# A Variational Approach for Combinatorial Optimization Problems on Graphs


## Abstract

Combinatorial Optimization (CO) problems are often NP-hard, thereby hindering the collection of solutions for supervised learning. Probabilistic method can provide an unsupervised framework, however, inference on it is intractable in general, and the objective is typically non-convex. Our work proposes an unsupervised method to learn a variational distribution for a graphical model, from which we can efficiently sample good solutions for CO problems. We show that our designed graphical model is guaranteed to concentrate on good solutions in both constrained and unconstrained scenarios. Additionally, the graphical model naturally introduces a temperature with which we are able to ease the learning by avoiding (encouraging) local optima during the beginning (ending) of training. We demonstrate our unsupervised learning framework on three CO problems on both synthetic and real-world graphs. The results show that our method achieves performance significantly better than other unsupervised neural methods as well as classical methods and integer solvers.


## 1 Introduction

Combinatorial Optimization (CO) problems occur whenever there is a requirement to select the best option from a finite set of alternatives. They arise in a wide range of application areas including business, medicine, and engineering (Paschos 2013). Many CO problems are NP-complete (Karp 1972; Garey and Johnson 1979). Thus, excluding the use of exact algorithms to find the optimal solution (Padberg and Rinaldi 1991; Wolsey and Nemhauser 1999), different heuristic methods are utilized to find good solutions in a reasonable amount of time (Nemhauser, Wolsey, and Fisher 1978; Dorigo, Birattari, and Stutzle 2006; Hopfield and Tank 1985; Kirkpatrick, Gelatt, and Vecchi 1983).

Recently, learning algorithms for CO problems has shown much promise, including supervised (Khalil et al. 2016; Gasse et al. 2019; Li, Chen, and Koltun 2018; Selsam et al. 2018; Nair et al. 2020), unsupervised (Karalias and Loukas 2020; Toenshoff et al. 2021), and reinforcement learning (Dai et al. 2017; Sun et al. 2020; Yolcu and Póczos 2019; Chen and Tian 2019) The success of supervised learning relies on labeled data, which could be computationally prohibitive to obtain for difficult problems (Yehuda, Gabel, and Schuster 2020). Reinforcement learning, suffering from its larger state space and lack of fully-differentiability, tends to be harder and more time consuming to train.



Unsupervised learning models the CO problem with a differentiable objective function whose minima represent discrete solutions (Hopfield and Tank 1985; Smith 1999). Naturally, this framework allows for efficient learning on large, unlabeled datasets. However, it also brings two challenges. First, in the absence of the labels, the objective function is typically non-convex (Mezard and Montanari 2009). This means that the learning can fall into local minima which results in the algorithm failing to find optimal solutions. Second, decoding good solutions from the soft assignments learned in unsupervised learning is not easy, especially when the CO problem is constrained (Toenshoff et al. 2021; Karalias and Loukas 2020).

To address those challenges, we propose a method to learn a variational distribution from which we can draw good solutions. Specifically, given a CO problem, we consider a graphical model $P \propto e^{-f(\mathbf{x})/\tau}$ such that the energy function $f$ is designed to have higher values on non-optimal or infeasible solutions, and that is guaranteed to concentrate on good solutions

We train a graph neural network (GNN) to predict the variational distribution $Q$ approximating $P$. In order to ease the learning, we employ *annealing training* and *forward iteration*. Annealing means beginning with a high temperature $\tau$ and decreasing the temperature gradually during the process (Kirkpatrick, Gelatt, and Vecchi 1983; Dowsland and Thompson 2012; Bilbro et al. 1988). The *annealing training* initially gives $P$ a smooth landscape and ultimately focuses $P$ on optimal solutions. In effect, it prevents $Q$ from falling into local minima in the early stage of learning. Although local minima are undesirable in the early stages of learning, the learned Q should be at a local optimum at the end. Hence, we apply differentiable *forward iteration* on top of the GNN to force a local optimality for $Q$. It not only introduces a prior knowledge to regulate $Q$, but also does not affect the easy end-to-end training. After obtaining $Q$, we employ a recurrent sampling strategy that sequentially decodes a solution based on the conditional distribution of the variables (Raghavan and Tompson 1987).

In our experiments, we demonstrate our unsupervised learning framework on three NP-hard graph CO problems:

minimum vertex cover (MVC), weighted max cut (MaxCut), and minimum dominating set (MDS). On both synthetic and real-world graphs, our method achieves significantly better performance compared to other unsupervised neural methods (Toenshoff et al. 2021; Karalias and Loukas 2020), classical algorithms (Aarts, Aarts, and Lenstra 2003; Bilbro et al. 1988; Wang, Chang, and Kolter 2017), and integer solvers (Achterberg 2009; Gurobi Optimization 2020). We also have an ablation study to validate our annealing training schedule and forward iteration. The results show that our variational learning framework offers a substantial improvement for unsupervised learning for CO problems on graphs.

In summary, our work has the following contributions:

- We propose a variational learning framework for generic unsupervised learning on combinatorial optimization problems.
- We exploit annealing training and forward iteration to address the non-convex problem in training and empirically justify our solutions in ablation study.
- We demonstrate the performance of our method on three NP-hard graph combinatorial problems and show that our method can achieve better performance compared to other unsupervised neural methods, classical methods, and integer solvers.

## 2 Related Work

Recently, there is a surge of interest in learning algorithms for CO problems (Bengio, Lodi, and Prouvost 2020). Supervised learning is widely used. Numerous works have combined GNNs with search procedures to solve classical CO problems, such as travelling salesman problem (Vinyals, Fortunato, and Jaitly 2015; Joshi, Laurent, and Bresson 2019; Prates et al. 2019), graph matching (Wang, Yan, and Yang 2019, 2020), quadratic assignments (Nowak et al. 2017), graph coloring (Lemos et al. 2019), and maximum independent set (Li, Chen, and Koltun 2018). Another fruitful direction is combining learning with existing solvers. For example, in the branch and bound algorithm, He, Daume III, and Eisner (2014); Khalil et al. (2016); Gasse et al. (2019); Nair et al. (2020) learn the variable selection policy by imitating the decision of oracle or rules designed by human experts. However, the success of supervised learning relies on large datasets with already solved instances, which is hard to efficiently generate in an unbiased and representative manner (Yehuda, Gabel, and Schuster 2020),

Many works, therefore, choose to use reinforcement learning instead. Dai et al. (2017) combines Q-learning with greedy algorithms to solve CO problems on graphs. Q-learning is also used in (Bai et al. 2020) for maximum subgraph problem. Sun et al. (2020) uses evolutionary strategy to learn variable selection in the branch and bound algorithm. Yolcu and Póczos (2019) employs REINFORCE algorithm to learn local heuristics for SAT problem. Chen and Tian (2019) uses actor-critic learning to learn a local rewriting algorithm. Despite being a promising approach that avoids using labelled data, reinforcement learning has its limitations. For example, reinforcement learning is typically sample inefficient and notoriously unstable to train, due to poor gradient estimations, correlations present in the sequence of observations, and hard explorations (Espeholt et al. 2018; Tang et al. 2017).

Works in unsupervised learning show promising results. Yao, Bandeira, and Villar (2019) train GNN for the max-cut problem by optimizing a relaxation of the cut objective, Toenshoff et al. (2021) trains RNN for maximum-SAT via maximizing the probability of its prediction. Karalias and Loukas (2020) summarize them as a probabilistic method to learn a distribution $g_\theta$, on which the objective $\min L = \mathbb{E}_{\mathbf{x}\sim g(\cdot)}[f(\mathbf{x})]$. The probabilistic method provides a good framework for unsupervised learning. However, optimizing the distribution is typically non-convex (Mezard and Montanari 2009). Differentiating through combinatorial structures is an important problem in machine learning (Paulus et al. 2020; Indelman and Hazan 2021; Pogančić et al. 2019; Paulus et al. 2021). In this work, to address this problem, we formulate the unsupervised learning as learning an annealed graphical model. Fruitful algorithms are designed for inference on graphical models, including linear relaxation (Raghavan and Tompson 1987), mean field approximation (Peterson 1993; ANDERSON 1988; Peterson and Söderberg 1989; Bilbro et al. 1988), message passing (Wainwright and Jordan 2008; Ravanbakhsh 2015), survey propagation (Braunstein, Mézard, and Zecchina 2005; Braunstein et al. 2006), and sub-problem decomposition (Vontobel and Koetter 2005; Elidan and Koller 2007; Kolmogorov and Wainwright 2012; Rouhani, Rahman, and Gogate 2020). The difference is the propagation in our method is automatically learned.

## 3 Method

In this section, we introduce our **V**ariational **G**raph **N**eural **N**etwork (VGNN) method.

### 3.1 Graphical Model

Before defining the variational distribution, we show how to transform a CO problem into a graphical model. First, let's consider an unconstrained problem:

$$\arg\min_{\mathbf{x}\in\{0,1\}^n} c(\mathbf{x}) \tag{1}$$

where $c(\cdot)$ is the objective function. We can define the probability of an assignment $\mathbf{x}$ as $P_\tau(\mathbf{x}) \propto e^{-c(\mathbf{x})/\tau}$, where $\tau > 0$ is the temperature that controls the sharpness of the distribution. Since optimal solutions have lower objective values, as $\tau$ approaches zero, $P$ gradually converges to a uniform distribution on optimal solutions.

For a constrained problem with $m$ constraints:

$$\arg\min_{\mathbf{x}\in\{0,1\}^n} c(\mathbf{x}) \quad \text{s.t. } \psi_i(\mathbf{x}) = 0, \quad i = 1, ..., m \tag{2}$$

where $\psi_i : \{0,1\}^n \to \{0,1\}$ is a membership function that maps an assignment $\mathbf{x}$ to 0 if the constraint is not violated, and to 1 otherwise. We first rewrite the constrained problem into an equivalent unconstrained form:

$$\arg\min_{\mathbf{x}\in\{0,1\}^n} f(\mathbf{x}) := c(\mathbf{x}) + \sum_{i=1}^{m} \beta_i \psi_i(\mathbf{x}) \tag{3}$$

Use $f(\mathbf{x})$ as energy function, we can define the graphical model:

$$P_\tau(\mathbf{x}) \propto e^{-f(\mathbf{x})/\tau} \tag{4}$$

For the energy function Eq. 3 and graphical model Eq. 4, we have following properties

**Property 1.** *Assume $\beta_i$ is sufficiently large and the CO problem Eq. 2 has a feasible solution. Then for any infeasible assignment $\mathbf{x}$, we can always find a $\mathbf{x}'$ feasible for Eq. 2 and $f(\mathbf{x}') < f(\mathbf{x})$.*

*Proof.* Consider we have a feasible solution $x$ and an infeasible solution $y$. By definition, there exists $i$ such that $\psi_i(y) = 1$. Hence,

$$\begin{aligned} f(x) &= c(x) + \sum_{i=1}^{m} \beta_i \psi_i(x) = c(x) \leq \frac{1}{2}\beta_j \\ &\leq c(y) + \beta_j \leq c(y) + \sum_{i=1}^{m} \beta_i \psi_i(y) \\ &= f(y) \end{aligned}$$

□

**Property 2.** *When $\tau$ decreases to zero, $P_\tau$ in Eq. 4 converges to optimal solutions for Eq. 3*

*Proof.* Consider the set of optimal solutions is $\mathcal{X}$ and its completion $\mathcal{X}^\perp = \{0,1\}^n \backslash \mathcal{X}$. WLOG, we assume the $f(x^*) = 0, \forall x^* \in \mathcal{X}$. The probability that a state at graphical model $P_\tau$ is not optimal can be characterized as a function of $\tau$:

$$F(\tau) = \frac{\sum_{x \in \mathcal{X}^\perp} e^{-f(x)/\tau}}{|\mathcal{X}| + \sum_{x \in \mathcal{X}^\perp} e^{-f(x)/\tau}}$$

As 1) $f(x) > 0, \forall x \in \mathcal{X}^\perp$ and 2) $\mathcal{X}^\perp$ is a finite set, there exists a $\delta > 0$ and $f(x) > \delta, \forall x \in \mathcal{X}^\perp$. We can see the numerator will decrease to 0 when $\tau$ goes to zero. The denominator is always larger than $|\mathcal{X}|$. Hence, we have the

$$\lim_{\tau \to 0} F(\tau) = 0$$

and this means the graphical model will converge to the optimal states as long as $\mathcal{X}$ is not empty. □

With a more sophisticated selection of penalty $\beta_i$, $P$ will have a smoother landscape. We leave the discussion about our selection of $\beta_i$ for each problem type in case study. Without causing confusion, we will abuse notation and let $f(\mathbf{x})$ denote either $c(\mathbf{x})$ or $c(\mathbf{x}) + \sum_{i=1}^{m} \beta_i \psi_i(\mathbf{x})$.

### 3.2 Variational Distribution and Annealing

Although the graphical model will focus on optimal solutions, drawing samples from it could still be difficult (Murray 2007). To address this problem, we use a simple variational distribution $Q$ to approximate $P_\tau$ by minimizing the KL-divergence between the two distributions. Unfortunately, the loss function $D_{\mathrm{KL}}(Q||P_\tau)$ is non-convex w.r.t. $Q$ in general (Mezard and Montanari 2009). The fa hindering the learning. We propose an annealing training schedule to resolve this issue. Specifically, we begin training with a high temperature $\tau$. In this case, $P_\tau$ has a smooth landscape with shallow local optima such that avoid $Q$ being trapped in local optima. Throughout the training process, we gradually decrease the temperature $\tau$. Consequently, the target distribution $P_\tau$ as well as $Q$ can gradually concentrate to the optimal solutions. When the annealing ultimately stops at a threshold, the learned $Q$ becomes a good approximation of optimal solutions.

The Anneal training is not only effective but also very simple to implement. Consider the KL-divergence between $Q$ and $P$:

$$D_{\mathrm{KL}}(Q||P_\tau) = \int Q(\mathbf{x}) \log Q(\mathbf{x}) d\mathbf{x} \tag{5}$$

$$- \int Q(\mathbf{x}) \log \frac{e^{-f(\mathbf{x})/\tau}}{\sum_{\mathbf{x} \in \{0,1\}^n} e^{-f(\mathbf{x})/\tau}} d\mathbf{x} \tag{6}$$

$$= \frac{1}{\tau} \mathbb{E}_{\mathbf{x} \sim Q(\cdot)}[f(\mathbf{x})] - H(Q) + \log \sum_{\mathbf{x} \in \{0,1\}^n} e^{-f(\mathbf{x})/\tau} \tag{7}$$

where $H(\cdot)$ represents the entropy of a distribution and the last term in Eq. 7 is constant. Assume Q is parameterized by $\theta$, our loss function is defined as:

$$L_\tau(\theta) = \mathbb{E}_{\mathbf{x} \sim Q_\theta(\cdot)}[f(\mathbf{x})] - \tau H(Q_\theta) \tag{8}$$

In this way, the annealing training schedule can be easily implemented by decreasing the weight of the entropy. In addition, when we set the temperature $\tau$ to zero, we have:

$$L_0(\theta) = \mathbb{E}_{\mathbf{x} \sim Q_\theta(\cdot)}[f(\mathbf{x})] \geq \min_{\mathbf{x} \in \{0,1\}^n} f(\mathbf{x}) \tag{9}$$

It shows $L_0$ is an upper bound on the optimal value and the equality is achieved when $Q$ is supported on optimal solutions. One thing needs to be noticed is, although we use a non-zero temperature in training, we still use zero temperature loss $L_0$ in validation for model selection. For notation, when $\theta$ is determined by parameter $\phi$ and energy function is determined by instance $I$, we also write the loss function as $L_\tau(\phi, I)$. The algorithm for annealing training is summarized in Alg. 1.

### 3.3 Parameterization and Forward Iteration

In this work, we consider the simplest form of variational distribution

$$Q_\theta(\mathbf{x}) = \prod_{i=1}^{n} \theta_i^{x_i} (1 - \theta_i)^{1 - x_i} \tag{10}$$

which means all the variables are independent Bernoulli random variables with probability $\theta_i$. Instead of letting a GNN $G_\phi$ directly predict $\theta_i$, we add additional forward iteration to force the local optimality. Specifically, though we try to avoid local optima in the early stage of training, in the end, the parameter $\theta$ should be a local optimum, thereby each entry should satisfy:

$$\theta_i = \arg\min_{\theta_i} L_\tau(\theta_{-i}, \theta_i) \tag{11}$$

where $\theta_{-i}$ represents all the entries except $\theta_i$ in $\theta$. Though the local optimality is naturally satisfied when we directly optimize $\theta$ in Eq. 8, this is not case when we train $G_\phi$ to predict $\theta$. In order to encourage the local optimality, we apply forward iterations on top of $G_\phi$. That's to say, when $G_\phi$ gives probability $p_i^{(0)}$ for each variable $x_i$, we run forward iteration for $T$ times:

$$p_i^{(t+1)} = \arg\min_{p_i} L_\tau(p_{-i}^{(t)}, p_i) \tag{12}$$

and define $\theta_i = p_i^{(T)}$. A key point here is, when the temperature $\tau > 0$, the forward iteration is differentiable and we can still train $G_\phi$ end to end. Since we use independent Bernoulli random variables to parameterize our variational distribution, the forward iteration has simple close-form expression and we elaborate it for each problem type in case study.

### 3.4 Decoding

After obtaining variational distribution $Q_\theta$, we can easily sample binary solutions from it. In this work, we adopt a conditional decoding strategy (Raghavan 1988). Specifically, consider we decode the variables in an order $\pi$, WLOG, we say $\pi = (1, 2, ..., n)$. Assume we have already fixed the value of $x_1 = \hat{x}_1, ..., x_k = \hat{x}_k$, we decode $x_{k+1}$ as:

$$\hat{x}_{k+1} = \arg\min_{x_{k+1} \in \{0,1\}} \mathbb{E}_{\mathbf{x} \sim Q(\cdot|\hat{x}_{1:k}, x_{k+1})}[f(\mathbf{x})] \tag{13}$$

that's to say, we select the value of $x_{k+1}$ such that the expectation of the loss function on conditional distribution $Q(\cdot|\hat{x}_{1:k}, x_{k+1})$ is minimized. In this way, the expectation is non-decreasing during decoding and it guarantees to find solution $\hat{\mathbf{x}}$ bounded by the expected loss on $Q$.

$$f(\hat{\mathbf{x}}) = \mathbb{E}_{\mathbf{x} \sim Q(\cdot|\hat{\mathbf{x}}_{1:n})}[f(\mathbf{x})] \leq \mathbb{E}_{\mathbf{x} \sim Q}[f(\mathbf{x})] \tag{14}$$

As the decoded solution is determined by the order, we propose two strategies to decode solutions with different variable order:

- *Deterministic*: sample one solution and the variables are ordered by decreasing confidence $f_i$, which equals to $|2\theta_i - 1|$ for variable $x_i$.
- *Stochastic*: sample multiple solutions, where the variables are ordered by decreasing confidence in the first run and randomly sampled for the remaining runs.

As the *deterministic* version only requires one run, it is able to find a good solution in milliseconds on large graphs with 1000 nodes and 4000 edges. The *stochastic* version requires more time, but in return, it is able to significantly improve the solution quality. In our parameterization, the conditional decoding in Eq. 13 is equivalent with an asynchronous forward iteration in zero temperature, thus can be implemented very efficiently. We elaborate it for each problem type in case study.

Algorithm 1: Annealing Training

**Input**: Initial temperature $\tau_0$, decay rate $\alpha$, threshold temperature $\tau_{\text{th}}$, GNN $G_\phi$, epoch $T$, batch size $b$, training data $I_{\text{train}}$, valid data $I_{\text{valid}}$
**Output**: $\phi_{\text{out}}$
Initialize $\tau \leftarrow \tau_0$, $\phi_{\text{out}} \leftarrow \phi$, $L_{\text{best}} \leftarrow L_0(\phi, I_{\text{valid}})$
**for** $t = 1, ..., T$ **do**
  **for** batch $B$ in $I_{\text{train}}$ **do**
    batch gradient $d\phi \leftarrow \frac{\partial L_\tau(\phi, B)}{\partial \phi}$
    update $\phi$ by gradient descent
  **end for**
  **if** $L' = L_0(\phi, I_{\text{valid}}) < L_{\text{best}}$ **then**
    $\phi_{\text{out}} \leftarrow \phi$, $L_{\text{bext}} \leftarrow L'$
  **end if**
  update temperature $\tau \leftarrow \max\{\alpha\tau, \tau_{\text{th}}\}$.
**end for**

## 4 Case Study

We focus on three NP-hard problems in this work: minimum vertex cover (MVC), maximum cut (Max-Cut) and minimum dominating set (MDS). Consider a graph $G = (V, E, w)$, where $V$ is the set of vertices, $E$ is the set of edges, $w : E \to \mathbb{R}_+$ is the edge weight function. Let $n = |V|$, $m = |E|$, we show energy function for these three problems and their forward iteration as well as conditional decoding. The details of the derivation and complexity analysis are given in appendix.

### 4.1 Minimum Vertex Cover

A vertex cover of a graph is a set of vertices that includes at least one endpoint of every edge of the graph. To find a minimum cover, the original objective function is $c(\mathbf{x}) = \sum_{v \in V} x_v$. Considering constraints, we set the penalty coefficient $\beta_i = 2$ and define the energy function as:

$$f(\mathbf{x}) = \sum_{v \in V} x_v + 2 \sum_{(u,v) \in E} (1 - x_u)(1 - x_v) \tag{15}$$

If an edge $e = (u_0, v_0)$ is not covered, we can always set an endpoint as 1 to reduce the objective. Hence, every local minimum is feasible and the global minimum is an optimal solution of the original problem. Given a temperature $\tau$, the close-form expression of the forward iteration is:

$$p_i^{(t+1)} = 1/(\exp([2 \sum_{j:(i,j) \in E} (1 - p_j^{(t)}) - 1]/\tau) + 1)$$

In conditional decoding, this means we decode $x_i = 0$ if $2 \sum_{j:(i,j) \in E} (1 - p_j^{(t)}) < 1$, and $x_i = 1$ if $2 \sum_{j:(i,j) \in E} (1 - p_j^{(t)}) > 1$. The complexity of this decoding is $O(m + n)$.

### 4.2 Max Cut

A cut partitions the graph's vertices into two complementary sets $S$ and $T$. We aim to find a cut such that the weighted edge sum between $S$ and $T$ is maximized. Since max-cut is unconstrained, we can directly define the energy function

Table 1: Evaluation MVC on 1000 random graphs

| Size | 256-300 | | 512-600 | | 1024-1100 | |
|---|---|---|---|---|---|---|
| Method | *ratio* | avg time (s) | *ratio* | avg time (s) | *ratio* | avg time (s) |
| Greedy | $1.0330 \pm 0.0125$ | $1.53e^{-4}$ | $1.0345 \pm 0.0086$ | $5.78e^{-4}$ | $1.0353 \pm 0.0064$ | $1.49e^{-3}$ |
| MFA | $1.0109 \pm 0.0074$ | $9.50e^{-3}$ | $1.0115 \pm 0.0054$ | $1.65e^{-2}$ | $1.0124 \pm 0.0040$ | $3.11e^{-2}$ |
| RUN-CSP | $1.0117 \pm 0.0076$ | $4.41e^{-2}$ | $1.0138 \pm 0.0059$ | $6.07e^{-2}$ | $1.0153 \pm 0.0034$ | $9.81e^{-2}$ |
| Erdos(d) | $1.0147 \pm 0.0085$ | $4.19e^{-3}$ | $1.0153 \pm 0.0059$ | $4.34e^{-3}$ | $1.0159 \pm 0.0044$ | $4.72e^{-3}$ |
| Erdos(s) | $1.0145 \pm 0.0084$ | $1.12e^{-2}$ | $1.0151 \pm 0.0058$ | $1.96e^{-2}$ | $1.0156 \pm 0.0043$ | $3.27e^{-2}$ |
| BP | $1.0604 \pm 0.0162$ | $1.60e^{-1}$ | $1.0616 \pm 0.0114$ | $8.10e^{-2}$ | $1.0630 \pm 0.0084$ | $1.2e^{-1}$ |
| PBP | $1.0675 \pm 0.0183$ | $2.05e^{-1}$ | $1.0632 \pm 0.0198$ | $1.17e^{-1}$ | $1.0592 \pm 0.0072$ | $1.57e^{-1}$ |
| VGNN(d) | $1.0087 \pm 0.0066$ | $5.89e^{-3}$ | $1.0108 \pm 0.0046$ | $6.67e^{-3}$ | $1.0099 \pm 0.0036$ | $6.90e^{-3}$ |
| VGNN(s) | $1.0052 \pm 0.0052$ | $1.26e^{-2}$ | $\mathbf{1.0063 \pm 0.0039}$ | $2.26e^{-2}$ | $\mathbf{1.0074 \pm 0.0032}$ | $3.44e^{-2}$ |
| SCIP(0.1s) | $1.0265 \pm 0.0117$ | $1.00e^{-1}$ | $1.0278 \pm 0.0080$ | $1.02e^{-1}$ | $1.1857 \pm 0.8189$ | $1.00e^{-1}$ |
| Gurobi(0.1s) | $\mathbf{1.0026 \pm 0.0068}$ | $1.38e^{-1}$ | $1.0795 \pm 0.1088$ | $1.91e^{-1}$ | $1.3261 \pm 0.0244$ | $2.32e^{-1}$ |

equals to its objective function

$$f(\mathbf{x}) = c(\mathbf{x}) = \sum_{(u,v)\in E} w(u,v)(x_u + x_v - 2x_u x_v) \quad (16)$$

Given a temperature $\tau$, the close-form expression of the forward iteration is:

$$p_i^{(t+1)} = 1/(\exp(\sum_{j:(i,j)\in E} w(i,j)(2p_j^{(t)} - 1)/\tau) + 1)$$

In conditional decoding, this means we decode $x_i = 0$ if $\sum_{j:(i,j)\in E} w(i,j)(2p_j^{(t)} - 1) > 0$, and $x_i = 1$ if $\sum_{j:(i,j)\in E} w(i,j)(2p_j^{(t)} - 1) < 0$. The complexity of decoding is $O(m+n)$.

### 4.3 Minimum Dominating Set

A dominating set $D$ is a subset of $V$ such that every vertex not in $D$ is adjacent to at least one member of $D$. To minimize the size of $D$ the original objective function is $c(\mathbf{x}) = \sum_{v\in V} x_v$. Considering constraints, we set the penalty coefficient $\beta_i = 2$ and define the energy function as:

$$f(\mathbf{x}) = \sum_{v\in V} x_v + 2\sum_{v\in V}(1 - x_v) \prod_{j:(i,j)\in E} (1 - x_j) \quad (17)$$

It can be noticed that, if a vertex $v$ is not dominated, we can set $v$ or one of its neighbor as 1 to reduce the objective. Hence, every local minimum is feasible and the global minimum is an optimal solution of the original problem. Define the augmented neighbor $N_i := \{j : j = i \text{ or } (i,j) \in E\}$ and $N_i^k := \{j : j \neq k, j = i \text{ or } (i,j) \in E\}$, given a temperature $\tau$, the close-form expression of the forward iteration is:

$$p_i^{(t+1)} = 1/(\exp((\sum_{j\in N_i} 2 \prod_{k\in N_j^i} (1 - p_k^{(t)}) - 1)/\tau) + 1)$$

In conditional decoding, this means we decode $x_i = 0$ if $\sum_{j\in N_i} 2\prod_{k\in N_j^i}(1 - p_k^{(t)}) < 1$, and $x_i = 1$ if $\sum_{j\in N_i} 2\prod_{k\in N_j^i}(1-p_k^{(t)}) > 1$. The complexity of decoding is $O(m+n)$.

## 5 Experiments

We now present comparative experiments against classical algorithms, unsupervised neural methods, and integer solvers to justify our variational method. We also give an ablation study to validate the benefits from annealing training and forward iteration.

### 5.1 Setup

**Dataset**: We focus on minimum vertex cover (MVC), weighted maximum cut (MaxCut), and minimum dominating set (MDS) in our experiments. We use both synthetic and real-world graph as our dataset. For synthetic dataset, we use Barabasi-Albert (BA) (Barabási and Albert 1999) with attachment 4 to generate random graphs in sizes {32-40, 64-75, 128-150, 256-300, 512-600, 1024-1100}. For Maxcut, we assign edge weights as uniform distribution between $[0, 1]$. For real-world dataset, we use social network *Github* (Morris et al. 2020) and *Memetracker* (Dai et al. 2017). Since *Github* is unweighted, we assign weight 1 to all edges when we solve Maxcut.

**Implementation**: We denote VGNN with *deterministic* and *stochastic* decoding strategies as VGNN(d) and VGNN(s), respectively. The details of the implementation are given in Appendix.

**Baseline**: We compare against several unsupervised neural methods, classical algorithms, and integer solvers. The unsupervised neural methods include Erdos-GNN (Karalias and Loukas 2020) and RUN-CSP (Toenshoff et al. 2021). Erdos-GNN is equivalent to a special case of our VGNN with temperature $\tau = 0$ and forward iteration $T = 0$. Erdos-GNN with *deterministic* and *stochastic* decoding strategies are denoted as Erdos(d) and Erdos(s), respectively. RUN-CSP is only capable solving MVC as it is designed for unweighted graph with binary constraints. We train it using the default setting in (Toenshoff et al. 2021) and for evaluation, we run it multiple times until a large time budget is met and return the optimal solution (Karalias and Loukas 2020). For classical algorithms, we use greedy (Greedy), belief propagation, disturbed belief propagation(Ravanbakhsh 2015), and mean filed annealing algorithms (MFA) (Bilbro et al.

Table 2: Evaluation MaxCut on 1000 random graphs

| Size | 256-300 | | 512-600 | | 1024-1100 | |
|---|---|---|---|---|---|---|
| Method | *ratio* | avg time (s) | *ratio* | avg time (s) | *ratio* | avg time (s) |
| Greedy | $1.0626 \pm 0.0136$ | $8.90e^{-4}$ | $1.0545 \pm 0.0105$ | $3.62e^{-3}$ | $1.0461 \pm 0.0077$ | $1.49e^{-2}$ |
| MFA | $1.0209 \pm 0.0088$ | $1.60e^{-2}$ | $1.0126 \pm 0.0080$ | $3.34e^{-2}$ | $\mathbf{1.0043 \pm 0.0056}$ | $6.59e^{-2}$ |
| Mixing | $1.0529 \pm 0.0150$ | $3.91e^{-2}$ | $1.0417 \pm 0.0115$ | $1.34e^{-1}$ | $1.0323 \pm 0.0085$ | $4.50e^{-1}$ |
| Erdos(d) | $1.0324 \pm 0.0104$ | $4.63e^{-3}$ | $1.0237 \pm 0.0088$ | $4.96e^{-3}$ | $1.0156 \pm 0.0061$ | $5.45e^{-3}$ |
| Erdos(s) | $1.0307 \pm 0.0100$ | $1.95e^{-2}$ | $1.0218 \pm 0.0087$ | $3.71e^{-2}$ | $1.0133 \pm 0.0059$ | $7.35e^{-2}$ |
| PBP | $1.0468 \pm 0.0123$ | $1.68e^{-1}$ | $1.0365 \pm 0.0098$ | $3.92e^{-2}$ | $1.0272 \pm 0.0068$ | $3.98e^{-1}$ |
| BP | $1.0453 \pm 0.0123$ | $1.97e^{-1}$ | $1.0359 \pm 0.0100$ | $4.35e^{-2}$ | $1.0251 \pm 0.0071$ | $4.62e^{-1}$ |
| VGNN(d) | $1.0253 \pm 0.0095$ | $5.36e^{-3}$ | $1.0164 \pm 0.0082$ | $5.76e^{-3}$ | $1.0105 \pm 0.0057$ | $6.19e^{-3}$ |
| VGNN(s) | $\mathbf{1.0159 \pm 0.0081}$ | $2.36e^{-2}$ | $\mathbf{1.0104 \pm 0.0073}$ | $4.47e^{-2}$ | $1.0048 \pm 0.0055$ | $8.68e^{-2}$ |
| SCIP(5.0s) | $1.1203 \pm 0.0152$ | $5.00e^{0}$ | $1.1170 \pm 0.0156$ | $5.00e^{0}$ | $1.1056 \pm 0.0105$ | $5.00e^{0}$ |
| Gurobi(5.0s) | $1.1052 \pm 0.0249$ | $5.07e^{0}$ | $1.1340 \pm 0.0357$ | $5.10e^{0}$ | $1.1310 \pm 0.0257$ | $5.14e^{0}$ |

Table 3: Excess ratio on real-world graphs

| Data | Github | | | Memetracker | | |
|---|---|---|---|---|---|---|
| | MVC | MaxCut | MDS | MVC | MaxCut | MDS |
| Greedy | $5.61e^{-3}$ | $2.02e^{-2}$ | $6.24e^{-3}$ | $1.81e^{-2}$ | $5.33e{-}2$ | $2.38e^{-2}$ |
| MFA | $2.20e^{-4}$ | $9.97e^{-3}$ | $3.02e^{-5}$ | $1.37e^{-3}$ | $4.32e{-}2$ | $1.68e^{-4}$ |
| Erdos(s) | $7.81e^{-4}$ | $6.56e^{-3}$ | $1.28e^{-4}$ | $9.22e^{-4}$ | $2.79e{-}2$ | $5.66e^{-5}$ |
| VGNN(s) | $\mathbf{3.52e^{-5}}$ | $\mathbf{4.45e^{-3}}$ | $\mathbf{1.51e^{-5}}$ | $\mathbf{6.60e^{-5}}$ | $\mathbf{2.18e^{-2}}$ | $\mathbf{3.37e^{-5}}$ |

1988) for each problem. For greedy algorithms, we use node greedy for MVC, local adjustment greedy for Maxcut and sequential coverage greedy for MDS. MFA combines mean field approximation (ANDERSON 1988) and simulated annealing (SA) (Kirkpatrick, Gelatt, and Vecchi 1983), is able to achieve comparable performance with SA in around fifty times faster speed. We set its initial temperature to $\tau_0 = 5.0$, decay rate to $\alpha = 0.99$, and final temperature to the same as the threshold temperature of VGNN in each problem type. We also use semi-definite programming (Mixing) (Wang, Chang, and Kolter 2017) to solve MaxCut problem. For integer solvers, we use SCIP 7.0.2 (SCIP) (Achterberg 2009) and Gurobi 9.1 (Gurobi) (Gurobi Optimization 2020), the SotA open source and commericial solver, respectively.

**Metric**: The metric we use to evaluate the solution quality is the *ratio* $r(\cdot)$. In MVC and MDS, $r(\mathbf{x}) := f(\mathbf{x})/f(\mathbf{x}^*)$ and in Max-Cut, $r(\mathbf{x}) := f(\mathbf{x}^*)/f(\mathbf{x})$. The smaller the ratio, the better the solution is. In practice, the optimal value could be hard to find, so we used Gurobi with 3,600 second time limit and used its solution as a substitute.

## 5.2 Performance on Random Graphs

For each problem type, we generate {50k, 10k} graphs with {256-300} nodes for training and validation. Then, we generate 1k graphs with {256-300, 512-600, 1024-1100} nodes for evaluation. In evaluation, we use the sample size $n = 200$. That's to say, for LS, MFA, Erdos(s), and VGNN(s), we sample 200 solutions and report the best ratio for each method. The results of average *ratio* and average solving time (in seconds) on 1000 random graphs with size are summarized in Table 1, Table 2, Table 4 for MVC, MaxCut, and MDS, respectively. We bold the leading ratios among classical algorithms and neural methods, excluding integer solvers as they are much slower.

We can see that our VGNN achieves the best performance among all heuristic methods except for having similar performance with MFA on MaxCut (1024-1100) and MDS (256-300). Also, we are able to find better solutions and use less time than integer solvers SCIP and Gurobi. It is interesting to observe that VGNN always has better ratios than Erdos. This means, although Erdos is trying to approximate a better target distribution $P_0$, the difficulty in learning suppress its actual performance. Another phenomenon is that VGNN obtains more improvements from *stochastic* decoding than Erdos did, especially in MVC. The reason is that, under non-zero temperature, the $Q$ learned by VGNN has larger entropy so as to avoid concentrating on a single point.

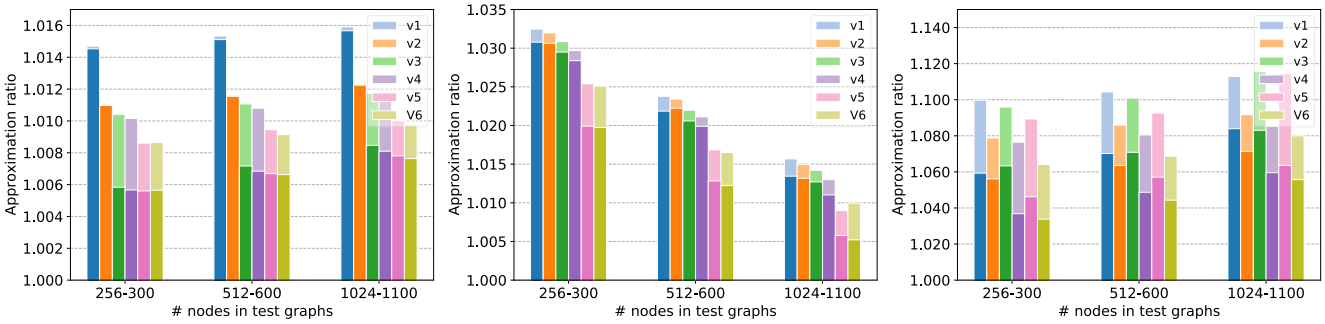
## 5.3 Performance on Real World Graphs

For *Github*, we split the dataset with ratio 6:2:2. We use 7635 graphs for training, 2545 graphs for validation, and 2545 graphs for evaluation. For *Memetracker*, we use the widely-adopted Independent Cascade model and sample a diffusion cascade from the full graph with constant set to 7, and consider the largest connected component in the graph as a single graph instance. We use 10000 graphs for training, 1000 graphs for valid, and 1000 graphs for testing. Since the *ratio* is fairly small on real-world graphs, we report the

Table 4: Evaluation MDS on 1000 random graphs

| Size | 256-300 | | 512-600 | | 1024-1100 | |
|---|---|---|---|---|---|---|
| Method | *ratio* | avg time (s) | *ratio* | avg time (s) | *ratio* | avg time (s) |
| Greedy | $1.0877 \pm 0.0395$ | $1.14e^{-4}$ | $1.0925 \pm 0.0261$ | $4.56e^{-4}$ | $1.0971 \pm 0.0191$ | $1.21e^{-3}$ |
| MFA | $\mathbf{1.0311 \pm 0.0216}$ | $4.63e^{-2}$ | $1.0431 \pm 0.0164$ | $9.43e^{-2}$ | $1.0584 \pm 0.0124$ | $1.81e^{-1}$ |
| Erdos(d) | $1.0997 \pm 0.0409$ | $4.34e^{-3}$ | $1.1042 \pm 0.0277$ | $4.62e^{-3}$ | $1.1129 \pm 0.0574$ | $5.43e^{-3}$ |
| Erdos(s) | $1.0593 \pm 0.0298$ | $3.63e^{-2}$ | $1.0702 \pm 0.0210$ | $6.79e^{-2}$ | $1.0839 \pm 0.0337$ | $1.14e^{-1}$ |
| VGNN(d) | $1.0664 \pm 0.0317$ | $4.89e^{-3}$ | $1.0686 \pm 0.0225$ | $5.18e^{-3}$ | $1.0799 \pm 0.0325$ | $6.29e^{-3}$ |
| VGNN(s) | $1.0319 \pm 0.0234$ | $5.13e^{-2}$ | $\mathbf{1.0410 \pm 0.0177}$ | $9.58e^{-2}$ | $\mathbf{1.0557 \pm 0.0139}$ | $1.79e^{-1}$ |
| SCIP(0.5s) | $1.0787 \pm 0.0376$ | $4.96e^{-1}$ | $1.0854 \pm 0.0247$ | $5.02e^{-1}$ | $1.0919 \pm 0.0883$ | $5.00e^{-1}$ |
| Gurobi(0.5s) | $1.0065 \pm 0.0136$ | $4.52e^{-1}$ | $1.0211 \pm 0.0162$ | $5.95e^{-1}$ | $1.0611 \pm 0.0307$ | $6.69e^{-1}$ |

Figure 1: Ablation results with VGNN, comparing the performance for six different versions.




MVC MaxCut MDS

*excess ratio* $\tilde{r}(\cdot) := r(\cdot) - 1$ instead. From the results in Table 3, we can see that VGNN still achieves significantly better ratios than all the other methods, meaning our method can successfully apply to real-wrold graphs.

### 5.4 Ablation Study

In this section, we present an ablation study for the annealing training and forward iteration. We compare six versions of VGNN indexed by the vector $v = (\tau_0, \tau_{\text{th}}, T)$,

- $v_1 = (0.0, 0.0, 0)$: nothing.
- $v_2 = (1.0, 0.0, 0)$: annealing.
- $v_3 = (0.2, 0.2, 0)$: threshold temperature.
- $v_4 = (1.0, 0.2, 0)$: annealing, threshold temperature.
- $v_5 = (0.2, 0.2, 1)$: forward iteration, threshold temperature.
- $v_6 = (1.0, 0.2, 1)$: forward iteration, annealing, threshold temperature.

We show the *ratios* for these six VGNN in Fig. 1. The light bars represent the *ratio* by deterministic decoding and the dark bars represent the *ratio* by *stochastic* decoding with 100 samples. The results strongly support our method. In MVC, $v_3, v_4, v_5, v_6$ are significantly better than $v_1, v_2$, showing the importance of non-zero temperature. In MaxCut, $v_5, v_6$ are significantly better than $v_1, v_2, v_3, v_4$, showing the importance of forward iteration. In MDS, $v_4, v_6$ are significantly better than $v_1, v_2, v_3, v_5$, showing the importance of annealing training schedule.

These phenomenons are directly related to the property for each problem. MVC has a lot of shallow local optima, hence using a non-zero temperature can significantly improve the performance. On the contrary, MDS has deeper local optima. Consequently, the annealing training is crucially needed to prevent being trapped in local optima. Differing from MVC and MDS having variables that tend strongly towards either 0 or 1, the optimal solutions on MaxCut are symmetric, i.e. flipping all variables does not affect the partition. As a result, the predicted $Q$ tends to collapse and give a neutral probability close to one half. Thus, the forward iteration can significantly improve the ratio on MaxCut.

## 6 Conclusion

In this paper, we propose a generic unsupervised learning framework for constrained or unconstrained CO problems. We introduce two methods, annealing training and forward iteration, naturally arisen from our framework, to help the learning of GNN. We have quantitatively verified our approach compared to other methods on both synthetic and real-world graphs. Further work includes employing more flexible varitional distributions and more sophisticated annealing schedules. In summary, we believe this work provides a new perspective for learning algorithms for CO problems.

## References


Aarts, E.; Aarts, E. H.; and Lenstra, J. K. 2003. *Local search in combinatorial optimization*. Princeton University Press.

Achterberg, T. 2009. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1): 1–41.

ANDERSON, C. P. J. 1988. Neural networks and NP-complete optimization problems; a performance study on the graph bisection problem. *Complex Systems*, 2: 58–59.

Bai, Y.; Xu, D.; Wang, A.; Gu, K.; Wu, X.; Marinovic, A.; Ro, C.; Sun, Y.; and Wang, W. 2020. Fast detection of maximum common subgraph via deep q-learning. *arXiv preprint arXiv:2002.03129*.

Barabási, A.-L.; and Albert, R. 1999. Emergence of scaling in random networks. *science*, 286(5439): 509–512.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2020. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*.

Bilbro, G.; Mann, R.; Miller, T.; Snyder, W.; van den Bout, D.; and White, M. 1988. Optimization by mean field annealing. *Advances in neural information processing systems*, 1: 91–98.

Braunstein, A.; Mézard, M.; Weigt, M.; and Zecchina, R. 2006. Constraint Satisfaction by Survey Propagation. *Computational Complexity and Statistical Physics*, 107: part–2.

Braunstein, A.; Mézard, M.; and Zecchina, R. 2005. Survey propagation: An algorithm for satisfiability. *Random Structures & Algorithms*, 27(2): 201–226.

Chen, X.; and Tian, Y. 2019. Learning to perform local rewriting for combinatorial optimization. *Advances in Neural Information Processing Systems*, 32: 6281–6292.

Dai, H.; Khalil, E. B.; Zhang, Y.; Dilkina, B.; and Song, L. 2017. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*.

Dorigo, M.; Birattari, M.; and Stutzle, T. 2006. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4): 28–39.

Dowsland, K. A.; and Thompson, J. 2012. Simulated annealing. *Handbook of natural computing*, 1623–1655.

Elidan, J.; and Koller, D. 2007. Using combinatorial optimization within max-product belief propagation. *Advances in neural information processing systems*, 19: 369.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. 2018. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, 1407–1416. PMLR.

Fey, M.; and Lenssen, J. E. 2019. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428*.

Garey, M. R.; and Johnson, D. S. 1979. Computers and intractability. *A Guide to the*.

Gasse, M.; Chételat, D.; Ferroni, N.; Charlin, L.; and Lodi, A. 2019. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*.

Gurobi Optimization, L. 2020. Gurobi optimizer reference manual. Gurobi Optimization Inc.

He, H.; Daume III, H.; and Eisner, J. M. 2014. Learning to search in branch and bound algorithms. *Advances in neural information processing systems*, 27: 3293–3301.

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

Indelman, H. C.; and Hazan, T. 2021. Learning randomly perturbed structured predictors for direct loss minimization. In *International Conference on Machine Learning*, 4585–4595. PMLR.

Ioffe, S.; and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, 448–456. PMLR.

Joshi, C. K.; Laurent, T.; and Bresson, X. 2019. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*.

Karalias, N.; and Loukas, A. 2020. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *arXiv preprint arXiv:2006.10643*.

Karp, R. M. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*, 85–103. Springer.

Khalil, E.; Le Bodic, P.; Song, L.; Nemhauser, G.; and Dilkina, B. 2016. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *science*, 220(4598): 671–680.

Kolmogorov, V.; and Wainwright, M. 2012. On the optimality of tree-reweighted max-product message-passing. *arXiv preprint arXiv:1207.1395*.

Lemos, H.; Prates, M.; Avelar, P.; and Lamb, L. 2019. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 879–885. IEEE.

Li, Z.; Chen, Q.; and Koltun, V. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. *arXiv preprint arXiv:1810.10659*.

Mezard, M.; and Montanari, A. 2009. *Information, physics, and computation*. Oxford University Press.

Morris, C.; Kriege, N. M.; Bause, F.; Kersting, K.; Mutzel, P.; and Neumann, M. 2020. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*.

Murray, I. 2007. *Advances in Markov chain Monte Carlo methods*. Ph.D. thesis, University of London.

Nair, V.; Bartunov, S.; Gimeno, F.; von Glehn, I.; Lichocki, P.; Lobov, I.; O'Donoghue, B.; Sonnerat, N.; Tjandraatmadja, C.; Wang, P.; et al. 2020. Solving Mixed Integer Programs Using Neural Networks. *arXiv preprint arXiv:2012.13349*.

Nemhauser, G. L.; Wolsey, L. A.; and Fisher, M. L. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming*, 14(1): 265–294.

Nowak, A.; Villar, S.; Bandeira, A. S.; and Bruna, J. 2017. A note on learning algorithms for quadratic assignment with graph neural networks. *stat*, 1050: 22.

Padberg, M.; and Rinaldi, G. 1991. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1): 60–100.

Paschos, V. T. 2013. *Applications of combinatorial optimization*. John Wiley & Sons.

Paulus, A.; Rolínek, M.; Musil, V.; Amos, B.; and Martius, G. 2021. CombOptNet: Fit the Right NP-Hard Problem by Learning Integer Programming Constraints. *arXiv preprint arXiv:2105.02343*.

Paulus, M. B.; Choi, D.; Tarlow, D.; Krause, A.; and Maddison, C. J. 2020. Gradient estimation with stochastic softmax tricks. *arXiv preprint arXiv:2006.08063*.

Peterson, C. 1993. Solving optimization problems with mean field methods. *Physica A: Statistical Mechanics and its Applications*, 200(1-4): 570–580.

Peterson, C.; and Söderberg, B. 1989. A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems*, 1(01): 3–22.

Pogančić, M. V.; Paulus, A.; Musil, V.; Martius, G.; and Rolinek, M. 2019. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*.

Prates, M.; Avelar, P. H.; Lemos, H.; Lamb, L. C.; and Vardi, M. Y. 2019. Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 4731–4738.

Raghavan, P. 1988. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2): 130–143.

Raghavan, P.; and Tompson, C. D. 1987. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4): 365–374.

Ravanbakhsh, S. 2015. Message passing and combinatorial optimization. *arXiv preprint arXiv:1508.05013*.

Rouhani, S.; Rahman, T.; and Gogate, V. 2020. A Novel Approach for Constrained Optimization in Graphical Models. *Advances in Neural Information Processing Systems*, 33.

Selsam, D.; Lamm, M.; Bünz, B.; Liang, P.; de Moura, L.; and Dill, D. L. 2018. Learning a SAT solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*.

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

Sun, H.; Chen, W.; Li, H.; and Song, L. 2020. Improving Learning to Branch via Reinforcement Learning. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*.

Tang, H.; Houthooft, R.; Foote, D.; Stooke, A.; Chen, X.; Duan, Y.; Schulman, J.; De Turck, F.; and Abbeel, P. 2017. # exploration: A study of count-based exploration for deep reinforcement learning. In *31st Conference on Neural Information Processing Systems (NIPS)*, volume 30, 1–18.

Toenshoff, J.; Ritzert, M.; Wolf, H.; and Grohe, M. 2021. Graph neural networks for maximum constraint satisfaction. *Frontiers in artificial intelligence*, 3: 98.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. *arXiv preprint arXiv:1506.03134*.

Vontobel, P. O.; and Koetter, R. 2005. Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes. *arXiv preprint cs/0512078*.

Wainwright, M. J.; and Jordan, M. I. 2008. *Graphical models, exponential families, and variational inference*. Now Publishers Inc.

Wang, P.-W.; Chang, W.-C.; and Kolter, J. Z. 2017. The mixing method: low-rank coordinate descent for semidefinite programming with diagonal constraints. *arXiv preprint arXiv:1706.00476*.

Wang, R.; Yan, J.; and Yang, X. 2019. Learning combinatorial embedding networks for deep graph matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 3056–3065.

Wang, R.; Yan, J.; and Yang, X. 2020. Combinatorial Learning of Robust Deep Graph Matching: an Embedding based Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Wolsey, L. A.; and Nemhauser, G. L. 1999. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018a. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018b. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, 5453–5462. PMLR.

Yao, W.; Bandeira, A. S.; and Villar, S. 2019. Experimental performance of graph neural networks on random instances of max-cut. In *Wavelets and Sparsity XVIII*, volume 11138, 111380S. International Society for Optics and Photonics.

Yehuda, G.; Gabel, M.; and Schuster, A. 2020. It's not what machines can learn, it's what we cannot teach. In *International Conference on Machine Learning*, 10831–10841. PMLR.

Yolcu, E.; and Póczos, B. 2019. Learning Local Search Heuristics for Boolean Satisfiability. In *NeurIPS*, 7990–8001.