UNSUPERVISED LEARNING FOR COMBINATORIAL OPTIMIZATION NEEDS META LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

A general framework of unsupervised learning for combinatorial optimization (CO) is to train a neural network (NN) whose output gives a problem solution through directly optimizing the CO objective. Albeit with some advantages over traditional solvers, the current framework optimizes an averaged performance over the distribution of historical problem instances, which misaligns with the actual goal of CO that looks for a good solution to every future encountered instance. With this observation, we propose a new objective of unsupervised learning for CO where the goal of learning is to search good initialization for future problem instances rather than give direct solutions. We propose a meta-learning-based training pipeline for this new objective. Our method achieves good empirical performance. We observe that even just the initial solution given by our model before fine-tuning can significantly outperform the baselines under various evaluation settings including cross multiple datasets, and the case with big shifts in the problem scale. The reason we conjecture is that meta-learning-based training lets the model loosely tied to each local optima for a training instance while being more adaptive to the changes of optimization landscapes across instances.

1 INTRODUCTION

Combinatorial optimization (CO), aiming to find out the optimal solution from discrete search space, has pivotal position in scientific and engineering fields (Papadimitriou & Steiglitz, 1998; Crama, 1997). Most CO problems are NP-complete or NP-hard. Conventional heuristics or approximation requires insightful comprehension of the particular problem. Starting from the seminal work from Hopfield & Tank (1985), researchers apply neural networks (NNs) (Smith, 1999; Vinyals et al., 2015) to solve CO problems. The motivation is that NNs may learning heuristics through solving historical problems, which could be useful to solve similar problems in the future.

Many NN-based methods (Selsam et al., 2018; Joshi et al., 2019; Hudson et al., 2021; Gasse et al., 2019; Khalil et al., 2016) require optimal solutions to the CO problem as supervision in training. However, optimal solutions are hard to get in practice and the obtained model often does not generalize well (Yehuda et al., 2020). Methods based on reinforcement learning (RL) (Mazyavkina et al., 2021; Bello et al., 2016; Khalil et al., 2017; Yolcu & Póczos, 2019; Chen & Tian, 2019; Yao et al., 2019; Kwon et al., 2020; 2021; Delarue et al., 2020; Nandwani et al., 2021) do not need labels while they often suffer from notoriously unstable training. Recently, unsupervised learning methods have attracted much attention (Toenshoff et al., 2021; Amizadeh et al., 2018; Yao et al., 2019; Karalias & Loukas, 2020; Wang et al., 2022). A common strategy of these methods is to design an NN whose output gives a solution to the CO problem and then train the NN via gradient descent by directly optimizing the CO objectives over a set of training instances. This strategy is superior in its faster training, good generalization, and strong capability of dealing with large-scale problems.

Despite the prominent progress, current unsupervised learning methods always optimize NNs towards an *averaged good* performance over training instances. This means even if a testing instance comes from the same distribution of the training instances, the solution to this single instance may not have good quality, let alone the case when the testing instance is out-of-distribution (OOD). This induces a concern when we apply NNs in practice because practical problems often expect to have a good solution to every encountered instance. For example, allocating surveillance cameras is crucial for each-time exhibition in every art gallery. Solvers when applied to this problem (O'rourke,

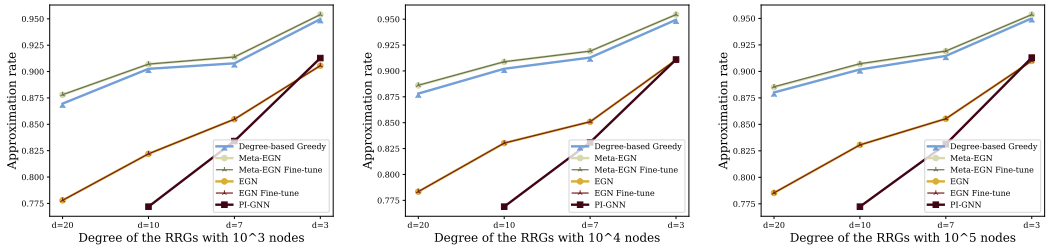


Figure 1: Approximation Rates of different methods in the MIS problem. Meta-EGN and EGN (Karalias & Loukas, 2020) are trained on RRGs with 1000 nodes and with node degree randomly sampled from $3, 7, 10, 20$. Meta-EGN and EGN are evaluated over larger RRGs with $10^3 \sim 10^5$ nodes. More details about the setting are in Secs. 5.1 and 5.4. Meta-EGN outperforms DGA (Angelini & Ricci-Tersenghi, 2019) by about $0.3\% - 0.5\%$ in approximation rates on average.

1987; Yabuta & Kitazawa, 2008) should output a good solution every time. Traditional CO solvers are designed towards this goal. However, it is time-consuming and unable to learn heuristics from historical instances. So, can we leverage the benefit of learning from history while with the goal of achieving an instance-wise good solution instead of an averaged good solution?

This motivates us to study a new formulation of unsupervised learning for CO. We regard the objective of learning from history as to search a good initialization for each future instance rather than give a direct solution. Since in practice, future instances are unavailable during the training stage, we propose to view each training instance as a pseudo-new instance for the rest training instances. Then, our learning objective is to learn a good initialization of this model, such that further optimization of the initialization could achieve good solutions on each of these pseudo-new instances. We observe meta learning is suitable to implement this idea, and propose to adopt MAML (Finn et al., 2017) in our training pipeline as a proof of concept. Note that the step of optimization on each pseudo-new instance shares a similar spirit with fine-tuning a model over each down-streaming task as traditional meta learning does. However, each task in our case corresponds to optimization over each training instance.

We name our method Meta-EGN by extending the previous framework EGN (Karalias & Loukas, 2020) via meta learning. Our key observation is that with this new objective, even the initial solution given by Meta-EGN (before fine-tuning on a test instance) is substantially better than the solution given by EGN and other methods that optimize the averaged performance over training instances. Our conjectured reason is that the new objective, by taking into account fine-tuning the model over new instances, trains the model to avoid being trapped into a local minima induced by each training instance while being more adaptive to the changes of optimization landscapes across instances.

We demonstrate the benefits of Meta-EGN via experiments within three benchmark CO problems (max clique, vertex cover and max independent set) on multiple synthetic graphs and three real-world graph datasets, with the number of nodes ranging from 100 to 5000. Meta-EGN significantly outperforms state-of-the-art learning-based baselines (Karalias & Loukas, 2020; Toenshoff et al., 2021), greedy algorithms, and the commercial CO solver Gurobi9.5 (Gurobi Optimization, 2022) in most cases. Meta-EGN also shows super OOD generalization performance when the training and test datasets are different or have graphs of entirely different sizes.

Moreover, recently, Angelini & Ricci-Tersenghi (2022) have shown that the learning-based method in (Schuetz et al., 2022) could not achieve comparable results with the degree-based greedy algorithm (DGA) (Angelini & Ricci-Tersenghi, 2019) in the max independent set (MIS) problem on large-scaled random-regular graphs (RRGs), which raises attentions from machine learning community. We observe the issues come from two aspects: (1) graph neural networks (GNNs) used to encode the regular graph suffer from the node ambiguity issue due to their limited expressive power (**?**); (2) the model in (Schuetz et al., 2022) was not trained and learned from history. By addressing these two issues, Meta-EGN can consistently outperform DGA while maintaining the same time complexity to generate solutions. Fig. 1 show the results.

2 Related Work

In the following, we review two groups of works: unsupervised learning for CO and meta learning.

Previous works on unsupervised learning for CO have studied max-cut Yao et al. (2019) and TSP problems Hudson et al. (2021), while these works depend on carefully selected problem-specific objectives. Some works have investigated satisfaction problems Amizadeh et al. (2018); Toenshoff et al. (2019). Applying these approaches to general CO problems requires problem reductions. The works most relevant to us are Karalias & Loukas (2020), Wang et al. (2022) and Schuetz et al. (2022). Karalias & Loukas (2020) propose an unsupervised learning framework EGN for general CO problems based on the Erdős's probabilistic method, which bonds the quality of the final solutions with probability. Wang et al. (2022) generalize EGN and prove that if the CO objective can be relaxed into an entry-wise concave form, a solution of good quality can be deterministically achieved. This further inspires the design of proxy objectives for the CO problems that may not have closed-form objectives, such as those in circuit design. Schuetz et al. (2022) have recently extended EGN to large-scale max independent set problems on random-regular graphs.

Meta learning is proposed to learn hyper-parameters or initialization from historical tasks and achieve fast adaption to new tasks. Finn et al. (2017) propose model-agnostic meta learning (MAML), which aimed to obtain good parameter initialization and accommodated to few-shot learning tasks with limited steps of fine-tuning. Nichol et al. (2018) accelerate MAML by adopting first-order approximation on the gradient estimation. Rajeswaran et al. (2019) introduce implicit-MAML that adopts an objective with fine-tuning till the stationary points on new tasks. Implicit-MAML does not fit our case because we try to avoid long-time fine-tuning. Hsu et al. (2018) study unsupervised learning under the meta learning framework and focused exclusively on vision tasks. To the best of our knowledge, our work is the first one to apply meta learning to unsupervised learning for CO.

3 Preliminaries: Notations and Problem Formulation

Combinatorial Optimization on Graphs. We follow the settings considered in (Karalias & Loukas, 2020; Wang et al., 2022; Schuetz et al., 2022) and study CO problems on graphs whose solutions can be represented as a subset of nodes of the input graph instance, although our method could be applied to a broader range of problems. Suppose $\mathcal{G}$ is the universe of graph instances. Let $G(V, E) \in \mathcal{G}$ denote a graph instance where $V = \{1, 2, ..., n\}$ is the node set and E is the edge set. Let $X = (X_i)_{1 \le i \le n} \in \{0, 1\}^n$ denote the discrete optimization variables defined on V, where $X_i = 1$ denotes that node i is selected in the output node subset. A CO problem on G consists of a cost function $f(\cdot; G) : \{0, 1\}^n \to \mathbb{R}_{\ge 0}$ and a feasible set $\Omega \subseteq \{0, 1\}^n$ that stands for the finite set of all feasible X's, and asks to solve

$$\min_{X \in \{0,1\}^n} f(X; G) \qquad \text{s.t.} \quad X \in \Omega \tag{1}$$

Unsupervised Learning for CO. The Erdös-Goes-Neural (EGN) framework of unsupervised learning for CO proposed in (Karalias & Loukas, 2020) is reviewed as follows. Here, we use the notation system in a follow-up work (Wang et al., 2022) as it is more clear. Learning for CO problem is to learn an algorithm $\mathcal{A}_\theta(\cdot) : \mathcal{G} \to \{0, 1\}^n$ typically parameterized by an NN where θ denotes the parameters of the NN such that given a graph instance G, $X = \mathcal{A}_\theta(G)$ gives a solution of Eq. 1. In practice, directly optimizing the parameters θ is hard in general.

Therefore, we may consider a relaxed cost function $f_r(\cdot; G) : [0, 1]^n \to \mathbb{R}_{\ge 0}$ where $f_r(X; G) = f(X; G)$ on any discrete points $X \in \{0, 1\}^n$ and a relaxed constraint $g_r(\cdot; G) : [0, 1]^n \to \mathbb{R}_{\ge 0}$ where $\{X \in \{0, 1\}^n : g_r(X; G) = 0\}$ and $\{X \in \{0, 1\}^n : g_r(X; G) \ge 1\}$ defines the feasible set Ω and the infeasible set Ω^c respectively. Also, suppose the NN in $\mathcal{A}_\theta$ can give soft solutions $\bar{X} \in [0, 1]^n$. Then, we may train θ by minimizing a label-independent loss function:

$$\min_\theta \; l(\theta; G) \triangleq f_r(\bar{X}; G) + \beta g_r(\bar{X}; G), \quad \bar{X} = \mathcal{A}_\theta(G), \text{ for some } \beta > 0. \tag{2}$$

The significant observation made by (Wang et al., 2022), which generalizes the argument in (Karalias & Loukas, 2020), is a type of performance guarantee on the condition that f_r and g_r are entry-wise concave, which is satisfied in all the cases studied in this work: If the loss that achieves $l(\theta; G) < \beta$ for some $\beta > \max_{X \in \{0,1\}^n} f(X; G)$, then the discrete solution X obtained by rounding the soft solution $\bar{X} = \mathcal{A}_\theta(G)$ according to Def. 1 is feasible $X \in \Omega$ and of good quality $f(X; G) \le l(\theta; G)$.

Definition 1 (Rounding)**.** *For a soft solution $\bar{X} \in [0, 1]^n$ and an arbitrary order of the entries (w.l.o.g 1,2,...,n), fix all the other entries unchanged and round $\bar{X}_i$ into 0 or 1 as $X_i = \arg\min_{j=0,1} f_r(X_1, ..., X_{i-1}, j, \bar{X}_{i+1}, ..., \bar{X}_n) + \beta g_r(X_1, ..., X_{i-1}, j, \bar{X}_{i+1}, ..., \bar{X}_n)$, replace $\bar{X}_i$ with X_i and repeat this operation until all the entries are discrete.*

4 META LEARNING FOR ERDÖS GOES NEURAL (META-EGN)

The above performance guarantee lays the theoretical foundation for EGN. However, the following practical issue motivates us to incorporate meta learning into EGN.

4.1 MOTIVATION: WHAT NEEDED IS LEARNING FOR INSTANCE-WISE GOOD SOLUTIONS

It is often time consuming to perform online optimization of $l(\theta; G)$ for each encountered instance G. This also mismatchs the goal of learning, i.e., learning heuristics from history/data. Therefore, a pipeline commonly adopted is as follows. Suppose there is a set of training instances G_i, $1 \leq i \leq m$, IID sampled from a distribution $\mathbb{P}_{\mathcal{G}}$. We optimize θ by following

$$\min_{\theta} \sum_{i=1}^{m} l(\theta; G_i), \tag{3}$$

which is similar to empirical risk minimization (ERM) in standard supervised learning. When a test instance G appears, we apply the learned $\mathcal{A}_\theta$ to get a soft solution and round it to the final solution.

This pipeline cannot guarantee the quality for this instance G. Even if the training instances G_i, $1 \leq i \leq m$ are in a large quantity (so in-distribution generalization is not a problem), and even if the test instance G also follows $\mathbb{P}_{\mathcal{G}}$, we may not guarantee a low $l(\theta; G)$ for one particular G because ERM only guarantees a low averaged performance $\mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[l(\theta; G)]$. This issue may also voilate the condition to have performance guarantee as reviewed in Sec. 3, as it is instance-wise. Here, we highlight that in practice even the minimal averaged loss $\min_\theta \mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[l(\theta; G)]$ is often strictly greater than averaged instance-wise minimal loss $\mathbb{E}_{G\sim\mathbb{P}_{\mathcal{G}}}[\min_\theta l(\theta; G)]$, because practical NNs are not expressive enough to remember the optimal solution to every instance.

Unfortunately, many practical CO problems actually expect *instance-wise good solutions*. This is because every instance in practice is crucial. A terrible solution for one instance may raise a security issue (e.g., the surveillance-camera allocation problem) or cause huge economic losses (e.g., the routing problem in a transportation system). With this observation, our work is to address the problem by studying *unsupervised learning for instance-wise good solutions to CO problems*.

4.2 TRAINING TOWARDS INSTANCE-WISE OPTIMALITY VIA META LEARNING

Our idea to address the problem is to regard the goal of learning from history as to search good initialization for future instances rather than give direct solutions. Such good initialization can be quickly fine-tuned by further optimizing the model for each instance, which ultimately gives instance-wise good solutions. However, in practice, we do not have access to any future/test instances. So, can we just use historical/training instances to implement the above idea? Our strategy is to view each training instance G_i as a pseudo-test instance to test and optimize the quality of initialization given by the model. Specifically, this strategy gives us an objective

$$\min_{\theta} \sum_{i=1}^{m} \tilde{l}_i(\theta), \quad \text{where } \tilde{l}_i(\theta) = \min_{\theta_i} l(\theta_i; G_i) \text{ with } \theta_i = \theta \text{ as initialization.} \tag{4}$$

Eq. 4 has some abuse of notations. The minimum $\tilde{l}_i(\theta)$ depending on the initialization θ is because of the non-convex nature of $\min_{\theta_i} l(\theta_i; G_i)$, where the initialization $\theta_i = \theta$ matters significantly.

We further simplify Eq. 4 with some practical consideration. In fact, we may not allow further optimizing θ with so many gradient-descent steps for each instance, especially during the online test stage. As a proof of concept, we consider the case with only one-step gradient descent, which already gives good empirical results. Specifically, our training objective follows

$$\textbf{Our Objective: } \min_{\theta} \sum_{i=1}^{m} l_i(\theta), \quad \text{where } l_i(\theta) = l(\theta_i; G_i) \text{ with } \theta_i = \theta - \alpha \nabla_\theta l(\theta; G_i). \tag{5}$$

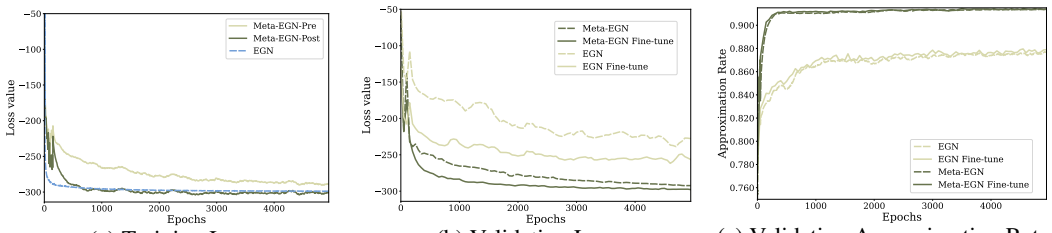
Here, θ is to give a good initialization $\mathcal{A}_\theta(G_i)$ over each instance G_i while θ_i is with one-step fine-tune to achieve a G_i-specified good solution $\mathcal{A}_{\theta_i}(G_i)$.

Optimization in Eq. 5 can be implemented via the meta learning pipeline MAML (Finn et al., 2017). We name the obtained model Meta-EGN and summarize its training and testing in Alg. 1. In step 3,

Algorithm 1 Train Meta-EGN and Test Meta-EGN with/without Fine-tuning

Require: Training instances $\Xi = \{G_1, G_2, ..., G_m\}$; Hyperparameters: α, γ.

1: Randomly initialize $\theta^{(0)}$
2: **for** each randomly sampled mini-batch $B_j \subset \Xi$, $j = 0, 1, ..., K-1$ **do** ▷ Training starts
3: For each $G_i \in B_j$, compute the adapted parameter: $\theta_i^{(j)} = \theta^{(j)} - \alpha \nabla_{\theta^{(j)}} l(\theta^{(j)}; G_i)$
4: Update: $\theta^{(j+1)} \leftarrow \theta^{(j)} - \gamma \nabla_{\theta^{(j)}} \sum_{G_i \in B_j} l(\theta_i^{(j)}; G_i)$
5: **end for**
6: **return** $\theta \leftarrow \theta^{(K)}$ ▷ Training ends
7: For a given testing instance G': ▷ Testing starts
8: **if** fine-tuning is allowed **then**
9: Fine-tune the parameters: $\theta_{G'} \leftarrow \theta - \alpha \nabla_\theta l(\theta; G')$
10: Use Def. 1 to round the relaxed solution given by $\mathcal{A}_{\theta_{G'}}(G')$ ▷ With fine-tuning
11: **else**
12: Use Def. 1 to round the relaxed solution given by $\mathcal{A}_\theta(G')$ ▷ Without fine-tuning
13: **end if** ▷ Testing ends



(a) Training Loss (b) Validation Loss (c) Validation Approximation Rate

Figure 2: Training/validating dynamics of Meta-EGN and EGN Karalias & Loukas (2020) for the MIS problem. Detailed experiment settings follow Sec. 5.1.

Meta-EGN performs the one-step gradient descent on each training instance. Note that we consider two testing cases with or without fine-tuning, because the latter saves much inference time. A simple extension of the Theorem 1 in (Wang et al., 2022) gives a performance guarantee for Meta-EGN in Theorem 1 as follows. Here, for a test instance G, we even allow $l(\theta; G)$ to violate the original condition $l(\theta; G) < \beta$ in (Wang et al., 2022) to some extend. After one-step fine-tuning in step 9, the performance guarantee is still achievable. The detailed proof is in Appendix. A.

Theorem 1 (Performance Guarantee)**.** *Suppose the relaxations f_r and g_r are entry-wise concave as required in (Wang et al., 2022). Let θ denote the learned parameter after training. Given a test instance G, suppose locally $l(\cdot; G)$ is L-smooth at θ, i.e., $\|\nabla_{\theta'} l(\theta'; G) - \nabla_\theta l(\theta; G)\| \leq L\|\theta' - \theta\|$ for all θ' that satisfies $\|\theta' - \theta\| \leq \epsilon$. Then, if $l(\theta; G) < \beta + \triangle$ (even if $l(\theta; G) \geq \beta$), there exists α such that Meta-GNN with one-step finetuning outputs a feasible solution X of good quality $f(X; G) \leq l(\theta; G) - \triangle$. Here, $\triangle = \|\nabla_\theta l(\theta; G)\|\epsilon - \frac{L}{2}\epsilon^2$ if $\epsilon < \frac{1}{L}\|\nabla_\theta l(\theta; G)\|$ or $\triangle = \frac{1}{2L}\|\nabla_\theta l(\theta; G)\|^2$ o.w..*

To better understand Meta-EGN, we show its training/testing dynamics in Fig. 2. As we expected, the training loss of EGN is somewhere in-between the losses of Meta-EGN before and after the fine-tune step. Training EGN is stabler and converges faster than training Meta-EGN. However, what is unexpected is that in validation, Meta-EGN has a much lower loss and achieves much better performance than EGN even before fine-tuning.

This implies that Meta-EGN holds better generalization than EGN. We conjecture the reasons are as follows. First, the optimization landscape for CO problems is extremely non-convex (Mezard & Montanari, 2009) due to the intersected feasible-infeasible regions and the high penalty coefficient β. EGN that has low losses for training instances may give a high loss even when the optimization landscape is just slightly shifted (from training to a test instance). However, the parameters of Meta-EGN are loosely tied to a local minimum for each training instance. Instead, those parameters, as being aware of follow-up instance-wise fine-tuning steps, are likely to fall into some location close to a local minimum for each instance while being not trapped in anyone of them, which makes the model robust to landscape shifts across instances. Second, a CO problem could vary a lot across graph instances even for those generated from the same distribution, especially when the instances

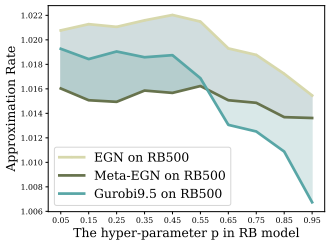
Table 1: Comparison between different unsupervised frameworks. G denotes the test instance and G_i, $1 \leq i \leq m$ are training instances. The standard EGN pipeline does not adopt any fine-tuning.

	EGN (Karalias & Loukas, 2020)	P-I GNN (Schuetz et al., 2022)	Meta-EGN (Ours)	Classical Solver Gurobi Optimization (2022)
Obj. to optimize the NN	$\sum_{i=1}^{m} l(\theta; G_i)$	$l(\theta; G)$	$\sum_{i=1}^{m} l(\theta - \nabla_\theta l(\theta; G_i); G_i)$	$f(X; G)$ s.t. $X \in \Omega$
Training or not	Yes	No	Yes	No
Fine-tune timing	No	Long	Short/No	Long
Generalization	Good	-	Better	-

Table 2: The discrete objectives (Eq. 1) and their relaxations (Eq. 2) for the three problems to be studied.

MC	Discrete Obj.	$\max_X \sum_{1 \leq i \leq n} X_i \quad \text{s.t.} \quad (i,j) \in E \text{ if } X_i, X_j = 1$
	Relaxation	$l_{\text{MC}}(\theta; G) \triangleq -(\beta+1) \sum_{(i,j) \in E} \bar{X}_i \bar{X}_j + \frac{\beta}{2} \sum_{i \neq j} \bar{X}_i \bar{X}_j$
MVC	Discrete Obj.	$\min_X \sum_{1 \leq i \leq n} X_i \quad \text{s.t.} \quad X_i + X_j \geq 1 \text{ if } (i,j) \in E$
	Relaxation	$l_{\text{MVC}}(\theta; G) \triangleq \sum_{1 \leq i \leq n} \bar{X}_i + \beta \sum_{(i,j) \in E} (1 - \bar{X}_i)(1 - \bar{X}_j)$
MIS	Discrete Obj.	$\max_X \sum_{1 \leq i \leq n} X_i \quad \text{s.t.} \quad X_i X_j = 0 \text{ if } (i,j) \in E$
	Relaxation	$l_{\text{MIS}}(\theta; G) \triangleq -\sum_{1 \leq i \leq n} \bar{X}_i + \beta \sum_{(i,j) \in E} \bar{X}_i \bar{X}_j$

Figure 3: Performance v.s. hyper-parameter ρ of the RB model



are large. So, it is reasonable to view the problem over each instance as a separate but relevant task. Meta learning has shown good generalization when data distributions shift across tasks, which has empirical evidence in CV and NLP applications (Jeong & Kim, 2020; Conklin et al., 2021).

As a summary, we provide a comparison between different unsupervised frameworks to solve CO problems in Table 1. Note that PI-GNN (Schuetz et al., 2022) is directly fine-tuned on each test instance without training so the fine-tuning time is long. Also, although PI-GNN also pursues instance-wise good solutions, its performance could be bad because it does not learn from training instances. The instance-wise solutions could be just bad local minima.

5 EXPERIMENTS

We study three CO problems, namely *max clique (MC)* to find the largest set of nodes where each pair of nodes are connected, *minimum vertex covering (MVC)* to find the smallest set of nodes that every edge is connected to at least one nodes in the set, and *max independent set (MIS)* to find the largest set where any two vertices in the set are not adjacent. Their objectives (Eq. 1) and relaxations (Eq. 2) are listed in Table 2. For the detailed derivation, see Appendix C.

5.1 SETTINGS

Datasets: We conduct experiments on the MC, MVC problems over three real datasets Twitter (Leskovec & Krevl, 2014), COLLAB and IMDB (Yanardag & Vishwanathan, 2015) and two synthetic datasets with 200 and 500 nodes generated by the RB model (Xu, 2007). We name them RB200 and RB500, respectively. We make RB200 and RB500 extremely hard by setting a small hyper-parameter $\rho = 0.25$ of the RB model (Xu, 2007). The difficulty-ρ relationship on the MVC problem with 500 vertices is shown in Fig. 3, where the models are pre-trained on the RB graphs with uniformly sampled $\rho \in [0.3, 1.0]$ and tested on different RB graphs generated with single ρ's. We keep all the other hyper-parameters the same. As ρ increases, Meta-EGN and Gurobi9.5 all tend to achieve better performance. Meta-EGN could outperform Gurobi9.5 in hard instances $\rho \in (0, 0.55]$ while remains a gap on the easy ones. To verify performances for data-scale generalization, we also generate large-graph datasets RB1000, RB2000 and RB5000 with $\rho = 0.25$. As for the MIS problem, random-regular graphs (RRGs) are often used as benchmarks because they are challenging. Our experiments also use RRGs by following the settings of (Schuetz et al., 2022) with the node number ranging from 10^2 to 10^5 and the node degree selected from the set $\mathcal{D} = \{3, 7, 10, 20\}$. Here, node degree equaling 20 is the hardest setting (Angelini & Ricci-Tersenghi, 2022). A summary of these datasets are in Table. 10 in Appendix.

Data Splitting & The Evaluation Metric: For the real datasets, training/validation/test instances are randomly divided with the ratio of 8:1:1; For RB200 and RB500, $2000/100/100$ graphs are generated for training/validation/test instances; For RB1000, RB2000, RB5000, we generate 100

Table 3: ApR (time: second/graph) on the MC problem. ApR is the larger the better. 'report' denotes the reported performance in Karalias & Loukas (2020), 're-impl' denotes re-implementation, 'f-t' stands for fine-tune. Pareto-optimal results are in bold.

	Twitter	COLLAB	IMDB	RB 200	RB 500
EGN (report)	0.924 ± 0.133 (0.17)	0.982 ± 0.063 (0.10)	1.000 (0.08)	-	-
EGN (re-impl)	0.926 ± 0.113(0.17)	0.982 ± 0.069 (0.10)	1.000 (0.08)	0.820 ± 0.188 (0.26)	0.829 ± 0.192 (0.29)
EGN (re-impl) f-t	0.949 ± 0.102(0.49)	0.986 ± 0.060(0.27)	1.000 (0.20)	0.846 ± 0.180 (0.81)	0.864 ± 0.181 (0.89)
RUN-CSP	0.909 ± 0.145 (0.21)	0.912 ± 0.188 (0.14)	0.823 ± 0.191 (0.11)	0.858 ± 0.731 (2.05)	0.748 ± 0.689 (2.16)
Meta-EGN	**0.976 ± 0.048(0.17)**	0.988 ± 0.059 (0.10)	1.000 (0.08)	**0.834 ± 0.178 (0.26)**	**0.834 ± 0.198 (0.29)**
Meta-EGN f-t	0.990 ± 0.028(0.49)	0.993 ± 0.038 (0.27)	1.000 (0.20)	**0.874 ± 0.169 (0.81)**	**0.878 ± 0.181 (0.89)**
Toenshoff-Greedy	**0.917 ± 0.126 (0.08)**	0.969 ± 0.087 (0.06)	0.987 ± 0.050 (1e-3)	0.786 ± 0.195 (2.25)	0.793 ± 0.202 (2.38)
Gurobi9.5 ($\leq$0.20s)	0.737 ± 0.267 (0.17)	**0.871 ± 0.242 (0.04)**	**1.000 (1e-3)**	-	-
Gurobi9.5 ($\leq$1.00s)	**1.000 (0.37)**	0.979 ± 0.117 (0.06)	1.000 (1e-3)	-	-
Gurobi9.5 ($\leq$2.50s)	1.000 (0.37)	0.997 ± 0.036 (0.06)	1.000 (1e-3)	0.667 ± 0.188 (2.10)	0.663 ± 0.188 (2.41)
Gurobi9.5 ($\leq$4.00s)	1.000 (0.37)	**1.000 (0.06)**	1.000 (1e-3)	0.755 ± 0.225 (3.96)	0.742 ± 0.213 (3.88)

Table 4: ApR (time: second/graph) on the MVC problem. ApR is the smaller the better.. 'f-t' stands for one-step fine-tune. Pareto-optimal results are in bold.

	Twitter	COLLAB	IMDB	RB 200	RB 500
EGN	1.033 ± 0.023(0.29)	1.013 ± 0.022 (0.15)	1.000 (0.08)	1.031 ± 0.004 (0.26)	1.021 ± 0.002 (0.48)
EGN f-t	1.028 ± 0.021(0.80)	1.008 ± 0.015 (0.38)	1.000 (0.32)	1.030 ± 0.005 (0.80)	1.021 ± 0.002 (1.59)
RUN-CSP	1.180 ± 0.435 (0.16)	1.208 ± 0.198 (0.19)	1.188 ± 0.178 (0.08)	1.124 ± 0.001 (0.28)	1.062 ± 0.005 (1.65)
Meta-EGN	1.019 ± 0.017(0.29)	1.003 ± 0.010 (0.15)	1.000 (0.08)	**1.028 ± 0.005 (0.26)**	**1.016 ± 0.002 (0.48)**
Meta-EGN f-t	1.017 ± 0.017(0.80)	1.002 ± 0.010 (0.38)	1.000 (0.32)	1.027 ± 0.006 (0.80)	1.016 ± 0.002 (1.59)
Greedy	1.014 ± 0.014 (1.95)	1.209 ± 0.198 (1.79)	1.180 ± 0.077 (0.02)	1.124 ± 0.002 (5.02)	1.062 ± 0.005 (15.59)
Gurobi9.5 ($\leq$0.25s)	**1.028 ± 0.054 (0.09)**	1.002 ± 0.010 (0.10)	**1.000 (0.01)**	-	-
Gurobi9.5 ($\leq$0.50s)	1+1e-3 ± 0.001 (0.13)	**1.000 (0.10)**	1.000 (0.01)	-	-
Gurobi9.5 ($\leq$1.00s)	**1.000 (0.13)**	1.000 (0.10)	1.000 (0.01)	**1.011 ± 0.003 (0.63)**	1.019 ± 0.003 (0.69)
Gurobi9.5 ($\leq$2.00s)	1.000 (0.13)	1.000 (0.10)	1.000 (0.01)	**1.008 ± 0.002 (1.16)**	1.019 ± 0.003 (1.68)

test instances. As to RRG datasets, the training set contains 3000 RRGs, of which each has 1000 nodes and the node degree is uniformly sampled from $\mathcal{D}$. We generate 30/20 graphs for each node degree configuration in $\mathcal{D}$ for validation/test. Our evaluation metric uses approximation rate (ApR). All results are summarized based on 5 times independent experiments with different random seeds.

Baselines: Our baselines include unsupervised learning methods, heuristics and traditional CO solvers. For the MC and MVC problems, we take our direct baseline EGN (Karalias & Loukas, 2020), and also take RUN-CSP (Toenshoff et al., 2021) as another baseline. We do not consider other learning-based methods because they generally perform worse than EGN (Karalias & Loukas, 2020). As to the heuristics, we use the greedy algorithms as the heuristic baselines. For traditional CO solvers, we compare against the best commercial CO problem solver Gurobi9.5 (Gurobi Optimization, 2022) via converting the problems into integer programming form. We track the time t that the models use from the start of inferring to the end of rounding to output feasible solutions. We set this time t as the time budget of Gurobi9.5 for purely solving the integer programming, and list the actual time usage of Gurobi9.5 which includes pre-processing plus t. As to the MIS problem, we take PI-GNN (Schuetz et al., 2022) and EGN Karalias & Loukas (2020) as the learning-based baselines. We take the random greedy algorithm (RGA) and degree-based greedy algorithm (DGA) as introduced in Angelini & Ricci-Tersenghi (2019) as the heuristic baselines. When we consider fine-tuning EGN and Meta-EGN over a test instance, we use 1-step gradient descent as fine-tuning.

Implementation: For the MC and MVC problems, we use 4-layer GIN (Xu et al., 2018) as the backbone network for both meta-EGN and EGN. We use 1e-3 as both the outer learning rate (γ) of Meta-EGN and the learning rate of EGN. Here, the backbone and the learning rate are same as those in (Karalias & Loukas, 2020). For the MIS problem, we use 6-layer GIN. The outer learning rate (γ) of Meta-EGN and the learning rate of EGN are set as 1e-4. The inner learning rate (α) of Meta-EGN is always set as 5e-5. We run all experiments by using a Xeon(R) Gold 6248 CPU with 26 threads and a Quadro RTX 6000 GPU. All codes run on the PyTorch platform (Paszke et al., 2019). For more details, see Appendix. C.

Overcoming the limited expressive power of GNNs: GNNs are known with limited expressive power (Xu et al., 2018; Morris et al., 2019). Specifically over RRGs, the GIN backbone will associate each node with the same representation, unless node representations are initialized not equal. To keep fair comparison, for the MC and MVC problem, we follow Karalias & Loukas (2020) and adopt the initialization based on a single random node seed (one selected node is initialized as 1,

Table 5: Scale generalization performance on the MC and MVC problems. ApR is the larger the better for MC while the smaller the better for MVC. All the models are trained on RB500 training data. 'Fast/Medium/Accurate' denotes GNNs (without fine-tuning) using $1/4/8$ random single node seed(s) per testing instance. 'Fine-tuning' use 1-step Fine-tuning the best trial among the 8 node seed(s). 'Gap' represents the averaged gap defined as $c \times$ (# of nodes in the optimal solution - # of nodes by the given method) where $c = 1$ for MC and $c = -1$ for MVC. 'Rank' is the averaged ranks of solutions among the three methods. Optimal solutions are generated via Gurobi9.5 with the time limit 3000 seconds. Approximation rate for MC larger than 1, highlighted by *, indicates the model outperforms Gurobi9.5 solver with 3000s time budget. Pareto-optimal results are in bold.

	Dataset	Method	Fast (1)			Medium (4)			Accurate (8)			Fine-tune		
			ApR(s/g)	Gap	Rank	ApR(s/g)	Gap	Rank	ApR(s/g)	Gap	Rank	ApR(s/g)	Gap	Rank
MC	RB1000	EGN	0.6462±0.282(0.05)	11.48	2.406	0.8433±0.229(0.17)	6.47	2.237	0.9099±0.205(0.33)	4.86	2.025	0.9631±0.186(0.98)	4.13	1.693
		Meta-EGN	0.7692±0.276(0.05)	8.57	1.943	**0.9388±0.196(0.17)**	**4.61**	**1.543**	**0.9408±0.205(0.33)**	**4.97**	**1.581**	**0.9745±0.195(0.98)**	**4.01**	**1.625**
		Gurobi9.5	**0.8851±0.197(6.11)**	**5.08**	**1.650**	0.8851±0.197(6.18)	5.08	2.218	0.8851±0.197(6.48)	5.08	2.393	0.8851±0.197(7.01)	5.08	2.681
	RB2000	EGN	0.6793±0.290(0.10)	12.38	2.408	0.8968±0.184(0.29)	5.23	2.136	0.9454±0.160(0.58)	3.81	2.208	0.9714±0.154(2.03)	3.61	1.983
		Meta-EGN	0.8077±0.114(0.10)	8.13	1.991	**0.9783±0.157(0.29)**	**3.48**	**1.591**	**0.9958±0.146(0.58)**	**1.28**	**1.466**	**1.0112±0.134(2.03)***	**0.55**	**1.483**
		Gurobi9.5	**0.9510±0.145(24.14)**	**3.01**	**1.600**	0.9510±0.145(24.56)	3.01	2.091	0.9510±0.145(25.01)	3.01	2.325	0.9510±0.145(25.66)	3.01	2.533
	RB5000	EGN	0.9603±0.159(0.33)	2.42	2.130	1.0203±0.139(1.02)*	-1.26	2.060	1.0272±0.140(2.50)*	-1.68	1.980	1.0475±0.188(9.66)*	-2.86	1.970
		Meta-EGN	**1.0288±0.138(0.33)***	**-1.62**	**1.820**	**1.0684±0.233(1.02)***	**-4.02**	**1.820**	**1.0727±0.234(2.50)***	**-4.42**	**1.790**	**1.0778±0.233(9.66)***	**-4.72**	**1.710**
		Gurobi9.5	1.0000(201.55)	0.00	2.050	1.0000(202.36)	0.00	2.120	1.0000(205.64)	0.00	2.230	1.0000(214.35)	0.00	2.320
		Gurobi9.5	1.0000(3000)	0.00	-	1.0000(3000)	0.00	-	1.0000(3000)	0.00	-	1.0000(3000)	0.00	-
MVC	RB1000	EGN	1.0161±0.0048(0.20)	16.46	2.250	1.0135±0.0013(0.72)	13.73	1.920	1.0138±0.0013(1.37)	13.29	1.860	1.0138±0.0013(3.05)	13.28	1.960
		Meta-EGN	1.0145±0.0016(0.20)	14.81	1.935	**0.0131±0.0012(0.72)**	**13.40**	**1.700**	**1.0125±0.0012(1.37)**	**12.75**	**1.545**	**1.0124±0.0012(3.05)**	**12.69**	**1.455**
		Gurobi9.5	**1.0143±0.0018(1.92)**	**14.58**	**1.835**	1.0143±0.0018(2.58)	14.58	2.380	1.0143±0.0018(3.08)	14.58	2.595	1.0143±0.0018(4.96)	14.58	2.585
	RB2000	EGN	1.0114±0.0026(0.34)	22.02	2.350	1.0096±0.0008(1.32)	18.57	1.765	1.0094±0.0007(2.69)	18.17	1.765	1.0093±0.0007(6.27)	17.98	1.890
		Meta-EGN	**0.0103±0.0015(0.34)**	**19.94**	**1.740**	**1.0095±0.0008(1.32)**	**18.41**	**1.635**	**1.0092±0.0007(2.69)**	**17.82**	**1.510**	**1.0090±0.0006(6.27)**	**17.38**	**1.360**
		Gurobi9.5	1.0104±0.0010(5.63)	20.18	2.910	1.0104±0.0010(6.65)	20.18	2.600	1.0104±0.0010(8.04)	20.18	2.725	1.0104±0.0010(13.24)	20.18	2.750
	RB5000	EGN	1.0071±0.0014(1.01)	34.19	2.170	1.0064±0.0004(3.99)	30.83	1.985	1.0062±0.0004(7.95)	29.87	1.865	1.0062±0.0004(18.41)	29.68	1.960
		Meta-EGN	**1.0067±0.0005(1.01)**	**32.51**	**2.045**	**1.0062±0.0005(3.99)**	**29.96**	**1.600**	**1.0061±0.0004(7.95)**	**29.44**	**1.555**	**1.0060±0.0003(18.41)**	**29.15**	**1.470**
		Gurobi9.5	1.0066±0.0006(24.60)	31.88	1.785	1.0066±0.0006(28.72)	31.88	2.415	1.0066±0.0006(32.16)	31.88	2.580	1.0066±0.0006(42.62)	31.88	2.570

Table 6: Generalization performance from Twitter to RB2000 on the MC and MVC problems. Pareto-optimal results are in bold.

Method	MC (Approximation Rate ↑ (time))				MVC (Approximation Rate ↓ (time))			
	Fast (1)	Medium (4)	Accurate (8)	Fine-tune	Fast (1)	Medium (4)	Accurate (8)	Fine-tune
EGN	0.594±0.210(0.07)	0.788±0.201(0.16)	0.819±0.195(0.29)	0.831±0.192(0.89)	1.055±0.005(0.11)	1.053±0.004(0.37)	1.052±0.004(0.48)	1.050±0.004(1.59)
Meta-EGN	**0.690±0.201(0.07)**	**0.793±0.197(0.16)**	**0.833±0.193(0.29)**	**0.876±0.182(0.89)**	1.036±0.005(0.11)	1.030±0.003(0.37)	1.029±0.002(0.48)	1.021±0.003(1.59)
Gurobi9.5	0.663±0.188(2.92)	0.663±0.188(2.92)	0.669±0.191(3.08)	0.742±0.213(3.88)	**1.019±0.003(1.12)**	**1.019±0.003(1.30)**	**1.019±0.003(1.35)**	**1.017±0.002(2.40)**

others as 0). We use 8 single random node seeds for EGN and Meta-EGN in the experiments of Sec. 5.2 and report the best among the 8 trials. We try different numbers of random node seeds in the experiments of Sec. 5.3. For the large-scale MIS problem studied in Sec. 5.4, we find such single node initialization is too local to generate valid global solutions. So, we adopt initialization based on the solutions of greedy algorithms DGA (for Figs. 1,2) and RGA (for Fig. 4). Then, EGN and Meta-EGN can be viewed as to learn heuristics to improve the greedy solutions. Note that learning heuristics to tune these solutions is non-trivial (Andrade et al., 2012; Rahman & Virag, 2017).

5.2 META-EGN BOOSTS THE PERFORMANCE WITHOUT DISTRIBUTION SHIFTS

We first compare the performances of different methods when the datasets used for training and test are from the same distribution. Table 3 and Table 4 show the results for the MC problem and the MVC problem respectively. In both problems and across the five datasets, Meta-EGN siginificantly outperforms EGN and RUN-CSP, both before and after the fine-tuning step. In comparison with the traditional CO solvers, Meta-EGN narrows the gap from Gurobi9.5 on those real small graphs. For RB graphs, Meta-EGN outperforms Gurobi9.5 for the MC problem. For the MVC problem, Meta-EGN outperforms Gurobi9.5 on RB500.

We notice that both EGN and Meta-EGN perform generally well on the MC problem while not as competitive on the MVC problem. This results from the initialization of GNN inputs. The MC problem outputs clusters that are more local while MVC asks for global assignments, which makes such single-seed-based initialization less fit for the MVC problem.

5.3 META-EGN BOOSTS THE PERFORMANCE WITH DISTRIBUTION SHIFTS

Problem Scale Shift: Here, we use large-scale RB graphs of 1000-5000 nodes to test EGN and Meta-EGN that are trained based on RB500. Table 5 shows the results. Both methods show good generalization while Meta-EGN is always better. As the scale increases, Meta-EGN outperforms Gurobi9.5. For example, it takes Meta-EGN with 4 random initializations only 1.02s to beat Gurobi9.5 that runs for 3000 seconds on RB5000 dataset in the MC problem. Moreover, Meta-EGN can even outperform Gurobi9.5 on the MVC problem when the problem scale be comes large.

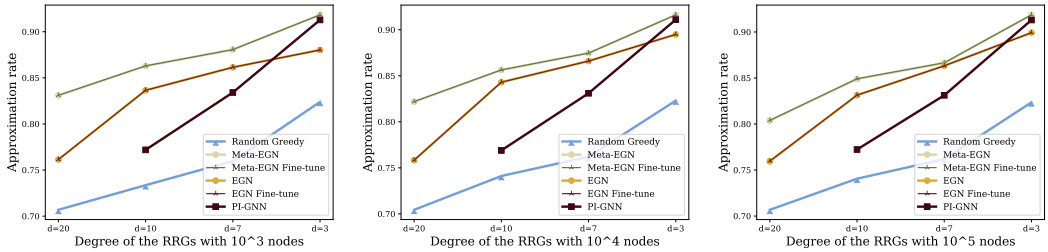


Figure 4: ApRs in the MIS problem on RRGs. Meta-EGN and EGN are both trained with the output of Random Greedy Algorithm (RGA) as initialization.

Real-Synthetic Distribution Shift: Here, we train EGN and Meta-EGN on Twitter and test them on RB500. Table 6 shows the results. Compare Table 6 with Tables 3,4. We observe better generalization performance of Meta-EGN compared to EGN. For example, for the MC problem, Meta-EGN has almost the same performance whether there is a dataset shift or not (0.833 v.s. 0.834 before fine-tuning, 0.876 v.s. 0.878 after fine-tuning) while EGN has a bigger gap in performance when there is a shift (0.819 v.s. 0.829 before fine-tuning, 0.831 v.s. 0.864 after fine-tuning). For the MVC problem, although the performance drop of Meta-EGN is larger, such a drop is still much smaller than that of EGN.

5.4 MAX INDEPENDENT SET: A RESPONSE TO (ANGELINI & RICCI-TERSENGHI, 2022)

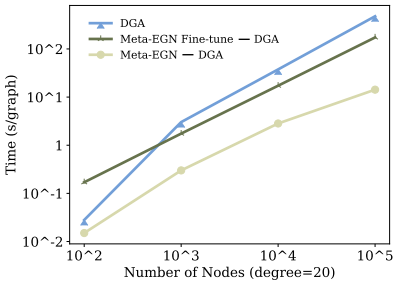


Figure 5: Time cost v.s. Graph Scales.

For the MIS problem on large-scale RRGs, Angelini & Ricci-Tersenghi (2022) have recently posted a concern on learning-based methods by arguing that PI-GNN in Schuetz et al. (2022) could not achieve comparable results with the heuristic algorithm DGA (Angelini & Ricci-Tersenghi, 2019). We see the reason comes from an improper usage of learning-based methods in Schuetz et al. (2022) as stated in Sec. 1: 1) PI-GNN does not learn from history; 2) GNN generally suffers from a node ambiguity issue on RRGs. To resolve the problem, we utilize the outputs of DGA and RGA as the initialization of GNN inputs (EGN, Meta-EGN, PI-GNN) and expect to learn heuristics to further tune the solutions given by the greedy algorithms. We train GNN models on RRGs with 1000 nodes with node degrees randomly sampled from $3, 7, 10, 20$, and test on larger RRGs (up-to 10^5 nodes). Experiments show that Meta-EGN can further improve DGA (in Fig. 1) and RGA (in Fig. 4), while EGN and PI-GNN fail to better tune DGA. Note that here EGN and Meta-EGN adopt the exactly same backbones. We attribute the improvement to meta-learning-based training as adopted by Meta-EGN. See Table 7 in Appendix for more details of the numerical improvement by Meta-EGN. We also see in these cases, one-step fine-tuning does not contribute much to the performance of EGN or Meta-EGN, indicating the model before fine-tuning has been very close to a local minimum.

We also check the extra time cost by running Meta-EGN to improve DGA solutions in Fig. 5 and Fig. 6 in Appendix. The extra time cost is just 1% (without fine-tuning) - 30% (with fine-tuning) of the time cost of DGA. In theory, the extra time cost without fine-tuning should be $O(|E|)$ for GNN inference plus $O(|V|)$ for rounding, which is in the same order of DGA, while the GNN parallel inference substantially reduces the time.

6 CONCLUSION

This work proposes an unsupervised learning framework Meta-EGN with the goal of optimizing NNs towards instance-wise good solutions to CO problems. Meta-EGN leverages MAML to achieve the goal. Meta-EGN views each training instance as a separate task and learns a good initialization for all these tasks. Meta-EGN significantly improves the performance of its baseline and has shown good generalization when the data used for training and test has different scales or distributions. In addition, Meta-EGN can learn to improve the greedy heuristics while paying almost no extra time cost in the problem of maximum independent set on large-scale random regular graphs.

REFERENCES

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *International Conference on Learning Representations*, 2018.

Diogo V Andrade, Mauricio GC Resende, and Renato F Werneck. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, 2012.

Maria Chiara Angelini and Federico Ricci-Tersenghi. Monte carlo algorithms are very effective in finding the largest independent set in sparse random graphs. *Physical Review E*, 100(1):013302, 2019.

Maria Chiara Angelini and Federico Ricci-Tersenghi. Cracking nuts with a sledgehammer: when modern graph neural networks do worse than classical greedy algorithms. *arXiv preprint arXiv:2206.13211*, 2022.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Henry Conklin, Bailin Wang, Kenny Smith, and Ivan Titov. Meta-learning to compositionally generalize. In *ACL/IJCNLP (1)*, 2021.

Yves Crama. Combinatorial optimization models for production scheduling in automated manufacturing systems. *European Journal of Operational Research*, 99(1):136–153, 1997.

Arthur Delarue, Ross Anderson, and Christian Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. *Advances in Neural Information Processing Systems*, 33, 2020.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2022.

John J Hopfield and David W Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. In *International Conference on Learning Representations*, 2018.

Benjamin Hudson, Qingbiao Li, Matthew Malencia, and Amanda Prorok. Graph neural network guided local search for the traveling salesperson problem. *arXiv preprint arXiv:2110.05291*, 2021.

Taewon Jeong and Heeyoung Kim. Ood-maml: Meta-learning for few-shot out-of-distribution detection and classification. *Advances in Neural Information Processing Systems*, 33:3907–3916, 2020.

Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33: 6659–6672, 2020.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, and Youngjune Gwon. Matrix encoding networks for neural combinatorial optimization. *Advances in Neural Information Processing Systems*, 34, 2021.

Jure Leskovec and Andrej Krevl. Snap datasets: Stanford large network dataset collection, 2014.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Marc Mezard and Andrea Montanari. *Information, physics, and computation*. Oxford University Press, 2009.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 4602–4609, 2019.

Yatin Nandwani, Deepanshu Jindal, Parag Singla, et al. Neural learning of one-of-many solutions for combinatorial problems in structured output spaces. In *International Conference on Learning Representations*, 2021.

Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

Joseph O'rourke. *Art gallery theorems and algorithms*, volume 57. Oxford New York, 1987.

Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*. 2019.

Mustazee Rahman and Balint Virag. Local algorithms for independent sets are half-optimal. *The Annals of Probability*, 45(3):1543–1577, 2017.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32, 2019.

Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

Kate A Smith. Neural networks for combinatorial optimization: a review of more than a decade of research. *Informs journal on Computing*, 11(1):15–34, 1999.

Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Run-csp: unsupervised learning of message passing networks for binary constraint satisfaction problems. 2019.

Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3:580607, 2021.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

Haoyu Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. *Advances in neural information processing systems*, 35, 2022.

K BHOSLIB Xu. Benchmarks with hidden optimum solutions for graph problems. *URL http://www. nlsde. buaa. edu. cn/kexu/benchmarks/graph-benchmarks. htm*, 2007.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

Kenichi Yabuta and Hitoshi Kitazawa. Optimum camera placement considering camera specification for security monitoring. In *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2114–2117. IEEE, 2008.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1365–1374, 2015.

Weichi Yao, Afonso S Bandeira, and Soledad Villar. Experimental performance of graph neural networks on random instances of max-cut. In *Wavelets and Sparsity XVIII*, volume 11138, pp. 242–251. SPIE, 2019.

Gal Yehuda, Moshe Gabel, and Assaf Schuster. It's not what machines can learn, it's what we cannot teach. In *International conference on machine learning*, pp. 10831–10841. PMLR, 2020.

Emre Yolcu and Barnabás Póczos. Learning local search heuristics for boolean satisfiability. *Advances in Neural Information Processing Systems*, 32, 2019.

A PROOF OF THEOREM 1

Our proof is divided into the following two parts. In part 1, we first prove that if $l(\theta; G) < \beta + \triangle$ (even if $l(\theta; G) \geq \beta$), there exists α such that after the one-step finetuning, the loss value $l(\theta'; G)$ of Meta-EGN satisfies $l(\theta'; G). \leq l(\theta; G) - \triangle$, where $\triangle = \|\nabla_\theta l(\theta; G)\|\epsilon - \frac{L}{2}\epsilon^2$ if $\epsilon < \frac{1}{L}\|\nabla_\theta l(\theta; G)\|$ or $\triangle = \frac{1}{2L}\|\nabla_\theta l(\theta; G)\|^2$ o.w..

In part 2, we prove that once Meta-EGN achieves the loss value $l(\theta'; G)$ after the one-step finetuning, the rounding process would output a feasible X whose objective satisfies $f(X; G) \leq l(\theta'; G)$.

Part 1: By Taylor Expansion, we could get

$$\begin{aligned} l(\theta'; G) &= l(\theta) + \nabla_\theta l(\theta; G)(\theta' - \theta) + \frac{1}{2}(\theta' - \theta)^T \nabla_\theta^2 l(\theta; G)(\theta' - \theta) \\ &\overset{(a)}{\leq} l(\theta; G) + \nabla_\theta l(\theta)(\theta' - \theta) + \frac{1}{2}L\|\theta' - \theta\|_2^2 \\ &\overset{(b)}{=} l(\theta; G) + \frac{1}{2}L\|\theta' - \theta\|_2^2 - \alpha\|\nabla_\theta l(\theta; G)\|_2^2 \\ &= l(\theta; G) + (\frac{L\alpha^2}{2} - \alpha)\|\nabla_\theta l(\theta; G)\|_2^2, \end{aligned} \tag{6}$$

where (a) is due to the local L-smoothness of $l(\cdot; G)$ such that $\nabla_\theta^2 l(\theta; G) \preceq LI$, (b) is due to the definition of one-step finetuning $\theta' = \theta - \alpha\nabla_\theta l(\theta; G)$.

Let $\alpha = \frac{1}{L}$ in equation (6), we have:

$$l(\theta'; G) \leq l(\theta; G) - \frac{\|\nabla_\theta l(\theta; G)\|_2^2}{2L} \tag{7}$$

If $\epsilon < \frac{1}{L}\|\nabla_\theta l(\theta; G)\|$:

Let $\triangle = \|\nabla_\theta l(\theta; G)\|\epsilon - \frac{L}{2}\epsilon^2$, we have:

$$\min_\epsilon -\triangle = \min_\epsilon \frac{L}{2}\epsilon^2 - \|\nabla_\theta l(\theta; G)\|\epsilon = -\frac{\|\nabla_\theta l(\theta; G)\|^2}{2L}, \tag{8}$$

thus

$$l(\theta'; G) \leq l(\theta; G) - \triangle. \tag{9}$$

If $\epsilon \geq \frac{1}{L}\|\nabla_\theta l(\theta; G)\|$:

Let $\triangle = \frac{1}{2L}\|\nabla_\theta l(\theta; G)\|^2$, we would directly have:

$$l(\theta'; G) \leq l(\theta; G) - \triangle. \tag{10}$$

Part 2: The proof in this part follows the rounding analysis in Wang et al. (2022). Consider the rounding procedure from continuous space $\bar{X} = \mathcal{A}_\theta(G), \bar{X} \in [0,1]^n$ into the discrete feasible solution $X \in \{0,1\}^n$. Let $\bar{X}_i, X_i, i = \{0, 1, ..., n\}$ denote their entries. W.l.o.g, suppose the rounding order is from 1 to n and we have finished the rounding before the t-th node, we now analyze the rounding of t-th node:

$$\begin{aligned}
&f_r([X_1, ..., X_{t-1}, \bar{X}_t, \bar{X}_{t+1}, ..., \bar{X}_n]; G) + \beta g_r([X_1, ..., X_{t-1}, \bar{X}_t, \bar{X}_{t+1}, ..., \bar{X}_n]; G) \\
\overset{(d)}{\geq} &\bar{X}_t(f_r([X_1, ..., X_{t-1}, 1, \bar{X}_{t+1}, ...\bar{X}_n]; G) + \beta g_r([X_1, ..., X_{t-1}, 1, \bar{X}_{t+1}, ..., \bar{X}_n]; G)) \\
&+ (1 - \bar{X}_t)(f_r([X_1, ..., X_{t-1}, 0, \bar{X}_{t+1}, ..., \bar{X}_n]; G) + \beta g_r([X_1, ..., X_{t-1}, 0, \bar{X}_{t+1}, ..., \bar{X}_n]; G)) \\
\geq &\bar{X}_t(\min_{j_t=\{0,1\}} f_r([X_1, ..., X_{t-1}, j_t, \bar{X}_{t+1}, ..., \bar{X}_n]; G) + \beta g_r([X_1, ..., X_{t-1}, j_t, \bar{X}_{t+1}, ..., \bar{X}_n]; G)) \\
&+ (1 - \bar{X}_t)(\min_{j_t=\{0,1\}} f_r([X_1, ..., X_{t-1}, j_t, \bar{X}_{t+1}, ..., \bar{X}_n]; G) \\
&+ \beta g_r([X_1, ..., X_{t-1}, j_t, \bar{X}_{t+1}, ..., \bar{X}_n]; G)) \\
\overset{(e)}{=} &f_r([X_1, ..., X_{t-1}, X_t, \bar{X}_t, ..., \bar{X}_n]; G) + \beta g_r([X_1, ..., X_{t-1}, X_t, \bar{X}_t, ..., \bar{X}_n]; G)
\end{aligned} \tag{11}$$

where (d) is due to $l_r(\theta; G)$'s entry-wise concavity w.r.t $\bar{X}$ and Jensen's inequality, (e) is due to $X_t = \arg\min_{j=0,1} f_r(X_1, ..., X_{t-1}, t, \bar{X}_{t+1}, ..., \bar{X}_n) + \beta g_r(X_1, ..., X_{t-1}, t, \bar{X}_{t+1}, ..., \bar{X}_n)$ (the definition of our rounding process). The loss value is monotonically non-increasing through the whole rounding process according to the equation above, thus we could get:

$$l(\theta') \geq f(X; G) + \beta g(X; G) \tag{12}$$

B Supplementary Experiment Results

B.1 Supplementary Time cost v.s. Graph Scale in the MIS

we show the degree $3, 7, 10$ in the following Fig. 6. They show the same time-cost vs scale relation as that in Fig. 5. The extra time cost of GNN is $O(|E|)$ for inference plus $O(|V|)$ for rounding, which is in the same order of DGA.

B.2 How much does Meta-EGN modify DGA and RGA heuristics in the MIS

We display the average approximation rate improvement and the average node number increase by Meta-EGN over DGA and RGA in Table. 7.

B.3 Training the Models on Subsets of the Training Data

We display the average approximation rates of the models that are only trained on subsets of the original training data in the max clique problem on Twitter. The training dataset is randomly sampled

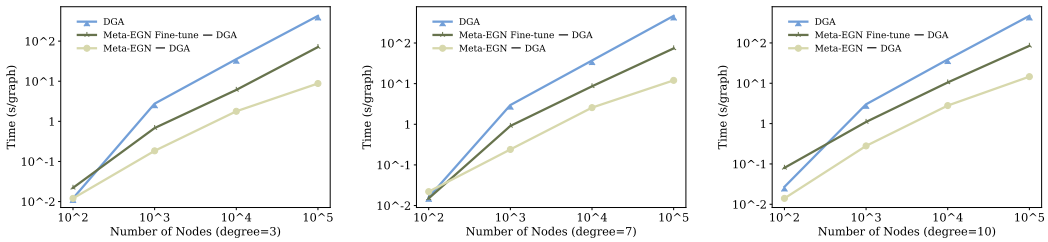


Figure 6: Time cost v.s. Graph Scales on degree $3, 7, 10$

Table 7: Improvement of Meta-EGN over DGA and RGA in the MIS on RRGs, 'Imp in ApR' denotes the average improvement in approximation rate and 'Imp in #Node' denotes the average number of nodes that Meta-EGN could find more than the heuristics.

	Scale/Degree	3		7		10		20	
		Imp in ApR	Imp in #Node	Imp in ApR	Imp in #Node	Imp in ApR	Imp in #Node	Imp in ApR	Imp in #Node
Meta-EGN improves DGA by	10^3	0.0043	1.950	0.0060	2.014	0.0044	1.254	0.0084	1.657
	10^4	0.0050	22.768	0.0062	20.811	0.0067	19.109	0.0079	15.588
	10^5	0.0032	145.718	0.0045	151.051	0.0051	145.69	0.0050	98.660
Meta-EGN improves RGA by	10^3	0.0944	42.986	0.1208	40.549	0.1292	36.849	0.1239	24.447
	10^4	0.0932	424.404	0.1125	377.628	0.1151	328.276	0.1173	231.456
	10^5	0.0871	3966.272	0.1045	3507.751	0.1083	3088.824	0.0969	1912.030

from the original training dataset and the testing dataset remains the same as that in Table. 3. Both the methods have worse performance as the number of training instances reduces, while Meta-EGN only has a 0.7% performance decrease from the full-size training dataset with 750 samples to the training subset with only 64 instances. In contrast, EGN decreases its performance by 1.7%.

Table 8: The approximation rate of the max clique problem on Twitter. Models are only trained on subsets of the dataset, 'training subset' denotes the number of instances in the training data.

training subset	64	128	256	512	Full (750)
EGN	0.909±0.122	0.911±0.118	0.914±0.118	0.922±0.115	0.926±0.113
Meta-EGN	0.970±0.058	0.973±0.055	0.975±0.055	0.975±0.051	0.976±0.048

B.4 Training the Models on RRGs with single Degrees in the MIS

We train the EGN and Meta-EGN models on RRGs with only 3 or 20 degrees and test them on RRGs with the rest degrees from $\{3, 7, 10, 20\}$. Models take the output of DGA as the initialization graph node feature. We show the performance of both the models without fine-tuning in Fig. 7. When only trained on RRGs with degree 3 (See the left two figures in Fig. 7), both the models could not generalize well, as neither of them could outperform the initialization input of DGA. Note that Meta-EGN still achieves better performance than EGN in this case. As to the models only trained on RRGs with degree 20 (See the right two figures in Fig. 7), we observe that both the model have relatively good generalization ability across different degrees, yet Meta-EGN could still marginally outperform EGN in this case. We attribute this phenomenon to the fact that solving the MIS on RRGs with degree 20 is much more complicated than those with degree 3 and thus may contain adequate heuristics for solving RRGs with lower degrees.

B.5 Comparison on the Training Time of the Models

We display the wall clock training time for the two methods to convergence in Table. 9 (from start to the the best epoch on validation set). We observe that Meta-EGN generally takes two to three times to converge compared with EGN, but their training time cost basically remain on the same order of magnitude.

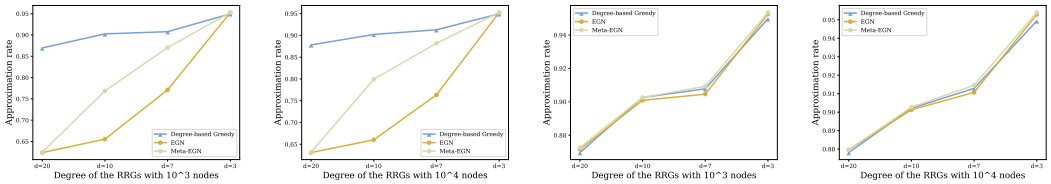


Figure 7: The left two figures show the ApRs on RRGs with 10^3 and 10^4 nodes of the models trained only on RRGs with degree 3. The right two figures show the ApRs on RRGs with 10^3 and 10^4 nodes of the models trained only on RRGs with degree 20.

Table 9: The wall clock training time to convergence of EGN and Meta-EGN in different problems.

Dataset/Time (min:second)	MC Twitter	MC RB200	MC RB500	MVC Twitter	MVC RB200	MVC RB500	MIS RRGs
EGN	46:50	104:37	282:57	100:58	83:27	128:39	733:02
Meta-EGN	101:55	210:04	609:47	276:38	168:25	282:15	1088:55

C Supplementary Implementation Details

C.1 Experiment Details

All the codes run on the PyTorch platform 1.9.0 (Paszke et al., 2019) and PyTorch Geometric framework 1.7.2 (Fey & Lenssen, 2019). The details of each datasets is shown in Table. 10, all of the real datasets are publicly available, we follow the code in (Toenshoff et al., 2021) to generate the RB model.

Table 10: The number of instances in each dataset. '20/scale/degree' means that we generate 20 testing instances for each different scale-degree pair. We generate RB1000, RB2000 and RB5000 only for testing.

Dataset	Twitter	COLLAB	IMDB	RB200	RB500	RB1000	RB2000	RB5000	RRGs
Training	750	3600	800	2000	2000	-	-	-	3000
Validation	100	450	100	100	100	-	-	-	30
Testing	100	450	100	100	100	100	100	100	20/scale degree

To balance the training time per epoch of EGN and Meta-EGN, we define the epoch as follows: For each epoch of EGN training, the whole dataset is split into mini-batches. EGN performs standard mini-batch training along these batches and optimizes over each mini-batch. As to Meta-EGN, for each training epoch Meta-EGN only randomly samples a single batch and do the meta learning algorithm on the batch. The batch sizes of the methods are controlled the same.

C.2 Detailed Derivation of the Loss Function Relaxation

In this part, we display the detailed loss function relaxation of the three problems in our study (the MC, the MVC and the MIS). The basic idea of training loss design and relaxation follow (Karalias & Loukas, 2020; Wang et al., 2022). In the following derivation, we use i, j to represent the nodes in graphs, we use $X_i, X_j \in \{0, 1\}$ to denote the discrete assignment of the binary optimization variables, and we use $\bar{X}_i, \bar{X}_j \in [0, 1]$ to denote the relaxed soft assignment of the binary optimization variables.

The maximum clique (MC): A clique is a set of nodes $S \in V$ such that any two distinct nodes in the set are adjacent. The MC aims to find out the clique with the largest number of nodes. We could formulated the optimization objective as follows:

$$\max_{X} \sum_{1 \leq i \leq n} X_i \qquad \text{s.t.} \quad (i, j) \in E \text{ if } X_i, X_j = 1, \tag{13}$$

X_i, X_j denotes whether to take the node into the clique set ($X_i = 1$) or not ($X_i = 0$). By setting a proper penalty coefficient β, we could formulate the loss function relaxation as follows (the detailed

derivation follows the corresponding case study in Karalias & Loukas (2020)).

$$l_{\text{MC}}(\theta; G) \triangleq -(\beta+1) \sum_{(i,j)\in E} \bar{X}_i \bar{X}_j + \frac{\beta}{2} \sum_{i \neq j} \bar{X}_i \bar{X}_j. \tag{14}$$

The minimum vertex covering (MVC): A vertex cover is a set of nodes $S \in V$ that any edge in the graph is connected to at least a node from the set. The MVC aims to find out the cover set with the smallest number of nodes. The optimization objective could be summarized as follows:

$$\min_X \sum_{1 \leq i \leq n} X_i \qquad \text{s.t.} \quad X_i + X_j \geq 1 \ \text{ if } (i,j) \in E, \tag{15}$$

where X_i, X_i denotes whether to take the node into the cover set ($X_i = 1$) or not ($X_i = 0$). We design the constraint function g to represent the total number of edges that have not been covered given a set of variable assignment X, and thus we write g as:

$$g_{\text{MVC}}(X; G) \triangleq \sum_{(i,j)\in E} (1 - X_i)(1 - X_j). \tag{16}$$

Then we relax the constraint g and add it into the training objective by multiplying a proper penalty coefficient β, following the relaxation principle in Wang et al. (2022):

$$l_{\text{MVC}}(\theta; G) \triangleq \sum_{1 \leq i \leq n} \bar{X}_i + \beta \sum_{(i,j)\in E} (1 - \bar{X}_i)(1 - \bar{X}_j). \tag{17}$$

By this, we aim to minimize the value of the loss function above in order to minimize the node number of the cover set as well as consider the covering property in the constraint.

The maximum independent set (MIS): An independent set is a set of nodes where any two distinct nodes in the set are not adjacent to each other. The MIS aims to find out the independent set with the largest number of nodes. We could formulate the objective of the MIS as follows:

$$\max_X \sum_{1 \leq i \leq n} X_i \qquad \text{s.t.} \quad X_i X_j = 0 \text{ if } (i,j) \in E, \tag{18}$$

where X_i, X_j denotes whether to take the node into the independent set ($X_i = 1$) or not ($X_i = 0$). We formulate the constraint g as the total number of edges whose two connected nodes at the endpoints are both assigned into the independent set. Therefore we could write the constraint as follows:

$$g_{\text{MIS}} \triangleq \sum_{(i,j)\in E} X_i X_j. \tag{19}$$

We then relax the constraint g into continuous space and add it into the c function with a proper penalty coefficient β, following the relaxation principle in (Wang et al., 2022), and thus we could write the training loss function as:

$$l_{\text{MIS}}(\theta; G) \triangleq -\sum_{1 \leq i \leq n} \bar{X}_i + \beta \sum_{(i,j)\in E} \bar{X}_i \bar{X}_j. \tag{20}$$

By this, we aim to minimize the value of the loss function above in order to maximize the node number of the independent set as well as consider the independent property in the constraint.

C.3 Separated Algorithm Tables

We separate the algorithm table of Meta-EGN into training and testing parts to make it clearer. The algorithm table is shown in Alg. 2 for training and Alg. 3.

C.4 Implementation of the heuristics

We run all of the greedy algorithms with PyThon 3.8 in this paper. A potential method to boost the time cost of these greedy algorithm is to use c++.

Algorithm 2 Train Meta-EGN

Require: Training instances $\Xi = \{G_1, G_2, ..., G_m\}$; Hyperparameters: α, γ.

Randomly initialize $\theta^{(0)}$
for each randomly sampled mini-batch $B_j \subset \Xi$, $j = 0, 1, ..., K-1$ **do** ▷ Training starts
    For each $G_i \in B_j$, compute the adapted parameter: $\theta_i^{(j)} = \theta^{(j)} - \alpha \nabla_{\theta^{(j)}} l(\theta^{(j)}; G_i)$
    Update: $\theta^{(j+1)} \leftarrow \theta^{(j)} - \gamma \nabla_{\theta^{(j)}} \sum_{G_i \in B_j} l(\theta_i^{(j)}; G_i)$
end for
return $\theta \leftarrow \theta^{(K)}$ ▷ Training ends

Algorithm 3 Test Meta-EGN with/without Fine-tuning

Require: Testing instance G'; Hyperparameter: α; Pre-trained parameter initialization θ.

For a given testing instance G': ▷ Testing starts
if fine-tuning is allowed **then**
    Fine-tune the parameters: $\theta_{G'} \leftarrow \theta - \alpha \nabla_\theta l(\theta; G')$
    Use Def. 1 to round the relaxed solution given by $\mathcal{A}_{\theta_{G'}}(G')$ ▷ With fine-tuning
else
    Use Def. 1 to round the relaxed solution given by $\mathcal{A}_\theta(G')$ ▷ Without fine-tuning
end if
return the rounded solution ▷ Testing ends

Random Greedy Algorithm for MIS (RGA): RGA takes a time to reach a solution that is linear in the problem size n. It starts from an empty independent set S. At each step $1 \leq t \leq n$, a node i is chosen at random from the graph G_t and added to the independent set. Then all the neighbors of i are removed from G_t to formulated a new graph G_{t+1}. The process iterates until G_{t^*} is empty at step t^*, the solution is S.

Degree-based Greedy Alforithm for MIS (DGA): DGA modifies RGA by sorting the degrees of the nodes before each iteration starts, and always put the node with the smallest degree into the independent set.

Teonshoff Greedy for MC: Toenshoff et al. (2021) convert the testing instances into its complement graph, and then run DGA to solve the MIS problem. It takes the solution to the MIS problem on the complement graph as the solution for MC on the original graph.

Greedy for MVC: Greedy for MVC starts from an empty covering set S. At each step $1 \leq t \leq n$, it first sorts the degrees of the nodes in the graph G_t and always add the node i with the largest degree into the covering set S. Then all the edges that connect with i are removed from G_t to formulate a new graph G_{t+1}. The process stops until G_{t^*} is empty at step t^*, the solution is S.

D Discussion on Limitations

• As mentioned in the end of Sec. 5.1, both EGN and Meta-EGN perform generally well in MC, which outputs the cliques that are more local in comparison with the vertex covering in MVCs that require more global assignments. The random initialization seed with one node randomly set as 1 and the others as 0 would potentially limit the performance of EGN and Meta-EGN in more global CO tasks.

• We use Meta-EGN and EGN to modify the solution of DGA and RGA in the MIS problem. In addition, there are also many other Monte Carlo (MC) algorithms (i.e. simulated annealing and parallel tempering) that could produce better results than DGA or RGA in RRGs (Angelini & Ricci-Tersenghi, 2022). An intuitive idea is to test whether we could learn Meta-EGN to further fine-tune these more advanced MC algorithms in the MIS problem on RRGs.

We leave the research on modifying Meta-EGN to better deal with the CO problems that require global assignments and using Meta-EGN to improve other advanced MC algorithms as a future study.