiency and efficacy of COMBHELPER.
- Extensive experiments demonstrate the efficiency and efficacy of COMBHELPER on both synthetic and real-world datasets. In particular, traditional CO algorithms with COMBHELPER are at least 2 times faster than their original versions. Our code is available at Github[1].

Related Work

Neural Approaches on CO Problems. Neural CO approaches have been applied to design more effective heuristics and have shown significant performance on CO problems (Vinyals, Fortunato, and Jaitly 2015; Bello et al. 2016; Khalil et al. 2017; Li, Chen, and Koltun 2018; Ahn, Seo, and Shin 2020). Ptr-Net (Vinyals, Fortunato, and Jaitly 2015) is an improved Recurrent Neural Network(RNN) model whose output length is not fixed. At each iteration of solution generation, the decoder uses the attention mechanisms (Bahdanau, Cho, and Bengio 2014) to calculate a probability for each element and add the best one (with the largest probability) into the solution. (Bello et al. 2016) proposes a framework to solve the Traveling Salesman Problem (TSP). It trains a Ptr-Net (Vinyals, Fortunato, and Jaitly 2015) by reinforcement learning, which is used to sequentially generate solutions. The network parameters are optimized by an actor-critic algorithm that combines two different policy gradient methods. S2V-DQN (Khalil et al. 2017) solves three CO problems: Minimum Vertex Cover (MVC), Maximum Cut (MAXCUT) and TSP. It trains a deep Q-network (DQN), which is parameterized by a GNN called structure2vec (Dai, Dai, and Song 2016), to construct solutions in a greedy manner. (Li, Chen, and Koltun 2018) combines Graph Convolutional Network (GCN) (Kipf and Welling 2016) with a tree search module to solve NP-hard problems. GCN is trained to generate multiple likelihoods for each node and a set of potential solutions is obtained via the tree search procedure. Then the best one is selected as the final solution. LwD (Ahn, Seo, and Shin 2020) proposes a deep reinforcement learning framework to solve the Maximum Independent Set (MIS) problem on large graphs. The agent iteratively makes or defers a decision (add one node into the independent set) until all the nodes are determined.

Pruning Search Space for CO Problems. To improve the efficiency of existing CO algorithms, some CO frameworks (Grassia et al. 2019; Lauri et al. 2020; Manchanda et al. 2020; Fitzpatrick, Ajwani, and Carroll 2021) prune the search space of the CO problems and just pay attention to the reduced search space. (Grassia et al. 2019; Lauri et al. 2020) solve the Maximum Clique Enumeration (MCE) problem and design a multi-stage pruning strategy. In each stage, they learn a new classifier with hand-crafted features that contain both graph-theoretic and statistical features. GCOMB (Manchanda et al. 2020) is proposed to solve budget-constrained CO problems such as Maximum Coverage Problem (MCP) and Influence Maximization (IM). GCOMB trains a GCN model (Hamilton, Ying, and Leskovec 2017) to prune bad nodes and a Q-learning network is trained on the pruned search space to predict the solution sequentially. (Fitzpatrick, Ajwani, and Carroll 2021) proposes a framework to solve the TSP, which presents the edges with three different kinds of features and trains an edge-based classifier to prune edges unlikely to be included in the optimal solution.

Knowledge Distillation on GNNs. The concept of KD is first introduced in (Hinton, Vinyals, and Dean 2015) whose goal is to transfer the knowledge from a high-capacity teacher model to a simple student model without loss of validity. Recently, KD has been widely applied on GNNs (Yang et al. 2020; Yan et al. 2020; Deng and Zhang 2021; Zhang et al. 2021; Guo et al. 2023). (Yang et al. 2020; Yan et al. 2020) adopts KD to compress GNN models while preserving the local structure information, which achieves competitive performance on node classification. GFKD (Deng and Zhang 2021) is a KD framework for graph-level tasks, which does not need observable graph data. GLNN (Zhang et al. 2021) aims to distill knowledge from GNNs to Multi-Layer Perceptrons (MLPs), which accelerates the model inference process and performs well on node-level tasks. BGNN (Guo et al. 2023) designs two novel modules to enhance GNNs: adaptive temperature and weight boosting, which shows superior performance on both node-level and graph-level tasks.

Problem Formulation

In this paper, we focus on graph CO problems that appear routinely in multiple domains such as in biology (Hossain et al. 2020) and scheduling (Bansal and Khot 2010). In particular, we apply our neural framework on two CO problems over graphs. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote an undirected graph, where $\mathcal{V}$ and $\mathcal{E}$ are the node set and the edge set respectively. The problems are as follows:

- **Minimum Vertex Cover (MVC):** Given an undirected graph $\mathcal{G}$, find a subset of nodes $\mathcal{S} \subseteq \mathcal{V}$ such that each edge $e \in \mathcal{E}$ in the graph is adjacent to at least one node in subset $\mathcal{S}$, and the size of $\mathcal{S}$ is minimized.
- **Maximum Independent Set (MIS):** Given an undirected graph $\mathcal{G}$, find a subset of nodes $\mathcal{S} \subseteq \mathcal{V}$ such that no two nodes in $\mathcal{S}$ are connected by an edge $e \in \mathcal{E}$ in the graph, and the size of $\mathcal{S}$ is maximized.

Our Objective. Given a CO problem over graphs, our main objective is to prune the search space of the problem via a neural model and perform traditional CO algorithms on the reduced search space to obtain the final solution. We showcase the performance of our method on three well-known CO algorithms: linear programming (Land and Doig 2010), greedy algorithm (Mari 2017; Khalil et al. 2017) and local search (Hoos and Stützle 2004; Andrade, Resende, and Werneck 2012). These CO algorithms are illustrated in more detail in the supplementary in (Tian, Medya, and Ye 2023).

[1]https://github.com/1041877801/COMBHelper

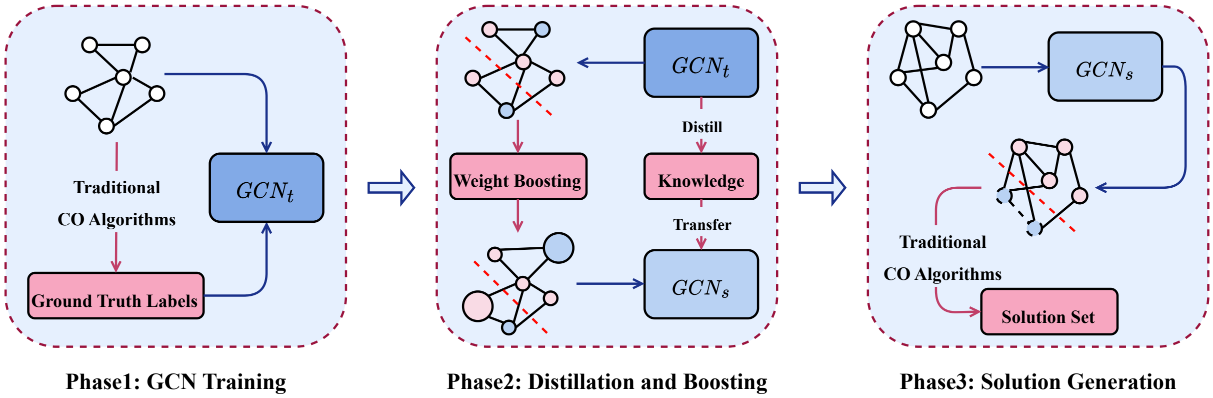


Figure 1: Overview of the proposed COMBHELPER. In phase 1, we generate the ground truth node labels with traditional CO algorithms on a small graph and train a teacher GCN (GCN_t) in a supervised manner; In phase 2, we train a student GCN (GCN_s) with two modules: (1) the KD module which transfers the knowledge from GCN_t to GCN_s and (2) the problem-specific boosting module which is used to amplify the weights of the nodes misclassified by GCN_t; In phase 3, we use GCN_s to prune the search space of traditional CO algorithms on a large graph, i.e., predict *good nodes* (nodes predicted as candidate elements in the solution). Then we perform traditional CO algorithms on the reduced search space to obtain the final solution.

Our Proposed Method: COMBHELPER

As mentioned above, most of the existing CO algorithms suffer from efficiency issues due to large search space. To mitigate this issue, we propose a framework called COMBHELPER (illustrated in Figure 1) to prune the search space for CO problems and hence accelerate the solution generation process. Given an undirected graph $\mathcal{G}$, our goal is to predict the importance of each node and find the nodes that are likely to be included in the final solution. To achieve this, we apply a GNN-based architecture, COMBHELPER, to identify the quality of the nodes. For simplicity, we exemplify by GCN (Kipf and Welling 2016) but any other model can be used. COMBHELPER has three major steps and each step is discussed separately in the following sections:

- **Classification with GCN:** We train a GCN model to identify promising nodes that are likely to be included in the solution and subsequently, prune the search space.
- **Knowledge Distillation and Problem-Specific Boosting:** To further improve the performance of the trained GCN model, we adopt the KD strategy during the training procedure and design a problem-specific boosting module for different CO problems.
- **Final Solution Generation:** Lastly, to generate the final solution, we again apply the traditional CO algorithms on the reduced search space.

Classification with GCN

First, we view the problem as a classification task where we classify the nodes based on their potential to be included in the solution set. The GCN model is trained in a supervised manner on the nodes with their ground truth labels. We use three traditional CO algorithms to generate the labels: (1) Linear Programming (LP), (2) Greedy algorithm (GD), and (3) Local Search (LS). Here, we show how to use LP to generate ground truth labels, which are often used to achieve optimal solutions for CO problems.

Generate Labels. We take the MVC problem defined in the previous section as an example: for a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, each node $v \in \mathcal{V}$ is denoted as a binary variable x_v. Then the LP problem can be formulated as follow:

$$\min \sum_{v \in \mathcal{V}} x_v \tag{1}$$

$$x_v \in \{0, 1\} \tag{2}$$

$$x_v + x_u \geq 1, (v, u) \in \mathcal{E} \tag{3}$$

where Equation 1 is the objective of MVC that minimizes the number of nodes in the solution; Equation 2 means the value of each node variable is selected as 0 or 1, where $x_v = 1$ means node v is included in the solution and $x_v = 0$ otherwise; Equation 3 makes sure that at least one node is included in the solution for each edge in the graph. After solving the LP problem, we obtain the final solution $\mathcal{S} = \{v | x_v = 1\}$ that helps us to create the one-hot labels for each node in the graph, i.e., $[1, 0]$ for $v \notin \mathcal{S}$ and $[0, 1]$ for $v \in \mathcal{S}$.

GCN Training. We describe the training procedure for GCN as follows. Assume $\mathcal{D} = (\mathcal{G}, \mathbf{Y})$ is a training instance, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the input graph and $\mathbf{Y} \in \mathbb{R}^{|\mathcal{V}| \times 2}$ is the ground truth label generated by LP, i.e. $\mathbf{y}_v$ is the one-hot label of node v. $\mathcal{G}$ is also associated with a feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, where the i-th row of $\mathbf{X}$ represents a d-dimensional feature vector of node v_i. Let $f(\mathcal{G}; \Theta)$ denote the GCN parameterized by Θ and its forward propagation process is illustrated as follows:

We initialize the input embedding of each node v as $h_v^{(0)} = \mathbf{x}_v$ and the embedding in the $(k+1)$-th layer is computed from that in the k-th layer as follow:

$$h_v^{(k+1)} = \sigma(h_v^{(k)} \theta_1^{k+1} + \sum_{u \in \mathcal{N}(v)} h_u^{(k)} \theta_2^{k+1}) \tag{4}$$

where $\mathcal{N}(v)$ is a set that contains the neighboring nodes of v, $\sigma(\cdot)$ is the ReLU (Nair and Hinton 2010) activation function, $\theta_1^{k+1} \in \mathbb{R}^{d^{(k)} \times d^{(k+1)}}$, $\theta_2^{k+1} \in \mathbb{R}^{d^{(k)} \times d^{(k+1)}}$ are trainable parameters and $d^{(k)}$, $d^{(k+1)}$ are the dimensions of the k-th layer and $(k+1)$-th layer, respectively.

Our objective is to minimize the cross-entropy loss during the supervised training process, which is as follow:

$$\mathcal{L}_{label} = -\sum_{v \in \mathcal{V}_{train}} \mathbf{y}_v \log(\hat{\mathbf{y}}_v) \tag{5}$$

where $\mathcal{V}_{train}$ is the set of training nodes, $\mathbf{y}_v$ is the label of node v, $\hat{\mathbf{y}}_v = \text{softmax}(z_v)$ is the prediction and $z_v = f_v(\mathcal{G}; \Theta)$ is the output logits of node v, where the logits is a non-normalized probability vector, which is always passed to a normalization function such as softmax$(\cdot)$ or sigmoid$(\cdot)$.

Knowledge Distillation & Boosting

To further improve the efficiency and efficacy of COMBHELPER, we apply a KD method on the trained GCN. More specifically, we integrate a problem-specific weight boosting module into the KD framework in order to compress the GCN model and improve its performance.

Knowledge Distillation. KD is used to reduce the number of parameters in the trained GCN model and improve the efficiency of COMBHELPER. We adapt the KD method introduced in (Hinton, Vinyals, and Dean 2015), which is widely used in many frameworks. The main idea is to transfer the knowledge learned by a larger teacher model to a simpler or smaller student model without loss of quality. In COMBHELPER, the trained GCN is the teacher model, which learns knowledge about the nodes from the traditional CO algorithms such as LP on the small graphs. Its knowledge is transferred to a student GCN model by minimizing the cross-entropy loss between its output logits (z_v^t) and the output logits (z_v^s) of the student GCN. This can be formulated as follows:

$$\mathcal{L}_{KD} = -\sum_{v \in \mathcal{V}_{train}} \hat{\mathbf{y}}_v^t \log(\hat{\mathbf{y}}_v^s) \tag{6}$$

where $\hat{\mathbf{y}}_v^t = \text{softmax}(z_v^t/T)$ and $\hat{\mathbf{y}}_v^s = \text{softmax}(z_v^s/T)$ are the predictions of the teacher GCN and the student GCN respectively, and T is the temperature parameter used to soften the output logits of GCN.

Boosting Module. In this study, we consider two different CO problems. To increase the efficacy of COMBHELPER, we design an additional problem-specific boosting module. This module improves the supervised training process of the student GCN. Such boosting module has been used in a different context to improve the training of the student model (Guo et al. 2023). The main idea is to amplify the weights of the nodes that are wrongly classified by the teacher GCN. This helps the student model to pay more attention to those nodes and to make a more precise prediction. We apply the weight updating function in the Adaboost algorithm (Freund, Schapire, and Abe 1999; Sun, Zhu, and Lin 2019) to update the node weights since our task is binary node classification. For each node $v \in \mathcal{V}_{train}$, its node weight is initialized as $w_v = \frac{1}{|\mathcal{V}_{train}|}$ and updated as follow:

$$w_v = \begin{cases} w_v \cdot \exp(-\frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}) & (7) \\ w_v \cdot \exp(\frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}) & (8) \end{cases}$$

where ϵ is the error rate of the teacher GCN. The node weight is updated by Equation 7 if it is classified correctly and updated by Equation 8 if it is misclassified.

Problem-Specific Design. To make our weight-boosting function problem-specific, we incorporate graph structural information such as node degree. In general, nodes with high degrees are more likely to be included in the solution set of the MVC problem, and nodes with low degrees are more likely to be included in the solution set of the MIS problem. We exploit this intuition and let the student GCN focus more on the nodes with high degrees while addressing the MVC problem and on the ones with low degrees for the MIS problem. The problem-specific weight function can be formulated as follows:

$$w_v = w_v \cdot \text{norm}(deg_v) \qquad \text{(For MVC)} \tag{9}$$

$$w_v = w_v \cdot \text{norm}(1/deg_v) \qquad \text{(For MIS)} \tag{10}$$

where deg_v is the degree of node v, norm$(\cdot)$ is the normalization function, e.g., $\text{norm}(deg_v) = \frac{deg_v}{\sum_{v \in \mathcal{V}_{train}} deg_v}$.

Overall Objective. The loss function of our framework COMBHELPER consists of two parts: the KD loss and the supervised training loss. They are combined with parameter λ, which is used to control the balance between them:

$$\begin{aligned} \mathcal{L} &= \lambda \mathcal{L}_{KD} + (1-\lambda)\mathcal{L}_{label} \\ &= -\lambda \sum_{v \in \mathcal{V}_{train}} \hat{\mathbf{y}}_v^t \log(\hat{\mathbf{y}}_v^s) - (1-\lambda) \sum_{v \in \mathcal{V}_{train}} w_v \mathbf{y}_v \log(\hat{\mathbf{y}}_v^s) \end{aligned} \tag{11}$$

Solution Generation

The last step of COMBHELPER is to generate the final solution. Firstly, we use the trained student GCN to assign a binary label for each node. Nodes predicted as label 1 (defined as *good nodes*) will be reserved in the search space since they are more likely to be included in the final solution set. Afterwards, we perform the traditional CO algorithms on the reduced search space (i.e., *the set of good nodes*) to obtain the final solution.

We demonstrate the MVC problem as an example and perform LP on the *good nodes* (denote as $\mathcal{V}_g$). Then the LP problem can be formulated as:

$$\min \sum_{v \in \mathcal{V}_g} x_v \tag{12}$$

$$x_v \in \{0, 1\} \tag{13}$$

$$x_v + x_u \geq 1, (v, u) \in \mathcal{E}, v \in \mathcal{V}_g \text{ or } u \in \mathcal{V}_g \tag{14}$$

where Equation 12 is the modified objective that minimizes the number of *good nodes* in the solution; Equation 13 means the value of each *good node* variable is selected from 0 and 1, where $x_v = 1$ means node v is included in the final solution and $x_v = 0$ otherwise; Equation 14 ensures that at least one *good node* is included in the solution for each edge in the graph.

Besides LP, we also perform GD and LS on *the set of good nodes* to obtain the final solution for the MVC problem. In addition, we generate solutions for the MIS problem with all these three traditional CO algorithms (Please see the supplementary in (Tian, Medya, and Ye 2023) for more details).

Experimental Evaluation

Experimental Setup

Datasets. Both synthetic and real-world datasets are used in our experiments and they are introduced as follows:

- **Synthetic datasets:** We use Barabási-Albert (BA) graph (Albert and Barabási 2002), which can model many real-world networks. We set the edge density to 4 and use BA-N to denote the BA graph with N nodes.
- **Real-world datasets:** The statistics of the datasets used in our experiments are shown in Table 1. Pubmed (Sen et al. 2008) and DBLP (Pan et al. 2016) are citation networks and other datasets are SNAP networks (Leskovec and Krevl 2014). All the networks are undirected.

Dataset	Nodes #	Edges #
Alpha	3.8K	24.2K
OTC	5.9K	35.6K
DBLP	17.7K	52.9K
Pubmed	19.7K	44.3K
Brightkite	58.2K	214.1K
Slashdot0811	77.4K	905.5K
Slashdot0922	82.2K	948.5K
Gowalla	196.6K	950.3K

Table 1: Statistics of real-world datasets.

Baselines. We select three traditional CO algorithms (mentioned in the Problem Formulation section) as the baselines: (1) linear programming (LP), (2) greedy algorithm (GD), and (3) local search (LS). For each baseline, we compare it with two improved versions: Baseline+COMBHELPER$_{pt}$ (COMBHELPER$_{pt}$ means COMBHELPER without KD and boosting modules, where pt stands for **p**re-trained **t**eacher) and Baseline+COMBHELPER.

Training and Testing. For synthetic datasets, COMBHELPER is trained on BA graphs with 1000 nodes (BA-1K) and tested on BA graphs with n nodes, where n ranges from $\{5K, 10K, 20K, 50K, 100K\}$. For real-world datasets, COMBHELPER is trained on Cora (Sen et al. 2008) and tested on the datasets listed in Table 1. See the supplementary in (Tian, Medya, and Ye 2023) for more details.

Evaluation Metrics. (1) To evaluate the efficiency of COMBHELPER, we compare the running time of each baseline with its improved versions (with COMBHELPER). (2) In addition, we use the average speed-up to reflect the efficiency of COMBHELPER, which is calculated by $speedup = \frac{time_B}{time_I}$, where $time_B$ and $time_I$ are the running times of the **B**aseline and its **I**mproved version (with COMBHELPER) respectively. (3) To evaluate the efficacy or quality of COMBHELPER, we report the solution size on the MIS problem (the larger the better). For the MVC problem, we report the solution size with the coverage (since we cannot cover all the edges, see the supplementary in (Tian, Medya, and Ye 2023) for more details). The coverage (the higher the better) is calculated by $coverage = \frac{|\mathcal{E}^*|}{|\mathcal{E}|}$, where $|\mathcal{E}^*|$ is the number of edges covered by the solution.

Results

We show that our proposed method produces better or comparable results (see the supplementary in (Tian, Medya, and Ye 2023) for results of synthetic datasets) while being much faster in all settings.

Performance on LP. All the LP-based solutions are obtained by CPLEX 12.9 (Nickel et al. 2021) and we set the time limit to 1 hour. Table 2 shows the results of the MVC problem. The solutions generated by our methods achieve high coverage and the solution sizes are close to the optimal ones, especially for LP+COMBHELPER. For example, the solution generated by LP+COMBHELPER on Pubmed is optimal. As expected, Table 3 also shows that LP+COMBHELPER outperforms LP+COMBHELPER$_{pt}$ on the MIS problem, and the solution sizes are close to the optimal ones. We attribute the high-quality solutions to the well-reduced search space that increases the purity of *good nodes*.

Datasets	LP	LP + COMBHELPER$_{pt}$	LP + COMBHELPER
OTC	1535	1509 (99.84)	1538 (**100.00**)
Pubmed	3805	3789 (99.96)	3806 (**100.00**)
Brightkite	21867	21141 (99.61)	21918 (**99.98**)
Slashdot0811	24046	23903 (99.92)	24098 (**99.98**)
Slashdot0922	25770	25698 (99.93)	25928 (**99.99**)
Gowalla	84223	82874 (99.82)	84471 (**99.99**)

Table 2: MVC results (solution size with coverage of all edges in the parenthesis) of LP on real-world datasets. COMBHELPER always outperforms COMBHELPER$_{pt}$ in terms of coverage and obtains the best solution on Pubmed.

After showing the superior quality of our method, we now investigate its efficiency. Figures 3 and 4 present that LP with COMBHELPER$_{pt}$ and COMBHELPER take significantly lower time to generate solutions for both the MVC and MIS problems than general LP. On the synthetic datasets, general LP can not solve both problems within 1 hour but COMBHELPER$_{pt}$ and COMBHELPER generate solutions in less than 1 second. For real-world datasets, Figure 2 shows that LP with COMBHELPER$_{pt}$ and COMBHELPER are about 4 times faster than the general LP on the MVC problem and gain a speed-up range from 9.7 to 23.7 on the MIS problem. Our method is more efficient since un-

Method	OTC	Pubmed	Brightkite	Slashdot0811	Slashdot0922	Gowalla
LP	4346	15912	36361	53314	56398	112363
LP+COMBHELPER$_{pt}$	4222	15637	32981	48445	50502	89790
LP+COMBHELPER	**4289**	**15896**	**35776**	**50903**	**53848**	**110569**
GD	4342	15912	36119	53314	56393	112214
GD+COMBHELPER$_{pt}$	4222	15637	32981	48445	50502	89790
GD+COMBHELPER	**4287**	**15896**	**35767**	**50903**	**53845**	**110462**
LS	4100	14501	34750	51323	54184	102349
LS+COMBHELPER$_{pt}$	4222	15637	32981	48445	50502	89790
LS+COMBHELPER	**4288**	**15873**	**35608**	**50897**	**53840**	**108606**

Table 3: MIS results (solution size) on real-world datasets and the best results are marked in bold. Baseline+COMBHELPER outperforms Baseline+COMBHELPER$_{pt}$ and even generates better solutions than the Baseline in some cases.

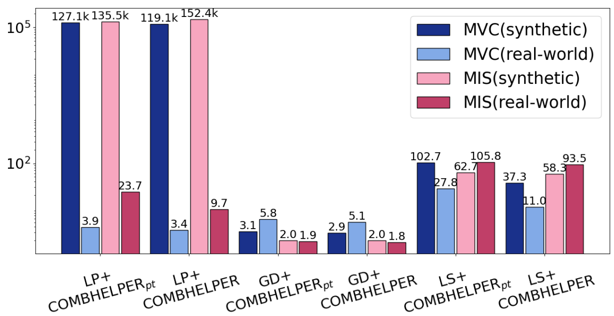


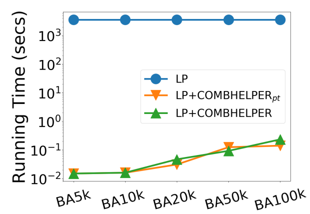
Figure 2: Average speed-up on both synthetic and real-world datasets. Baselines (LP, GD, and LS) with COMBHELPER$_{pt}$ and COMBHELPER are at least 2 times faster than their original versions.

likely nodes are dropped, reducing the search space size of LP.

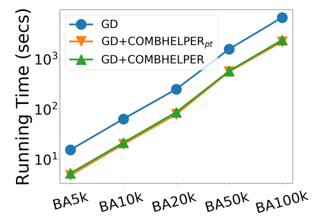
Performance on GD. Greedy (GD) has been a popular algorithm for many graph CO problems. Similar to the case in LP, we demonstrate that our method improves GD. Tables 3 and 4 present a similar conclusion that GD with both COMBHELPER$_{pt}$ and COMBHELPER still generate high-quality solutions for both the MVC and MIS problems. In specific, GD+COMBHELPER obtains better MVC solutions (cover all the edges with fewer nodes) than the general GD on OTC and Pubmed, which means COMBHELPER obtains a better search space for solution generation.

Datasets	GD	GD + COMBHELPER$_{pt}$	GD + COMBHELPER
OTC	1555	1519 (99.84)	1551 (**100.00**)
Pubmed	3866	3811 (99.96)	3846 (**100.00**)
Brightkite	22159	21285 (99.61)	22090 (**99.98**)
Slashdot0811	24347	23995 (99.92)	24339 (**99.98**)
Slashdot0922	26054	25828 (99.93)	26182 (**99.99**)
Gowalla	85450	83694 (99.82)	85343 (**99.99**)

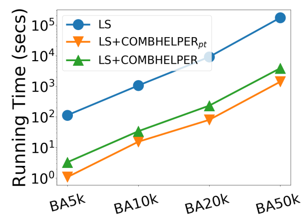
Table 4: MVC results (solution size with coverage of all edges in the parenthesis) of GD on real-world datasets. COMBHELPER always shows the best performance on GD.



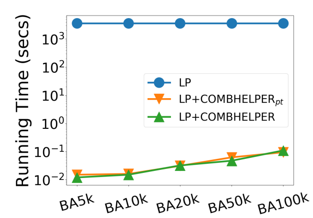
(a) LP on MVC



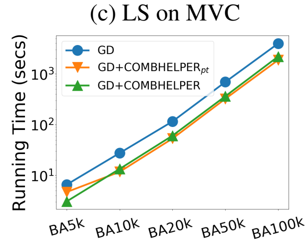
(b) GD on MVC



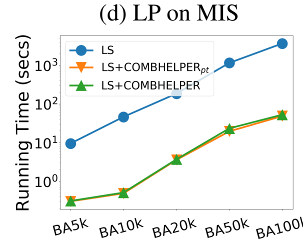
(c) LS on MVC



(d) LP on MIS



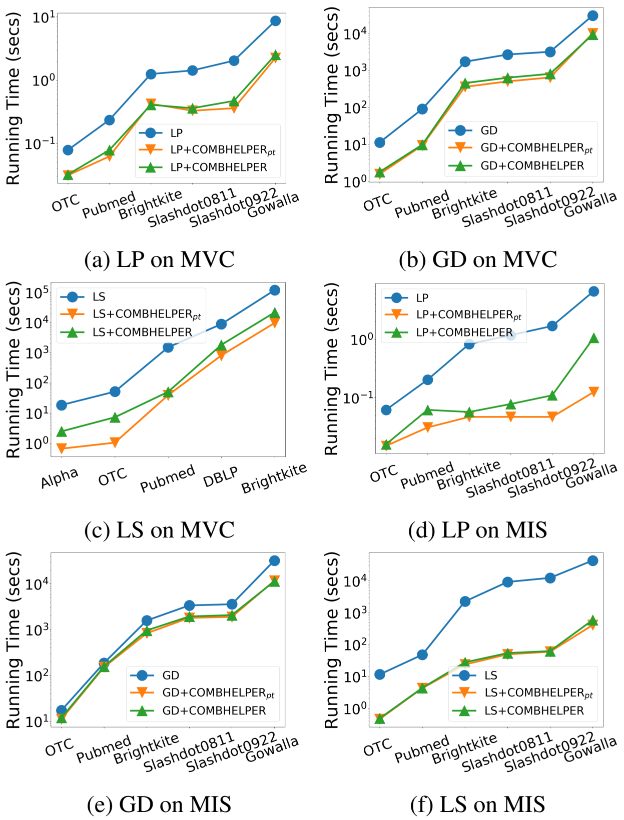
(e) GD on MIS



(f) LS on MIS

Figure 3: Running times (seconds) on synthetic datasets. Our method takes less time to generate solutions for both the MVC and MIS problems. Note that all the running times of general LP (blue line) in Figure 3a and 3d reach the time limit of 1 hour.

Figures 3 and 4 present the running times of GD and our methods. It is clear that both COMBHELPER$_{pt}$ and COMBHELPER accelerate the process of solution generation in all the settings. Figure 2 also indicates that GD with our method is about 2 to 6 times faster than the general GD, owing to the search space pruning procedure in COMBHELPER. We notice that the speed-up of GD is not so obvious as in LP and LS due to the limitation of GD itself, i.e., the efficiency of GD relies on the nature of CO problems (see the supplementary in (Tian, Medya, and Ye 2023) for further explanation).



(a) LP on MVC (b) GD on MVC

(c) LS on MVC (d) LP on MIS

(e) GD on MIS (f) LS on MIS

Figure 4: Running times (seconds) on real-world datasets. Our method generates solutions for both the MVC and MIS problems more efficiently.

Performance on LS. Similar observations can be seen from another popular CO algorithm called local search (LS). Since LS costs about 32 and 48 hours to generate MVC solutions on Brightkite and BA50k respectively, we do not test COMBHELPER on Slashdot0811, Slashdot0922, Gowalla, and BA100k for the MVC problem. Instead, we use another two small datasets Alpha and DBLP.

Tables 3 and 5 demonstrate that LS+COMBHELPER generate the best solutions in most cases. Specifically, for the MVC problem, the solution sizes of LS+COMBHELPER on Alpha, OTC, and Pubmed are smaller than those of LS. For the MIS problem, the solution sizes of LS+COMBHELPER are larger than those of LS on all the datasets except Slashdot0811 and Slashdot0922.

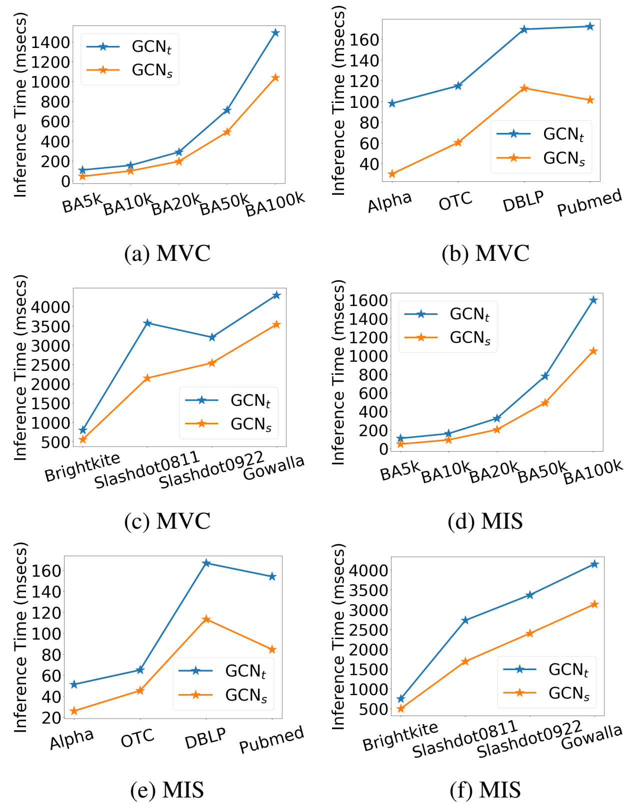
From Figures 3 and 4, it is easy to see that LS with COMBHELPER$_{pt}$ and COMBHELPER obtain the solutions more efficiently. Figure 2 also shows that LS with COMBHELPER$_{pt}$ and COMBHELPER gain a speed-up range from 11.0 to 102.7 on the MVC problem and are about 80 times faster than the general LS on the MIS problem. The major reason for such improvement in efficiency is that COMBHELPER accelerates both phases in LS: (1) initial solution generation and (2) neighborhood solution exploration.

Datasets	LS	LS + COMBHELPER$_{pt}$	LS + COMBHELPER
Alpha	1097	1050 (99.84)	1072 (**100.00**)
OTC	1564	1517 (99.84)	1552 (**100.00**)
Pubmed	3896	3799 (99.96)	3819 (**100.00**)
DBLP	8323	7818 (99.03)	8315 (**99.99**)
Brightkite	22622	21417 (99.96)	22187 (**99.99**)

Table 5: MVC results of LS on real-world datasets. Solutions generated by LS+COMBHELPER are the best.

Ablation Study

Efficiency of KD. In order to verify the efficiency of KD, we compare the inference time (time of predicting *good nodes*) of the teacher model (GCN$_t$) with that of the student model (GCN$_s$). From Figure 5, it is obvious that GCN$_s$ costs less time than GCN$_t$ in all the sittings since the model compressed via KD has fewer parameters and a simpler structure, which predicts the *good nodes* and reduce the search space more efficiently.



(a) MVC (b) MVC

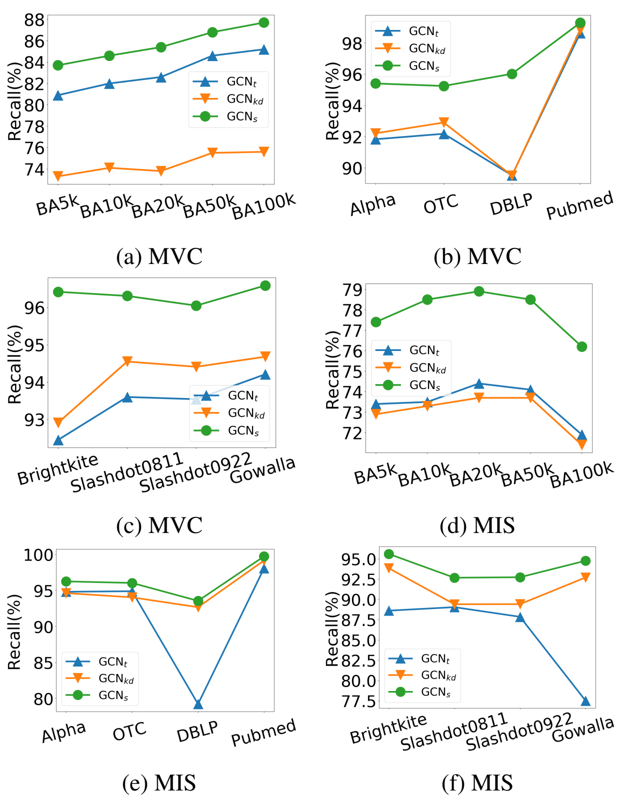
(c) MVC (d) MIS

(e) MIS (f) MIS

Figure 5: Inference time (milliseconds) on synthetic and real-world datasets. GCN$_s$ costs less time than GCN$_t$ in all the cases.

Efficacy of Problem-Specific Boosting. In our case, if we can predict the nodes in the optimal solution as more as possible, we can prune the search space more precisely. So we use a widely used binary classification metric called *recall* to evaluate the performance of GCN. It is formulated as $recall = \frac{TP}{TP+FN}$, where TP means **T**rue **P**ositive and FN means **F**alse **N**egative.

Figure 6 shows the recall of GCN models with different modules, where GCN$_t$ is the teacher GCN without KD and boosting, GCN$_{kd}$ is the student GCN with only KD, and GCN$_s$ is the student GCN with both KD and the problem-specific boosting module. GCN$_{kd}$ achieves higher recall than GCN$_t$ in some cases (Figure 6c and 6f) while it achieves recall close to GCN$_t$ (Figure 6b and 6e) and even lower than GCN$_t$ (Figure 6a and 6d). In contrast, GCN$_s$ obtains the highest recall in all the cases. Moreover, Tables 2 and 3 indicate that solutions generated by COMBHELPER (using GCN$_s$) achieve higher coverage on the MVC problem and have larger solution sizes on the MIS problem than those generated by COMBHELPER$_{pt}$ (using GCN$_t$). Based on all the above, we draw a conclusion that GCN with problem-specific boosting has better performance because it pays more attention to the misclassified and problem-related nodes.



(a) MVC (b) MVC

(c) MVC (d) MIS

(e) MIS (f) MIS

Figure 6: Recall on synthetic and real-world datasets. GCN$_s$ always achieves higher recall than both GCN$_t$ and GCN$_{kd}$.

Conclusion

In this paper, we have proposed a novel framework called COMBHELPER to improve the efficiency of existing CO algorithms on large graphs by precisely reducing the search space. We have applied a GNN-based model to identify the promising nodes to prune the search space. Afterwards, we have applied the traditional CO algorithms on the reduced search space to obtain the final solution. Moreover, COMBHELPER adopts a KD framework with our designed problem-specific boosting module for further improvement. The efficiency and efficacy of our proposed COMBHELPER have been demonstrated on both synthetic and real-world datasets across two graph CO problems. Traditional CO algorithms with COMBHELPER are at least 2 times faster than their original versions.

Acknowledgments

We thank the anonymous reviewers for their valuable and constructive comments. This work was supported partially by the National Natural Science Foundation of China (grant # 62176184), and the Fundamental Research Funds for the Central Universities of China.

References

Ahn, S.; Seo, Y.; and Shin, J. 2020. Learning what to defer for maximum independent sets. In *International Conference on Machine Learning*.

Albert, R.; and Barabási, A.-L. 2002. Statistical mechanics of complex networks. *Reviews of modern physics*.

Andrade, D. V.; Resende, M. G.; and Werneck, R. F. 2012. Fast local search for the maximum independent set problem. *Journal of Heuristics*.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bansal, N.; and Khot, S. 2010. Inapproximability of hypergraph vertex cover and applications to scheduling problems. In *International Colloquium on Automata, Languages, and Programming*.

Barrett, T.; Clements, W.; Foerster, J.; and Lvovsky, A. 2020. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Chaoji, V.; Ranu, S.; Rastogi, R.; and Bhatt, R. 2012. Recommendations to boost content spread in social networks. In *Proceedings of the 21st international conference on World Wide Web*.

Cox, D. R. 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society Series B: Statistical Methodology*.

Dai, H.; Dai, B.; and Song, L. 2016. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*.

Deng, X.; and Zhang, Z. 2021. Graph-Free Knowledge Distillation for Graph Neural Networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*.

Fitzpatrick, J.; Ajwani, D.; and Carroll, P. 2021. Learning to sparsify travelling salesman problem instances. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*.

Freund, Y.; Schapire, R.; and Abe, N. 1999. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2020. Message passing neural networks. *Machine learning meets quantum physics*.

Grassia, M.; Lauri, J.; Dutta, S.; and Ajwani, D. 2019. Learning multi-stage sparsification for maximum clique enumeration. *arXiv preprint arXiv:1910.00517*.

Guo, Z.; Zhang, C.; Fan, Y.; Tian, Y.; Zhang, C.; and Chawla, N. V. 2023. Boosting graph neural networks via adaptive knowledge distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems*.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hochba, D. S. 1997. Approximation algorithms for NP-hard problems. *ACM Sigact News*.

Hoos, H. H.; and Stützle, T. 2004. *Stochastic local search: Foundations and applications*. Elsevier.

Hossain, A.; Lopez, E.; Halper, S. M.; Cetnar, D. P.; Reis, A. C.; Strickland, D.; Klavins, E.; and Salis, H. M. 2020. Automated design of thousands of nonrepetitive parts for engineering stable genetic systems. *Nature biotechnology*.

James, J.; Yu, W.; and Gu, J. 2019. Online vehicle routing with neural combinatorial optimization and deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*.

Khalil, E.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*.

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

Land, A. H.; and Doig, A. G. 2010. *An automatic method for solving discrete programming problems*. Springer.

Lauri, J.; Dutta, S.; Grassia, M.; and Ajwani, D. 2020. Learning fine-grained search space pruning and heuristics for combinatorial optimization. *arXiv preprint arXiv:2001.01230*.

Leskovec, J.; and Krevl, A. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*.

Manchanda, S.; Mittal, A.; Dhawan, A.; Medya, S.; Ranu, S.; and Singh, A. 2020. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. *Advances in Neural Information Processing Systems*.

Mari, M. 2017. Study of greedy algorithm for solving maximum independent set problem. Technical report, University of Liverpool.

Nair, V.; and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*.

Nickel, S.; Steinhardt, C.; Schlenker, H.; Burkart, W.; Reuter-Oppermann, M.; Nickel, S.; Steinhardt, C.; Schlenker, H.; Burkart, W.; and Reuter-Oppermann, M. 2021. Ibm ilog cplex optimization studio. *Angewandte Optimierung mit IBM ILOG CPLEX Optimization Studio: Modellierung von Planungs-und Entscheidungsproblemen des Operations Research mit OPL*.

Pan, S.; Wu, J.; Zhu, X.; Zhang, C.; and Wang, Y. 2016. Tri-party deep network representation. *Network*.

Papadimitriou, C. H.; and Steiglitz, K. 1998. *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine*.

Sun, K.; Zhu, Z.; and Lin, Z. 2019. Adagcn: Adaboosting graph convolutional networks into deep models. *arXiv preprint arXiv:1908.05081*.

Tian, H.; Medya, S.; and Ye, W. 2023. COMBHelper: A Neural Approach to Reduce Search Space for Graph Combinatorial Problems. *arXiv preprint arXiv:2312.09086*.

Vazirani, V. V. 2001. *Approximation algorithms*. Springer.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. *Advances in neural information processing systems*.

Wilder, B.; Ou, H.-C.; de la Haye, K.; and Tambe, M. 2018. Optimizing Network Structure for Preventative Health. In *AAMAS*.

Williamson, D. P.; and Shmoys, D. B. 2011. *The design of approximation algorithms*. Cambridge university press.

Yan, B.; Wang, C.; Guo, G.; and Lou, Y. 2020. Tinygnn: Learning efficient graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

Yang, Y.; Qiu, J.; Song, M.; Tao, D.; and Wang, X. 2020. Distilling Knowledge From Graph Convolutional Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhang, S.; Liu, Y.; Sun, Y.; and Shah, N. 2021. Graph-less Neural Networks: Teaching Old MLPs New Tricks Via Distillation. In *International Conference on Learning Representations*.