Are Graph Neural Networks Optimal Approximation Algorithms?

Anonymous authors
Paper under double-blind review

Abstract

In this work we design graph neural network architectures that can be used to obtain optimal approximation algorithms for a large class of combinatorial optimization problems using powerful algorithmic tools from semidefinite programming (SDP). Concretely, we prove that polynomial-sized message passing algorithms can represent the most powerful polynomial time algorithms for Max Constraint Satisfaction Problems assuming the Unique Games Conjecture. We leverage this result to construct efficient graph neural network architectures, OptGNN, that obtain high-quality approximate solutions on landmark combinatorial optimization problems such as Max Cut and maximum independent set. Our approach achieves strong empirical results across a wide range of real-world and synthetic datasets against both neural baselines and classical algorithms. Finally, we take advantage of OptGNN's ability to capture convex relaxations to design an algorithm for producing dual certificates of optimality (bounds on the optimal solution) from the learned embeddings of OptGNN.

1 Introduction

Combinatorial Optimization is the class of problems that optimize functions subject to constraints over discrete search spaces. They are almost always NP-hard to solve and to approximate, owing to their typically exponential search spaces over nonconvex domains. Nevertheless, their important applications in science and engineering (Gardiner et al., 2000; Zaki et al., 1997; Smith et al., 2004; Du et al., 2017) has engendered a long history of study rooted in the following simple insight. In practice, CO instances are endowed with domain-specific structure that can be exploited by specialized algorithms (Hespe et al., 2020; Walteros & Buchanan, 2019; Ganesh & Vardi, 2020). In this context, neural networks are natural candidates for learning and then exploiting patterns in the data distribution over CO instances.

The emergence of research directions at the intersection of machine learning (ML) and CO which have obtained promising empirical results at several CO problems. However, similar to classical approaches to CO, ML pipelines have to manage a tradeoff between efficiency and optimality. Indeed, prominent works in this line of research forego optimality and focuses on parametrizing heuristics (Li et al., 2018; Khalil et al., 2017; Yolcu & Póczos, 2019; Chen & Tian, 2019) or by employing specialized models (Zhang et al., 2023; Nazari et al., 2018; Toenshoff et al., 2019; Xu et al., 2021; Min et al., 2022) and task-specific loss functions (Amizadeh et al., 2018; Karalias & Loukas, 2020; Wang et al., 2022; Karalias et al., 2022; Sun et al., 2022). Exact ML solvers that can guarantee optimality often leverage general techniques like branch and bound (Gasse et al., 2019; Paulus et al., 2022) and constraint programming (Parjadis et al., 2021; Cappart et al., 2019), which offer the additional benefit of providing approximate solutions together with a bound on the distance to the optimal solution. The downside of those methods is their exponential worst-case time complexity. This makes it clear that striking a balance between efficiency and optimality is challenging, which leads us to the central question of this paper:

*Are there neural architectures for **general** combinatorial optimization that can learn to adapt to a data distribution over instances yet capture algorithms with **optimal** worst-case approximation guarantees?*

To answer this question, we build on the extensive literature on approximation algorithms and semidefinite programming. Convex relaxations of CO problems via semidefinite programming are

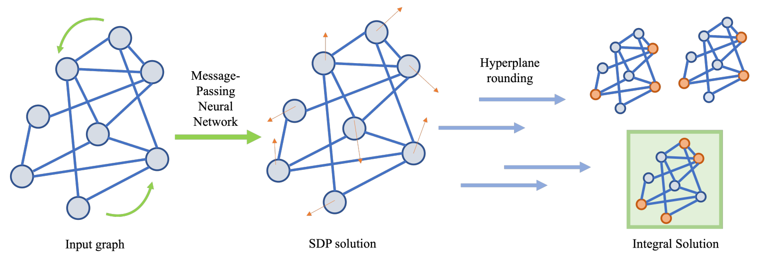

Figure 1: Message-passing neural networks for constraint satisfaction problems. Neural networks trained using the SDP objective as a loss function can be used to produce strong empirical results on Maximum Cut and Minimum Vertex Cover. We observe Maximum Cut performance of within 0.7% and Minimum Vertex Cover performance of within 3.1% of the integral value reported by Gurobi running with a time limit of 8 seconds.

the fundamental building block for breakthrough results in the design of efficient algorithms for NP-Hard combinatorial problems, such as the Goemans-Williamson approximation algorithm for Max Cut (Goemans & Williamson, 1995) and the use of the Lovász theta function to find the maximum independent set on perfect graphs (Lovász, 1979; Grötschel et al., 1981). In fact, it is known that if the Unique Games Conjecture is true, then the approximation guarantees obtained through semidefinite programming relaxations are indeed the best that can be achieved (Raghavendra, 2008; Barak & Steurer, 2014). We will leverage these results to provide an affirmative answer to our question. Our contributions can be organized into theory and experiments.

First, on the theory side we show that a polynomial time message passing algorithm approximates the solution of an SDP with the optimal integrality gap for the class of Maximum Constraint Satisfaction Problems, assuming the Unique Games Conjecture. The key theoretical insight is that a message-passing algorithm can be used to compute gradient updates for the augmented Lagrangian of an overparameterized reformulation of the SDP in (Raghavendra, 2008). This in turn leads to our main contribution, OptGNN, a graph neural network architecture that generalizes our message-passing algorithm and therefore captures its approximation guarantee.

Our second contribution is empirical. We show that our theoretical construction can be used directly within graph neural network pipelines for CO that are easy to implement and train. By training with the SDP objective as a loss function, OptGNN learns embeddings that can be regarded as feasible fractional solutions of an overparameterized low rank SDP, which are subsequently rounded to feasible integral solutions. On the primal side, we show OptGNN achieves strong empirical results against classical and neural baselines across a broad battery of datasets for landmark CO problems such as Max Cut, Vertex Cover, and Max Clique.

Finally, to underscore the fact that OptGNN captures powerful convex relaxations, we construct dual certificates of optimality, i.e bounds that are provably correct, from OptGNN embeddings for the Max Cut problem that are virtually tight for small synthetic instances. See our discussion on general neural certification schemes for extracting dual certificates from OptGNN networks in Appendix B.1.

To summarize, the contribution of this paper is twofold:

- We construct a polynomial time message passing algorithm for solving the SDP of Raghavendra (2008) for the broad class of maximum constraint satisfaction problems (including, Max Cut, max-SAT, etc.), that is optimal barring the possibility of significant breakthroughs in the foundations of algorithms.

- We construct graph neural architectures to capture this message passing algorithm and show that they achieve strong results against classical and neural baselines.

2 Related Work

Optimal approximation algorithms. In theoretical computer science, it is typically quite difficult to prove that an algorithm achieves the best approximation guarantee for a given problem, as it is hard to rule out the existence of a more powerful algorithm. The Unique Games Conjecture (UGC) (Khot, 2002) is a striking development in the theory of approximation algorithms because it is able to circumvent precisely this obstacle. If true, it implies several approximation hardness results which often match the approximation guarantees of the best-known algorithms (Raghavendra & Steurer, 2009; Raghavendra et al., 2012). In fact, it implies something even stronger: there is a *general* algorithm based on semi-definite programming that achieves the best possible approximation guarantees for several important problems (Raghavendra, 2008). Our theoretical contribution builds on these ideas to construct a neural architecture that is a candidate optimal approximation algorithm. For a complete exposition on the topic of UGC and approximation algorithms we refer the reader to Barak & Steurer (2014).

Semidefinite programming in machine learning. Semidefinite programming has already found applications in machine learning pipelines. In a similar spirit to our work, Wang et al. (2019) propose a differentiable SDP-based SAT solver based on previous works on low-rank SDPs (Wang & Kolter, 2019), while Krivách y et al. (2021) use neural networks to solve primal-dual SDP pairs for quantum information tasks. The key difference in our case is that our neural network architecture naturally aligns with solving an SDP and while our GNNs capture the properties of the SDP, in practice they can also improve upon it.

Other applications of semidefinite programming in machine learning include global minimization of functions by obtaining kernel approximations through a SDP (Rudi et al., 2020), deriving differentiable high-dimensional extensions of discrete functions (Karalias et al., 2022), and leveraging the connection between SVMs and the Lovász theta function to efficiently find the largest common dense subgraph among a collection of graphs (Jethava et al., 2013).

Neural combinatorial optimization Beyond semidefinite programming, prior work in the literature has examined the capabilities of neural networks to obtain solutions to combinatorial problems, including the ability of modern GNNs to achieve approximation guarantees for combinatorial problems Sato et al. (2019) and impossibility results for computing combinatorial properties of graphs (Loukas, 2019). It was previously shown that for maxSAT, a neural network can straightforwardly obtain a 1/2-approximation (Liu et al., 2021) which is also easily obtainable through a simple randomized algorithm Johnson (1973). SDPs can yield at least 3/4-approximations for max-SAT (Goemans & Williamson, 1995), so our approach improves significantly over previous results. Finally, a different divide and conquer approach was proposed by McCarty et al. (2021), which uses Baker's paradigm to solve maximum independent set on geometric intersection graphs by partitioning the problem into at most a linear number (on the size of the input) of subproblems of bounded size, which allows them to use a neural network on each subproblem to obtain an approximation guarantee.

In order to obtain exact solvers that are guaranteed to find the optimal solution, a prominent direction in the field involves combining solvers with neural networks either to provide a "warm start" (Benidis et al., 2023), or to learn branching heuristics for branch and bound (Gasse et al., 2019; Nair et al., 2020; Gupta et al., 2020; Paulus et al., 2022) and CDCL SAT solvers (Selsam & Bjørner, 2019; Kurin et al., 2020; Wang et al., 2021). Owing to their integration with powerful solvers, those ML pipelines are able to combine some of the strongest elements of classical algorithms and neural networks to obtain compelling results. Other related work in this vein includes constructing differentiable solvers and optimization layers (Wang et al., 2019; Agrawal et al., 2019), and the paradigm of neural algorithmic reasoning (Veličković & Blundell, 2021) which focuses on training neural networks to emulate classical polynomial-time algorithms and using them to solve various combinatorial problems (Ibarz et al., 2022; Georgiev et al., 2023). Other prominent neural approaches that have achieved strong empirical results with fast execution times follow different learning paradigms including reinforcement learning (Ahn et al., 2020; Böther et al., 2022; Tönshoff et al., 2022; Barrett et al., 2022) and unsupervised learning (Min et al., 2022; Ozolins et al., 2022).

The list of works mentioned here is by no means exhaustive, for a complete overview of the field we refer the reader to the relevant survey papers Cappart et al. (2023); Bengio et al. (2021).

3 An optimal approximation algorithm with graph neural networks

OptGNN is a neural network architecture that uses message passing to solve a semidefinite program (SDP) for a given combinatorial optimization problem. The model learns to produce a feasible low rank solution to a semidefinite program for the max-CSP. This is jointly achieved through the forward and backward pass. The message-passing steps in the forward pass are gradient updates to the node embeddings towards the direction that minimizes the augmented lagrangian of the SDP. The backward pass aids this process by backpropagating derivatives from the augmented Lagrangian $\mathcal{L}$ to the parameters of the neural network.

Before we define our OptGNN architecture, we first provide the basic technical background and intuition behind the central claim of the paper. First, we show that solving the Max Cut SDP via a simple projected gradient descent scheme amounts to executing a message-passing algorithm on the graph. This sets the stage for our main result which generalizes this observation. We describe a general maximum constraint satisfaction problem (CSP) and the standard semidefinite program (Optimization 1) that computes an approximate solution. Then we formally define OptGNN and show that message passing with OptGNN (algorithm 2) for a general CSP can solve the general SDP in Optimization 1.

3.1 Solving SDPs with message passing

To build intuition, we begin with the canonical example for the usage of semidefinite programming in combinatorial optimization: the Maximum Cut problem. Given a graph $G = (V, E)$ with vertices V, $|V| = N$ and edge set E, in the Max Cut problem we are looking to find a set of nodes in G that maximize the number of edges with exactly one endpoint in that set. Formally, this means solving the following quadratic integer program over variables $\mathbf{x} = (x_1, x_2, ..., x_N)$.

$$\begin{aligned} \max_{\mathbf{x}} \quad & \sum_{(i,j)\in E} \tfrac{1}{2}(1 - x_i x_j) & & (1) \\ \text{subject to:} \quad & x_i^2 = 1 & \forall i \in [N] & \end{aligned}$$

The global optimum of the integer program is the Max Cut. Unfortunately the discrete variables are not amenable to the tools of continuous optimization. A standard technique is to 'lift' the problem and solve it with a rank constrained SDP. Here we introduce a matrix $X \in \mathbb{R}^{N\times N}$ of variables, where we index the i'th row and j'th column entry as X_{ij}.

$$\begin{aligned} \max_{X} \quad & \textstyle\sum_{(i,j)\in E} \tfrac{1}{2}(1 - X_{ij}) & & (2) \\ \text{subject to:} \quad & X_{ii} = 1 & \forall i \in [N] & \\ & X \succeq 0 & & \\ & \text{rank}(X) = r & & \end{aligned}$$

The intuition is that a rank $r = 1$ solution to Algorithm 2 is equivalent to solving the integer program. Thus we wish to solve the optimization problem 2 for small r. A common approach is to replace the integer variables x_i with vectors $v_i \in \mathbb{R}^r$ and constrain v_i to lie on the unit sphere.

$$\begin{aligned} \min_{v_1, v_2, \ldots, v_N} \quad & -\textstyle\sum_{(i,j)\in E} \tfrac{1}{2}(1 - \langle v_i, v_j\rangle) & & (3) \\ \text{subject to:} \quad & \|v_i\| = 1 & \forall i \in [N] & \\ & v_i \in \mathbb{R}^r & & \end{aligned}$$

For r larger than $\Omega(\sqrt{N})$ Burer & Monteiro (2003) the landscape of this nonconvex optimization is benign in that all local minima are approximately global minima. Thus, for large r, simple algorithms such as projected gradient descent can find an approximate global optimum of the objective. However, for large r, it is unclear how to transform, i.e., round, the vectors $\mathbf{v}$ into a solution to the integral problem. Thus we need an approach that generalizes projected gradient descent that performs well for small r. In iteration t (and for T iterations), projected gradient descent updates vector v_i in $\mathbf{v}$ as

$$\hat{v_i}^{t+1} = v_i^t - \eta \sum\nolimits_{j \in N(i)} v_j \tag{4}$$

$$v_i^{t+1} = \frac{\hat{v_i}^{t+1}}{\|\hat{v_i}^{t+1}\|}, \tag{5}$$

where $\eta \in \mathbb{R}^+$ is an adjustable step size and we let $N(i)$ denote the neighborhood of node i. We explore a natural dynamic that we call OptGNN-MaxCut$_r(\mathbf{v})$ which generalizes gradient descent to the following form. Let $\{M_{1,t}\}_{t\in[T]} \in \mathbb{R}^{r\times r}$ and $\{M_{2,t}\}_{t\in[T]} \in \mathbb{R}^{r\times r}$ each be sets of T learnable matrices corresponding to T layers of a neural network. Then for layer t in max iterations T, for embedding v_i in $\mathbf{v}$, we have

$$\hat{v_i}^{t+1} := M_{1,t} v_i^t - M_{2,t} \sum\nolimits_{j \in N(i)} v_j + b \tag{6}$$

$$v_i^{t+1} := \frac{\hat{v_i}^{t+1}}{\|\hat{v_i}^{t+1}\|}, \tag{7}$$

where b is a learnable affine shift. Even more generally, we can write our dynamics as

$$\hat{v_i}^{t+1} := \textbf{UPDATE}(M_{1,t} v_i^t, \textbf{AGGREGATE}(M_{2,t}, \{v_j\}_{j\in N(i)}), b) \tag{8}$$

$$v_i^{t+1} := \textbf{NONLINEAR}(\hat{v_i}^{t+1}) \tag{9}$$

For efficiently computable functions **UPDATE** : $\mathbb{R}^{3r} \to \mathbb{R}^r$ and **AGGREGATE** : $\mathbb{R}^{r\times r} \times \mathbb{R}^{r|N(i)|} \to \mathbb{R}^r$ and **NONLINEAR** : $\mathbb{R}^r \to \mathbb{R}^r$. That is to say, our dynamic OptGNN-MaxCut(r) is a GNN. For an analogous discussion on Vertex Cover and Max Clique see A.

Rounding vs. Rank: To be clear, we do not regard OptGNN to be an SDP solver. Indeed we demonstrate numerous empirical examples where OptGNN outcompetes the SDP solver. Of course, on instances where the SDP is optimal, we should expect OptGNN to match the SDP performance though this is not a guarantee. We also do not regard OptGNN as an algorithm for learning a 'rounding'. The term 'rounding' is a vague 'catch-all' term for all post-processing of convex relaxations. Observe that 'rounding' is inextricably intertwined with the solving of low rank SDP formulations as for $r = 1$ no rounding is even necessary. Succinctly, OptGNN finds good CO solutions that captures powerful classes of convex relaxations. Our theory result, which we dive into next, pertains entirely to capturing the convex relaxation of Raghavendra (2008).

3.2 OptGNN

Given a set of constraints over variables, Max-CSP asks to find a variable assignment that maximizes the number of satisfied constraints. Max-CSP includes Max Cut, 3-SAT, boolean satisfiability, etc. Formally, we define Max-CSP as follows.

Definition 3.1 (Max-k-CSP). *A Constraint Satisfaction Problem $\Lambda = (\mathcal{V}, \mathcal{P}, q)$ consists of a set of N variables $\mathcal{V} := \{x_i\}_{i\in[N]}$ each taking values in an alphabet $[q]$ and a set of predicates $\mathcal{P} := \{P_z\}_{z\subset\mathcal{V}}$ where each predicate is a payoff function over k variables denoted $X_z = \{x_{i_1}, x_{i_2}, ..., x_{i_k}\}$. Here we refer to k as the arity of the Max-k-CSP. We adopt the normalization that each predicate P_z returns outputs in $[0,1]$. We index each predicate P_z by its domain z. The goal of Max-k-CSP is to maximize the payoff of the predicates.*

$$OPT := \max_{(x_1,...,x_N)\in[q]^N} \frac{1}{|\mathcal{P}|} \sum\nolimits_{P_z\in\mathcal{P}} P_z(X_z), \tag{10}$$

where we normalize by the number of constraints so that the total payoff is in $[0,1]$. Therefore we can unambiguously define an ϵ-approximate assignment as an assignment achieving a payoff of $OPT - \epsilon$.

There is a reformulation of the SDP of Raghavendra (2008) detailed in Optimization 1 that possesses the optimal integrality gap assuming the Unique Games conjecture. Precisely, for Max-k-CSP we define the approximation ratio to be

$$\text{Approximation Ratio} := \min_{\Lambda\in\text{Max-k-CSP}} \frac{OPT(\Lambda)}{SDP(\Lambda)},$$

where the minimization is being taken over all instances Λ with arity k. Then there is no polynomial time algorithm that can achieve a superior approximation ratio assuming the truth of the conjecture. Furthermore, there is a polynomial time rounding algorithm that achieves the integrality gap of SDP Optimization 1 (Raghavendra, 2008). Our main theoretical result is the following.

Theorem 3.1. *(Informal) Given a Max-k-CSP instance Λ, there exists a message passing Algorithm 2 on constraint graph G_λ with a per iteration update time of $poly(|\mathcal{P}|, q^k)$ that computes in $poly(\frac{1}{\epsilon}, |\mathcal{P}|, q^k, \log(\delta^{-1}))$ iterations an ϵ-approximate solution to SDP Optimization 1 with probability $1-\delta$. That is to say, Algorithm 2 computes a set of vectors $\boldsymbol{v}$ satisfying constraints of Optimization 1 to error ϵ with objective value denoted $OBJ(\boldsymbol{v})$ satisfying $|OBJ(\boldsymbol{v}) - SDP(\Lambda)| \leq \epsilon$.*

For the formal theorem and proof see Theorem B.1. Our algorithm is remarkably simple: perform gradient descent on the quadratically penalized objective of the reformulated SDP Optimization 1. We observe that the gradient takes the form of a message passing algorithm. For each predicate we associate q^k vectors, one vector for each assignment to each subset of k variables, for a total of $|\mathcal{P}|q^k$ vectors. The updates on each vector only depend on the vectors appearing in the same predicates. Therefore, if each variable x_i appears in no more than C predicates, every message update in the algorithm depends no more than Cq^k vectors rather than the total set of $|\mathcal{P}|q^k$ vectors. For Max Cut for example this would mean each vector corresponds to a node which is updated as a function of the vectors in adjacent vertices. This message passing form allows us to define a natural GNN generalization that captures the gradient iteration of Algorithm 2.

To make the above discussion precise, we define the constraint graph associated so a Max-CSP instance Λ as follows.

Definition 3.2. *[Constraint Graph $G_\Lambda = (V, E)$] Given a Max-k-CSP instance $\Lambda = (\mathcal{V}, \mathcal{P}, q)$ a constraint graph $G_\Lambda = (V, E)$ is comprised of vertices $V = \{v_{\phi,\zeta}\}$ for every subset of variables $\phi \subseteq z$ for every predicate $P_z \in \mathcal{P}$ and every assignment $\zeta \in [q]^k$ to the variables in z. The edges E are between any pair of vectors $v_{\phi,\zeta}$ and $v_{\phi',\zeta'}$ such that the variables in ϕ and ϕ' appear in a predicate together.*

Furthermore, we define message passing algorithms on the constraint graph G_Λ as follows.

Definition 3.3. *[Message Passing Algorithm] Given a graph $G = (V, E)$ with a set of vectors $\boldsymbol{v} = \{v_i\}_{i \in V}$ a message passing algorithm for T iterations is an update of the form; for vector v_i in $\boldsymbol{v}$ and for iteration $t \in [T]$,*

$$v_i^t = \textbf{\textit{UPDATE}}(\{v_j^t\}_{j \in N(i)}, v_i^t)$$

*For an arbitrary polynomial time computable function **UPDATE** : $\mathbb{R}^{r(|N(i)|+1)} \to \mathbb{R}^r$*

Then by inspection of the gradient iteration of Algorithm 2 we see that for a Max-k-CSP instance Λ, Algorithm 2 is a message passing algorithm on the associated constraint graph G_Λ.

Definition 3.4 (OptGNN). *Given a Max-k-CSP instance Λ, an $OptGNN_{(T,r,G_\Lambda)}(\boldsymbol{v})$ is a T layer, dimension r, neural network over constraint graph G_Λ with learnable matrices $\{M_{1,t}\}_{t \in [T]}$, $\{M_{2,t}\}_{t \in [T]}$, and affine shift $\{b_t\}_{t \in [T]}$ that generalizes the gradient iteration equation 32 of Algorithm 2 with an embedding $v \in \boldsymbol{v}$ for every node in G_Λ with updates of the form*

$$v_w^{t+1} = \textbf{\textit{UPDATE}}(M_{1,t} v_w^t, \textbf{\textit{AGGREGATE}}(M_{2,t}, \{v_j^t\}_{j \in N(w)}, v_w^t), b_t)$$

$$v_w^{t+1} = \textbf{\textit{NONLINEAR}}(v_w^{t+1})$$

*For arbitrary polynomial time computable functions **UPDATE** : $\mathbb{R}^{3r} \to \mathbb{R}^r$, **AGGREGATE** : $\mathbb{R}^{r \times r} \times \mathbb{R}^{r(|N(w)|+1)} \to \mathbb{R}^r$, and **NONLINEAR** : $\mathbb{R}^r \to \mathbb{R}^r$. Here by 'generalize' we mean there exists an instantiation of the learnable parameters $\{M_{1,t}\}_{t \in [T]}$ and $\{M_{2,t}\}_{t \in [T]}$ such that OptGNN is equivalent to equation 32.*

The crux of our proof is then to construct a message passing algorithm for the reformulated SDP Optimization 1 which is the subject of Appendix B.

3.3 Practical instantiation and Rounding

Our theoretical results lead to a simple practical recipe for solving CO problems with OptGNN. Consider a distribution over graphs $\mathcal{D}$. OptGNN computes node embeddings $V \in \mathbb{R}^r$ for mini-batches of training graphs, which are then plugged into the Lagrangian, which is used as a loss function. Going back to the maxcut example, the loss in that case would be calculated as $\mathcal{L}(V;G) = -\sum_{(i,j)\in E}\frac{1}{2}(1-\langle v_i, v_j\rangle)$. The network is then trained in a completely unsupervised fashion by minimizing $\mathbb{E}_{G\sim\mathcal{D}}[\mathcal{L}(V;G)]$ with a standard automatic differentiation package like Pytorch (Paszke et al., 2019). At inference time, the neural network produces fractional embeddings V that we discretize with hyperplane rounding followed by a simple greedy heuristic. This enables fast inference while also helping ensure feasibility in the case of problems with constraints. For an example of the Lagrangian in the case of an optimization problem with additional constraints like Vertex Cover, see Appendix A.

4 Experiments

In this section, we report experimental measurements of the performance of the OptGNN approach on two NP-complete combinatorial optimization problems, *Maximum Cut* and *Minimum Vertex Cover*.

4.1 Methods

Datasets Our experiments span a variety of randomly generated and real-world datasets. Our randomly generated datasets contain graphs from several random graph models, in particular Erdős-Rényi (with $p = 0.15$), Barabási–Albert (with $m = 4$), Holme-Kim (with $m = 4$ and $p = 0.25$), and Watts-Strogatz (with $k = 4$ and $p = 0.25$). Our real-world datasets are ENZYMES, PROTEINS, MUTAG, IMDB-BINARY, COLLAB (which we will together call **TU-small**), and REDDIT-BINARY, REDDIT-MULTI-5K, and REDDIT-MULTI-12K (which we will call **TU-REDDIT**).

We abbreviate the generated datasets using their initials and the range of vertex counts. For example, by ER (50,100) we denote Erdős-Rényi random graphs with a vertex count drawn uniformly at random from [50, 100]. In tables, we mark generated datasets with superscript [a], **TU-small** with [b], and **TU-REDDIT** with [c].

Baselines We compare the performance of our approach against classical and neural baselines. In terms of classical baselines, we run Gurobi with varying timeouts and include SDP results on smaller datasets. SDP scales extremely poorly with graph size so we omit the results for datasets with larger graphs. For minimum Vertex Cover, we include the classical baseline KaMIS, a maximum independent set solver. We also include a greedy baseline, which is the function `one_exchange` (for Maximum Cut) or `min_weighted_vertex_cover` (for minimum Vertex Cover) from `networkx` (Hagberg et al., 2008). Our neural baselines include LwD (Ahn et al., 2020) and DGL-TreeSearch (Li et al., 2018; Böther et al., 2022).

Validation and test splits For each dataset we hold out a validation and test slice for evaluation. In our generated graph experiments we set aside 1000 graphs each for validation and testing. Each step of training ran on randomly generated graphs. For **TU-small**, we used a train/validation/test split of 0.8/0.1/0.1. For **TU-REDDIT**, we set aside 100 graphs each for validation and testing.

Scoring To measure a model's score on a graph, we first run the model on the graph to generate an SDP output, and then round this output to an integral solution using 1,000 random hyperplanes. We ran validation periodically during each training run and retained the model that achieved the highest validation score. Then for each model and dataset, we selected the hyperparameter setting that achieved the highest validation score, and we report the average score measured on the test slice. Please see subsection C.2 for further details on the hyperparameter ranges used.

4.2 Performance

Table 1 presents the average integral cut value achieved by OptGNN and classical baselines on a variety of datasets. We note that Greedy achieves poor performance compared to OptGNN and Gurobi on every dataset, indicating that for these datasets, finding Maximum Cut is not trivial. On the worst case, WS (400, 500), OptGNN achieves a cut value within 1.1% on average of Gurobi with an 8s time limit. On other datasets, OptGNN is typically within a fraction of a percent. Notably, OptGNN is within 0.1% of Gurobi 8s on all the TU datasets.

Dataset	OptGNN	Greedy	Gurobi 0.1s	1.0s	8.0s
BA[a] (50,100)	351.49 (18)	200.10	351.87	352.12	352.12
BA[a] (100,200)	717.19 (20)	407.98	719.41	719.72	720.17
BA[a] (400,500)	2197.99 (66)	1255.22	2208.11	2208.11	2212.49
ER[a] (50,100)	528.95 (18)	298.55	529.93	530.03	530.16
ER[a] (100,200)	1995.05 (24)	1097.26	2002.88	2002.88	2002.93
ER[a] (400,500)	16387.46 (225)	8622.34	16476.72	16491.60	16495.31
HK[a] (50,100)	345.74 (18)	196.23	346.18	346.42	346.42
HK[a] (100,200)	709.39 (23)	402.54	711.68	712.26	712.88
HK[a] (400,500)	2159.90 (61)	1230.98	2169.46	2169.46	2173.88
WC[a] (50,100)	198.29 (18)	116.65	198.74	198.74	198.74
WC[a] (100,200)	389.83 (24)	229.43	390.96	392.07	392.07
WC[a] (400,500)	1166.47 (78)	690.19	1173.45	1175.97	1179.86
MUTAG[b]	27.95 (9)	16.95	27.95	27.95	27.95
ENZYMES[b]	81.37 (14)	48.53	81.45	81.45	81.45
PROTEINS[b]	102.15 (12)	60.74	102.28	102.36	102.36
IMDB-BIN[b]	97.47 (11)	51.85	97.50	97.50	97.50
COLLAB[b]	2622.41 (22)	1345.70	2624.32	2624.57	2624.62
REDDIT-BIN[c]	693.33 (186)	439.79	693.02	694.10	694.14
REDDIT-M-12K[c]	568.00 (89)	358.40	567.71	568.91	568.94
REDDIT-M-5K[c]	786.09 (133)	495.02	785.44	787.48	787.92

Table 1: Performance of OptGNN, Greedy, and Gurobi 0.1s, 1s, and 8s on Maximum Cut. For each approach and dataset, we report the average cut size measured on the test slice. Here, higher score is better. In parentheses, we include the average runtime in *milliseconds* for OptGNN.

Dataset	OptGNN	Greedy	Gurobi 0.1s	1.0s	8.0s
BA[a] (50,100)	42.88 (27)	51.92	42.82	42.82	42.82
BA[a] (100,200)	83.43 (25)	101.42	83.19	83.19	83.19
BA[a] (400,500)	248.74 (27)	302.53	256.33	246.49	246.46
ER[a] (50,100)	55.25 (21)	68.85	55.06	54.67	54.67
ER[a] (100,200)	126.52 (18)	143.51	127.83	123.47	122.76
ER[a] (400,500)	420.70 (41)	442.84	423.07	423.07	415.52
HK[a] (50,100)	43.06 (25)	51.38	42.98	42.98	42.98
HK[a] (100,200)	84.38 (25)	100.87	84.07	84.07	84.07
HK[a] (400,500)	249.26 (27)	298.98	247.90	247.57	247.57
WC[a] (50,100)	46.38 (26)	72.55	45.74	45.74	45.74
WC[a] (100,200)	91.28 (21)	143.70	89.80	89.80	89.80
WC[a] (400,500)	274.21 (31)	434.52	269.58	269.39	269.39
MUTAG[b]	7.79 (18)	12.84	7.74	7.74	7.74
ENZYMES[b]	20.00 (24)	27.35	20.00	20.00	20.00
PROTEINS[b]	25.29 (18)	33.93	24.96	24.96	24.96
IMDB-BIN[b]	16.78 (18)	17.24	16.76	16.76	16.76
COLLAB[b]	67.50 (23)	71.74	67.47	67.46	67.46
REDDIT-BIN[c]	82.85 (38)	117.16	82.81	82.81	82.81
REDDIT-M-12K[c]	81.55 (25)	115.72	81.57	81.52	81.52
REDDIT-M-5K[c]	107.36 (33)	153.24	108.73	107.32	107.32

Table 2: Performance of OptGNN, Greedy, and Gurobi 0.1s, 1s, and 8s on Minimum Vertex Cover. For each approach and dataset, we report the average Vertex Cover size measured on the test slice. Here, lower score is better. In parentheses, we include the average runtime in *milliseconds* for OptGNN.

Table 2 presents the average size of the Vertex Cover achieved by OptGNN and classical baselines on our datasets. For this problem OptGNN also performs nearly as well as Gurobi 8s, remaining within 1% on the TU datasets and 3.1% on the worst case, ER (100, 200).

Dataset	GAT	GCNN	GIN	GatedGCNN	OptGNN
ER[a] (50,100)	525.92 (25)	500.94 (17)	498.82 (14)	526.78 (14)	**528.95** (18)
ER[a] (100,200)	1979.45 (20)	1890.10 (26)	1893.23 (23)	1978.78 (21)	**1995.05** (24)
ER[a] (400,500)	16317.69 (208)	15692.12 (233)	15818.42 (212)	16188.85 (210)	**16387.46** (225)
MUTAG[b]	27.84 (19)	27.11 (12)	27.16 (13)	**27.95** (14)	**27.95** (9)
ENZYMES[b]	80.73 (17)	74.03 (12)	73.85 (16)	81.35 (9)	**81.37** (14)
PROTEINS[b]	100.94 (14)	92.01 (19)	92.62 (17)	101.68 (10)	**102.15** (12)
IMDB-BIN[b]	81.89 (18)	70.56 (21)	81.50 (10)	97.11 (9)	**97.47** (11)
COLLAB[b]	2611.83 (22)	2109.81 (21)	2430.20 (23)	2318.19 (18)	**2622.41** (22)

Table 3: Performance of various model architectures for selected datasets on Maximum Cut. Here, higher is better. GAT is the Graph Attention network (Veličković et al., 2018)
, GIN is the Graph Isomorphism Network (Xu et al., 2019), GCNN is the Graph Convolutional Neural Network (Morris et al., 2019), and GatedGCNN is the gated version (Li et al., 2015).

4.3 Ablation

Our approach of training on the SDP objective generalizes to neural network architectures other than OptGNN. We trained several architectures besides OptGNN on a subset of our datasets for both maximum cut and minimum vertex cover. We present the comparison of their performance to OptGNN for maximum cut in Table 3; please see subsection C.4 for the analogous table for minimum vertex cover. On the datasets we used, OptGNN outperforms the other architectures we tested. We note that compared to OptGNN, many other models performed fairly well; for instance, GatedGCNN achieves average cut values within a few percent of OptGNN on nearly all the datasets (excluding COLLAB). An interesting question for future investigation is what architectures may perform better than OptGNN.

5 Conclusion

We have presented OptGNN, a GNN that can capture provably optimal message passing algorithms for a large class of combinatorial optimization problems. OptGNN achieves the appealing combination of obtaining approximation guarantees while also being able to adapt to the data to achieve improved results. Empirically, we observed that the OptGNN architecture achieves strong performance on a wide range of datasets and on multiple problems. Since the landscape of combinatorial optimization is expansive, there are still important challenges that have to be addressed within the scope of this work such as the extension of our approach to problems with more complex constraints and objectives. OptGNN offers a novel perspective on the connections between general approximation algorithms and neural networks, and opens up new avenues for exploration. These include the design of more powerful and sound (neural) rounding procedures that can secure approximation guarantees, the construction of neural certificates that improve upon the ones we described in Appendix B.1, and the design of neural SDP-based branch and bound solvers.

REFERENCES

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *International Conference on Machine Learning*, pp. 134–144. PMLR, 2020.

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. 2018.

Boaz Barak and David Steurer. Sum-of-squares proofs and the quest toward optimal algorithms. *arXiv preprint arXiv:1404.5236*, 2014.

Thomas D Barrett, Christopher WF Parsonson, and Alexandre Laterre. Learning to solve combinatorial graph partitioning problems via efficient exploration. *arXiv preprint arXiv:2205.14105*, 2022.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021.

Konstantinos Benidis, Ugo Rosolia, Syama Rangapuram, George Iosifidis, and Georgios Paschos. Solving recurrent mips with semi-supervised graph neural networks. *arXiv preprint arXiv:2302.11992*, 2023.

Srinadh Bhojanapalli, Nicolas Boumal, Prateek Jain, and Praneeth Netrapalli. Smoothed analysis for low-rank solutions to semidefinite programs in quadratic penalty form, 2018.

Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. *arXiv preprint arXiv:2201.10494*, 2022.

Samuel Burer and Renato Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming, Series B*, 95:329–357, 02 2003. doi: 10.1007/s10107-002-0352-8.

Quentin Cappart, Emmanuel Goutierre, David Bergman, and Louis-Martin Rousseau. Improving optimization bounds using machine learning: decision diagrams meet deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1443–1451, 2019.

Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Velickovic. Combinatorial optimization and reasoning with graph neural networks. *J. Mach. Learn. Res.*, 24:130–1, 2023.

Xinyun Chen and Yuandong Tian. Learning to perform local rewriting for combinatorial optimization. In *Advances in Neural Information Processing Systems*, pp. 6278–6289, 2019.

Jiaoman Du, Xiang Li, Lean Yu, Ralescu Dan, and Jiandong Zhou. Multi-depot vehicle routing problem for hazardous materials transportation: A fuzzy bilevel programming. *Information sciences*, 399:201–218, 2017.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

Vijay Ganesh and Moshe Y Vardi. On the unreasonable effectiveness of sat solvers. *Beyond the Worst-Case Analysis of Algorithms*, pp. 547–566, 2020.

Eleanor J Gardiner, Peter Willett, and Peter J Artymiuk. Graph-theoretic techniques for macromolecular docking. *Journal of Chemical Information and Computer Sciences*, 40(2):273–279, 2000.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*, 2019.

Dobrik Georgiev, Danilo Numeroso, Davide Bacciu, and Pietro Liò. Neural algorithmic reasoning for combinatorial optimisation. *arXiv preprint arXiv:2306.06064*, 2023.

Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6): 1115–1145, 1995.

Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33: 18087–18097, 2020.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman (eds.), *Proceedings of the 7th Python in Science Conference*, pp. 11 – 15, Pasadena, CA USA, 2008.

Demian Hespe, Sebastian Lamm, Christian Schulz, and Darren Strash. Wegotyoucovered: The winning solver from the pace 2019 challenge, vertex cover track. In *2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing*, pp. 1–11. SIAM, 2020.

Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, et al. A generalist neural algorithmic learner. In *Learning on Graphs Conference*, pp. 2–1. PMLR, 2022.

Vinay Jethava, Anders Martinsson, Chiranjib Bhattacharyya, and Devdatt Dubhashi. Lovász ϑ function, svms and finding dense subgraphs. *The Journal of Machine Learning Research*, 14(1): 3495–3536, 2013.

Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan. How to escape saddle points efficiently, 2017.

David S Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pp. 38–49, 1973.

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

Nikolaos Karalias, Joshua Robinson, Andreas Loukas, and Stefanie Jegelka. Neural set function extensions: Learning with discrete functions in high dimensions. *Advances in Neural Information Processing Systems*, 35:15338–15352, 2022.

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

Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 767–775, 2002.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Tamás Kriváchy, Yu Cai, Joseph Bowles, Daniel Cavalcanti, and Nicolas Brunner. High-speed batch processing of semidefinite programs with feedforward neural networks. *New Journal of Physics*, 23(10):103034, 2021.

Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Can q-learning with graph networks learn a generalizable branching heuristic for a sat solver? *Advances in Neural Information Processing Systems*, 33:9608–9621, 2020.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

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

Minghao Liu, Fuqi Jia, Pei Huang, Fan Zhang, Yuchen Sun, Shaowei Cai, Feifei Ma, and Jian Zhang. Can graph neural networks learn to solve maxsat problem? *arXiv preprint arXiv:2111.07568*, 2021.

Andreas Loukas. What graph neural networks cannot learn: depth vs width, 2019.

László Lovász. On the shannon capacity of a graph. *IEEE Transactions on Information theory*, 25 (1):1–7, 1979.

Evan McCarty, Qi Zhao, Anastasios Sidiropoulos, and Yusu Wang. Nn-baker: A neural-network infused algorithmic framework for optimization problems on geometric intersection graphs. *Advances in Neural Information Processing Systems*, 34:23023–23035, 2021.

Yimeng Min, Frederik Wenkel, Michael Perlmutter, and Guy Wolf. Can hybrid geometric scattering networks help solve the maximum clique problem? *Advances in Neural Information Processing Systems*, 35:22713–22724, 2022.

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.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid Von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pp. 9839–9849, 2018.

Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural sat solver. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2022.

Augustin Parjadis, Quentin Cappart, Louis-Martin Rousseau, and David Bergman. Improving branch-and-bound using decision diagrams and reinforcement learning. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 18th International Conference, CPAIOR 2021, Vienna, Austria, July 5–8, 2021, Proceedings 18*, pp. 446–455. Springer, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Max B Paulus, Giulia Zarpellon, Andreas Krause, Laurent Charlin, and Chris Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International conference on machine learning*, pp. 17584–17600. PMLR, 2022.

Prasad Raghavendra. Optimal algorithms and inapproximability results for every csp? In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pp. 245–254, 2008.

Prasad Raghavendra and David Steurer. Towards computing the grothendieck constant. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 525–534. SIAM, 2009.

Prasad Raghavendra, David Steurer, and Madhur Tulsiani. Reductions between expansion problems. In *2012 IEEE 27th Conference on Computational Complexity*, pp. 64–73. IEEE, 2012.

Alessandro Rudi, Ulysse Marteau-Ferey, and Francis Bach. Finding global minima via kernel approximations. *arXiv preprint arXiv:2012.11978*, 2020.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems, 2019.

Daniel Selsam and Nikolaj Bjørner. Guiding high-performance sat solvers with unsat-core predictions. In *Theory and Applications of Satisfiability Testing–SAT 2019: 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings 22*, pp. 336–353. Springer, 2019.

Alexander Smith, Andreas Veneris, and Anastasios Viglas. Design diagnosis using boolean satisfiability. In *ASP-DAC 2004: Asia and South Pacific Design Automation Conference 2004 (IEEE Cat. No. 04EX753)*, pp. 218–223. IEEE, 2004.

David Steurer. Fast sdp algorithms for constraint satisfaction problems. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pp. 684–697, USA, 2010. Society for Industrial and Applied Mathematics. ISBN 9780898716986.

Haoran Sun, Etash K Guha, and Hanjun Dai. Annealed training for combinatorial optimization on graphs. *arXiv preprint arXiv:2207.11542*, 2022.

Jan Toenshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Run-csp: Unsupervised learning of message passing networks for binary constraint satisfaction problems. *arXiv preprint arXiv:1909.08387*, 2019.

Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any csp: Graph neural networks as fast global search heuristics for constraint satisfaction. *arXiv preprint arXiv:2208.10227*, 2022.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021. ISSN 2666-3899.

Jose L Walteros and Austin Buchanan. Why is maximum clique often easy in practice. *Oper. Res*, 2019.

Haoyu Peter Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. In *Advances in Neural Information Processing Systems*, 2022.

Po-Wei Wang and J Zico Kolter. Low-rank semidefinite programming for the max2sat problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1641–1649, 2019.

Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pp. 6545–6554. PMLR, 2019.

Wenxi Wang, Yang Hu, Mohit Tiwari, Sarfraz Khurshid, Kenneth McMillan, and Risto Miikkulainen. Neurocomb: Improving sat solving with graph neural networks. *arXiv preprint arXiv:2110.14053*, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

Yunqiu Xu, Meng Fang, Ling Chen, Gangyan Xu, Yali Du, and Chengqi Zhang. Reinforcement learning with multiple relational attention for solving vehicle routing problems. *IEEE Transactions on Cybernetics*, 52(10):11107–11120, 2021.

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

Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, et al. New algorithms for fast discovery of association rules. In *KDD*, volume 97, pp. 283–286, 1997.

Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial optimization problems with gflownets. *arXiv preprint arXiv:2305.17010*, 2023.

A Vertex Cover and Max Clique

Minimum Vertex Cover can be written as the following integer program

$$\textbf{Minimize:} \quad \text{VertexCover}(\mathbf{x}) := \sum_{i\in[N]} \frac{1+x_i}{2} \tag{11}$$

$$\textbf{Subject to:} \quad (1-x_i)(1-x_j) = 0 \qquad \forall (i,j)\in E \tag{12}$$

$$x_i^2 = 1 \qquad \forall i \in [N] \tag{13}$$

To deal with the constraint on the edges $(1-x_i)(1-x_j) = 0$, we add a quadratic penalty to the objective with a penalty parameter $\rho > 0$ yielding

$$\textbf{Minimize:} \quad \text{VertexCover}(\mathbf{x}) := \sum_{i\in[N]} \frac{1+x_i}{2} + \rho \sum_{(i,j)\in E} (1 - x_i - x_j + x_i x_j)^2 \tag{14}$$

$$\textbf{Subject to:} \quad x_i^2 = 1 \quad \forall i \in [N] \tag{15}$$

Analogously to Max Cut, we introduce a natural low rank vector formulation $\text{LiftVertexCover}_r(\mathbf{v})$ for vectors $\mathbf{v} = \{v_i\}_{i\in[N]}$ in r dimensions.

$$\textbf{Minimize:} \quad \text{LiftVertexCover}_r(\mathbf{v}) := \sum_{i\in[N]} \frac{1+\langle v_i, e_1\rangle}{2} + \rho \sum_{(i,j)\in E} (1 - \langle v_i, e_1\rangle - \langle v_j, e_1\rangle + \langle v_i, v_j\rangle)^2 \tag{16}$$

$$\textbf{Subject to:} \quad \|v_i\| = 1 \quad v_i \in \mathbb{R}^r \quad \forall i \in [N] \tag{17}$$

Now we can design a simple projected gradient descent scheme as follows. For iteration t in max iterations T, and for vector v_i in $\mathbf{v}$ we perform the following update.

$$\hat{v}_i^{\,t+1} := v_i^t - \eta\big(e_1 + 2\rho \sum_{j\in N(i)} (1 - \langle v_i, e_1\rangle - \langle v_j, e_1\rangle + \langle v_i, v_j\rangle)(-e_1 + v_j)\big) \tag{18}$$

$$v_i^{t+1} := \frac{\hat{v}_i^{t+1}}{\|\hat{v}_i^{t+1}\|} \tag{19}$$

We can then define a $\text{OptGNN-VertexCover}_r(\mathbf{v})$ analogously with learnable matrices $\{M_{1,t}\}_{t\in[T]} \in \mathbb{R}^{r\times r}$ and $\{M_{2,t}\}_{t\in[T]} \in \mathbb{R}^{r\times r}$ which are each sets of T learnable matrices corresponding to T layers of neural network. Then for layer t in max iterations T, for v_i in $\mathbf{v}$, we have

$$\hat{v}_i^{\,t+1} := M_1 v_i^t + M_2\big(e_1 + 2\rho \sum_{j\in N(i)} (1 - \langle v_i, e_1\rangle - \langle v_j, e_1\rangle + \langle v_i, v_j\rangle)(-e_1 + v_j)\big) + b \tag{20}$$

$$v_i^{t+1} := \frac{\hat{v}_i^{t+1}}{\|\hat{v}_i^{t+1}\|} \tag{21}$$

Which once again we see can be captured by the dynamic

$$\hat{v_i}^{t+1} = \textbf{UPDATE}(M_1 v_i^t, M_2\textbf{AGGREGATE}(\{v_j\}_{j\in N(i)}), b) \tag{22}$$

$$v_i^{t+1} = \textbf{NONLINEAR}(\hat{v_i}^{t+1}) \tag{23}$$

For functions **UPDATE** : $\mathbb{R}^{3r} \to \mathbb{R}^r$ and **AGGREGATE** : $\mathbb{R}^{r|N(i)|} \to \mathbb{R}^r$ and **NONLINEAR** : $\mathbb{R}^r \to \mathbb{R}^r$.

B Optimality of Message Passing for Max-CSP

Our primary theoretical result is that a polynomial time message passing algorithm on an appropriately defined constraint graph computes the approximate optimum of Optimization 1 which is notable for being an SDP that achieves the Unique Games optimal integrality gap.

Our proof roadmap is simple. First, we design an SDP relaxation Optimization 1 for Max-k-CSP that is provably equivalent to the SDP of Raghavendra (2008) and therefore inherits its complexity theoretic optimality. Finally, we design a message passing algorithm to approximately solve Optimization 1 in polynomial time to polynomial precision. Our message passing algorithm has the advantage of being formulated on an appropriately defined constraint graph. For a Max-k-CSP instance Λ with N variables, $|\mathcal{P}|$ predicates, over an alphabet of size q, it takes $|\mathcal{P}|q^k$ space to represent the Max-CSP. Our message passing algorithm achieves an additive ϵ approximation in time $poly(\frac{1}{\epsilon}, N, |\mathcal{P}|q^k)$ which is then polynomial in the size of the CSP and inverse polynomial in the precision.

Here we briefly reiterate the definition of Max-k-CSP. A Max-k-CSP instance $\Lambda = (\mathcal{V}, \mathcal{P}, q)$ consists of a set of N variables $\mathcal{V} := \{x_i\}_{i\in[N]}$ each taking values in an alphabet $[q]$ and a set of predicates $\mathcal{P} := \{P_z\}_{z\subset\mathcal{V}}$ where each predicate is a payoff function over k variables denoted $z = \{x_{i_1}, x_{i_2}, ..., x_{i_k}\}$. Here we refer to k as the arity of the GCSP, and we adopt the normalization that each predicate P_z returns outputs in $[0,1]$. We index each predicate P_z by its domain z and we will use the notation $\mathcal{S}(P)$ to denote the domain of a predicate P. The goal of Max-k-CSP is to maximize the payoff of the predicates.

$$\max_{(x_1,...,x_N)\in[q]^N} \frac{1}{|\mathcal{P}|} \sum_{P_z\in\mathcal{P}} P_z(X_z) \tag{24}$$

Where X_z denotes the assignment of variables $\{x_i\}_{i\in z}$.

There is an SDP relaxation of equation 24 that is the "qualitatively most powerful assuming the Unique Games conjecture" Raghavendra (2008). More specifically, the integrality gap of the SDP achieves the Unique Games optimal approximation ratio. Furthermore, there exists a rounding that achieves its integrality gap.

SDP Reformulation: Next we will introduce the SDP formulation we adopt in this paper. For the sake of exposition and notational simplicity, we will work with binary Max-k-CSP's where $q = \{0,1\}$. The extension to general q is straightforward and detailed in the appendix. TODO

We will adopt the standard pseudoexpectation and pseudodistribution formalism in describing our SDP. Let $\tilde{\mathbb{E}}_\mu[\mathbf{X}]$ be a matrix in dimension $\mathbb{R}^{(N+1)^{k/2}\times(N+1)^{k/2}}$ of optimization variables defined as follows

$$\tilde{\mathbb{E}}_\mu[\mathbf{X}] := \tilde{\mathbb{E}}_\mu[(1, x_1, x_2, ..., x_N)^{\otimes k/2}\big((1, x_1, x_2, ..., x_N)^{\otimes k/2}\big)^T] \tag{25}$$

Where we use $\otimes$ to denote tensor product. It is convenient to think of $\tilde{\mathbb{E}}_\mu[\mathbf{X}]$ as a matrix of variables denoting the up to k multilinear moments of a distribution μ over the variables $\mathcal{V}$. A multilinear polynomial is a polynomial of the form $X_\phi := \prod_{i\in\phi} x_i$ for some subset of the variables $\phi \subseteq \mathcal{V}$. We index the variables of the matrix $\tilde{\mathbb{E}}_\mu[\mathbf{X}]$ by the multilinear moment that it represents. Notice that this creates repeat copies as their are multiple entries representing the same monomial. This is dealt with by constraining the repeated copies to be equal with linear equality constraints.

Specifically, let z be a subset of the CSP variables $z \subset \{x_i\}_{i\in[N]}$ of size k. Let X_z denote the multilinear moment $X_z := \prod_{i\in z} x_i$. Then $\tilde{\mathbb{E}}_\mu[X_z]$ denotes the SDP variable corresponding to the multilinear moment $\mathbb{E}_\mu[X_z]$. Of course optimizing over the space of distributions μ over $\mathcal{V}$ is intractable, and so we opt for optimizing over the space of low degree pseudodistributions and their associated pseudoexpecation functionals. See Barak & Steurer (2014) for references therein.

In particular, for any subset of variables $X_z := \{x_{i_1}, ..., x_{i_k}\} \in \mathcal{V}$ we let $\tilde{\mathbb{E}}_\mu[\mathbf{X}]\big|_{z,d}$ denote the matrix of the up to degree up to d multilinear moments of the variables in z.

$$\tilde{\mathbb{E}}_\mu[\mathbf{X}]\big|_z := \tilde{\mathbb{E}}_\mu[(1, x_{i_1}, x_{i_2}, ..., x_{i_k})^{\otimes d/2}\big((1, x_{i_1}, x_{i_2}, ..., x_{i_k})^{\otimes d/2}\big)^T] \tag{26}$$

Subsequently, we describe a pseudoexpectation formulation of our SDP followed by a vector formulation.

Multilinear Formulation: A predicate for a boolean Max-k-CSP $P_z(X_z)$ can be written as a multilinear polynomial

$$P_z(X_z) := \sum_{\tau=(\tau_1,...,\tau_k)\in\{-1,1\}^k} w_{z,\tau} \prod_{x_i\in z} \frac{1+\tau_i x_i}{2} := \sum_{s\subseteq z} w_s X_s \tag{27}$$

For some real valued weights $w_{z,\tau}$ and w_s which are simply the fourier coefficients of the function P_z. Then the pseudoexpectation formulation of our SDP is as follows

$$\max_{\tilde{\mathbb{E}}_\mu[\mathbf{X}]} \sum_{P_z\in\mathcal{P}} \tilde{\mathbb{E}}_\mu[P_z(X_z)] \tag{28}$$

subject to the following constraints

1. **Unit:** $\tilde{\mathbb{E}}_\mu[1] = 1, \tilde{\mathbb{E}}_\mu[x_i^2] = 1$ for all $x_i \in \mathcal{V}$, and $\tilde{\mathbb{E}}_\mu[\prod_{i\in s} x_i^2 \prod_{j\in s'} x_j] = \tilde{\mathbb{E}}_\mu[\prod_{j\in s'} x_j]$ for all $s, s' \subseteq \mathcal{S}(P)$ for every predicate $P \in \mathcal{P}$ such that $2s + s' \leq k$. In expectation, the squares of all multilinear polynomials are equal to 1.
2. **Positive Semidefinite:** $\tilde{\mathbb{E}}_\mu[\mathbf{X}]|_{\mathcal{V},2} \succeq 0$ i.e the degree two pseudoexpectation is positive semidefinite. $\tilde{\mathbb{E}}_\mu[\mathbf{X}]\big|_{z,k} \succeq 0$ for all $z = \mathcal{S}(P)$ for all $P \in \mathcal{P}$. The moment matrix for the multilinear polynomials corresponding to every predicate is positive semidefinite.

Equivalently we can view the SDP in terms of the vectors in the cholesky decomposition of $\tilde{\mathbb{E}}_\mu[\mathbf{X}]$. We rewrite the above SDP accordingly. For this purpose it is useful to introduce the notation $\zeta(A, B) := A \cup B / A \cap B$. It is also useful to introduce the notation $\mathcal{C}(s)$ for the size of the set $\{g, g' \subseteq s : \zeta(g, g') = s\}$.

Lemma B.1. *For Max-k-CSP instance Λ, The SDP of Optimization 1 is at least as tight as the SDP of Raghavendra (2008).*

Proof. The SDP of Raghavendra (2008) is a based degree 2 SoS SDP augmented with k-local distributions for every predicate $P \in \mathcal{P}$. By using the vectors of the cholesky decomposition and constraining them to be unit vectors we automatically capture degree 2 SoS. To capture k local distributions we simply enforce degree $2k$ SoS on the boolean hypercube for the domain of every predicate. This can be done with the standard vector formulation written in Optimization 1. See Barak & Steurer (2014) for background and references. □

Theorem B.1. *Algorithm 2 computes in $poly(1/\epsilon, |\mathcal{P}|, 2^k, \log(\delta^{-1}))$ iterations a set of vectors $\mathbf{v} := \{\hat{v}_s\}$ for all $s \subseteq \mathcal{S}(P)$ for all $P \in \mathcal{P}$ that satisfy the constraints of Optimization 1 to error ϵ and approximates the optimum of Optimization 1 to error ϵ with probability $1 - \delta$*

$$\big|\sum_{P_z\in\mathcal{P}} \tilde{\mathbb{E}}_{\hat{\mu}}[P_z(X_z)] - OPTSDP(\Lambda)\big| \leq \epsilon$$

where $OPTSDP(\Lambda)$ is the optimum of Optimization 1.

Algorithm 1 SDP Vector Formulation for Max-k-CSP

```
1: procedure SDP VECTOR FORMULATION($\Lambda = (\mathcal{V}, \mathcal{P}, \{0,1\})$)    ▷ SDP Equivalent to UGC optimal
       Minimize: $\sum_{P_z \subset \mathcal{P}} \tilde{\mathbb{E}}_\mu[-P_z(X_z)] := \sum_{P_z \in \mathcal{P}} \sum_{s \subseteq z} w_s \frac{1}{|\mathcal{C}(s)|} \sum_{g,g' \subseteq s : \zeta(g,g')=s} \langle v_g, v_{g'} \rangle$    (29)
2:                                                     ▷ multilinear formulation of objective
       Subject to: $\|v_s\|^2 = 1 \quad \forall s \subseteq \mathcal{S}(P)\ \forall P \in \mathcal{P}$    (30)
       $\tilde{\mathbb{E}}_\mu[X_{\zeta(g,g')}] := \langle v_g, v_{g'} \rangle = \langle v_h, v_{h'} \rangle \quad \forall \zeta(g,g') = \zeta(h,h')$ s.t $g \cup g' \subseteq \mathcal{S}(P)\ \forall P \in \mathcal{P}$    (31)
                   ▷ First constraint is the square of multilinear polynomials are unit
3:     ▷ Second constraint are degree $2k$ SoS constraints for products of multilinear polynomials
4: end procedure
```

Algorithm 2 Message Passing for Max-CSP

```
1: procedure MESSAGE PASSING($\Lambda = (\mathcal{V}, \mathcal{P}, \{0,1\})$)
2:     $n \leftarrow |\mathcal{P}| 2^k \log(\delta^{-1})$
3:     $\eta, \psi, \sigma \leftarrow n^{-100}$            ▷ Initialize step size, noise threshold, and noise variance
4:     $\mathbf{v}^0 = \{v_s\}_{s \subseteq z : P_z \in \mathcal{P}} \leftarrow Uniform(\mathcal{S}^{n-1})$  ▷ Initialize vectors to uniform on the unit sphere
5:     for $t \in [poly(\frac{1}{\epsilon}, |\mathcal{P}|, 2^k, \log(\delta^{-1}))]$ do
6:         for $v_w^t \in \mathbf{v}_t$ do                                  ▷ Iterate over vectors
7:
               $\hat{v}_w^{t+1} \leftarrow v_w^t - \eta \sum_{P_z \in \mathcal{P} : w \subseteq z} \sum_{s \subseteq z : w \subseteq s} w_s \frac{1}{|\mathcal{C}(s)|} \sum_{w' \subseteq s : \zeta(w,w')=s} v_{w'}^t$    (32)
               $+2\rho \Bigg[ \sum_{P_z \in \mathcal{P} : w \subseteq z} \sum_{w',h,h' \subseteq s : \zeta(w,w')=\zeta(h,h')} \big( \langle v_w^t, v_{w'}^t \rangle - \langle v_h^t, v_{h'}^t \rangle \big) v_w'^t$    (33)
               $+(\|v_w^t\|^2 - 1) v_w^t \Bigg]$    (34)
                   ▷ Update each vector with neighboring vectors in constraint graph
8:
9:             if $\|v_w^{t+1} - v_w^t\| \leq \psi$ then
10:                $\zeta \leftarrow N(0, \sigma I)$
11:            else
12:                $\zeta \leftarrow 0$
13:            end if
14:            $v_w^{t+1} \leftarrow v_w^{t+1} + \zeta$        ▷ Add perturbed noise if gradient smaller than threshold
15:        end for
16:    end for
17:    return $\mathbf{v}^t$              ▷ Returns the vectors corresponding to solution to Optimization 1
18: end procedure
```

Proof. We begin by writing down the objective penalized by a quadratic on the constraints.

$$\mathcal{L}_\rho(\mathbf{v}) := \sum_{P_z \in \mathcal{P}} \tilde{\mathbb{E}}_\mu[P_z(X_z)] + \rho\left[\sum_{P_z \in \mathcal{P}} \sum_{g,g',h,h' \subseteq z : \zeta(g,g') = \zeta(h,h')} \big(\langle v_g, v_{g'}\rangle - \langle v_h, v_{h'}\rangle\big)^2 + \sum_{v_s \in \mathbf{v}} (\|v_s\|^2 - 1)^2\right] \tag{35}$$

For any monomial $X_s = \prod_{i \in s} x_i$ in $P_z(X_z)$ we write

$$\tilde{\mathbb{E}}_\mu[X_s] := \frac{1}{|\mathcal{C}(s)|} \sum_{g,g' \subseteq s : \zeta(g,g') = s} \langle v_g, v_{g'}\rangle \tag{36}$$

Where $\mathcal{C}(s)$ is the size of the set $\{g, g' \subseteq s : \zeta(g,g') = s\}$. In a small abuse of notation, we regard this as the definition of $\tilde{\mathbb{E}}_\mu[X_s]$ but realizet that we're referring to the iterates of the algorithm before they've converged to a pseudoexpectation. Now recall equation 27, we can expand the polynomial $P_z(X_z)$ along its standard monomial basis

$$P_z(X_z) = \sum_{s \subseteq z} w_s X_s \tag{37}$$

where we have defined coefficients w_s for every monomial in $P_z(X_z)$. Plugging equation 36 and equation 37 into equation 35 we obtain

$$(35) = \sum_{P_z \in \mathcal{P}} \sum_{s \subseteq z} w_s \frac{1}{|\mathcal{C}(s)|} \sum_{g,g' \subseteq s : \zeta(g,g') = s} \langle v_g, v_{g'}\rangle + \rho\left[\sum_{P_z \in \mathcal{P}} \sum_{g,g',h,h' \subseteq z : \zeta(g,g') = \zeta(h,h')} \big(\langle v_g, v_{g'}\rangle - \langle v_h, v_{h'}\rangle\big)^2 + \sum_{v_s \in \mathbf{v}} (\|v_s\|^2 - 1)^2\right] \tag{38}$$

Taking the derivative with respect to any $v_w \in \mathbf{v}$ we obtain

$$\frac{\partial \mathcal{L}_\rho(\mathbf{v})}{\partial v_w} = \sum_{P_z \in \mathcal{P} : w \subseteq z} \sum_{s \subseteq z : w \subseteq s} w_s \frac{1}{|\mathcal{C}(s)|} \sum_{w' \subseteq s : \zeta(w,w') = s} v_{w'} + 2\rho\left[\sum_{P_z \in \mathcal{P} : w \subseteq z} \sum_{w',h,h' \subseteq s : \zeta(w,w') = \zeta(h,h')} \big(\langle v_w, v_{w'}\rangle - \langle v_h, v_{h'}\rangle\big) v'_w + (\|v_w\|^2 - 1) v_w\right] \tag{39}$$

The gradient update is then what is detailed in Algorithm 2

$$v_w^{t+1} = v_w^t - \eta \frac{\partial \mathcal{L}_\rho(\mathbf{v})}{\partial v_w} \tag{40}$$

Thus far we have established the form of the gradient. To prove the gradient iteration converges we reference the literature on convergence of perturbed gradient descent (Jin et al., 2017). First we note that the SDP equation 29 has ℓ smooth gradient for $\ell \leq poly(\rho, |\mathcal{P}|, 2^k)$ and has γ lipschitz Hessian for $\gamma = poly(\rho, |\mathcal{P}|, 2^k)$ which we arrive at by bounding the size of every matrix involved in the objective and constraints of Optimization 1. Then by Theorem B.2 the iteration converges to an (ϵ', γ^2)-SOSP in no more than $\tilde{O}(\frac{1}{\epsilon'^2})$ iterations with probability $1 - \delta$. It remains to show that (ϵ', γ^2)-SOSP are approximately global optimum.

Thus far we have worked with the vector version of the SDP which is overparameterized and non-convex. For subsequent analysis we need to define the penalized loss which we denote $\mathcal{H}_\rho(\tilde{\mathbb{E}}[\mathbf{X}])$ in terms of the SDP moment matrix $\tilde{\mathbb{E}}[\mathbf{X}]$.

$$\mathcal{H}_\rho(\tilde{\mathbb{E}}[\mathbf{X}]) := \sum_{P_z\in\mathcal{P}} \tilde{\mathbb{E}}_{\hat{\mu}}[P_z(X_z)] + \rho\left[\sum_{P_z\in\mathcal{P}} \sum_{g,g',h,h'\subseteq z:\zeta(g,g')=\zeta(h,h')} \left(\tilde{\mathbb{E}}_{\hat{\mu}}[X_{g,g'}]-\tilde{\mathbb{E}}_{\hat{\mu}}[X_{h,h'}]\right)^2 + \sum_{X_s:s\subset\mathcal{S}(P),|s|\le k,\forall P\in\mathcal{P}} (\tilde{\mathbb{E}}_{\hat{\mu}}[X_s^2]-1)^2\right] \tag{41}$$

Note that although by definition $\mathcal{H}_\rho(\tilde{\mathbb{E}}[\mathbf{X}]) = \mathcal{L}_\rho(\mathbf{v})$, their gradients and hessians are distinct because $\mathcal{L}_\rho(\mathbf{v})$ is overparameterized.

For the SDP we are working a global optimum clearly exists which we denote $\tilde{\mathbb{E}}_{\tilde{\mu}}[\tilde{\mathbf{X}}]$ with a cholesky decomposition $\tilde{\mathbf{v}}$. Let $\hat{\mathbf{v}}$ be the set of vectors outputted by Algorithm 2 with associated pseudoexpectation $\tilde{\mathbb{E}}_{\hat{\mu}}[\hat{\mathbf{X}}]$. Then, we can bound

$$\mathcal{L}_\rho(\hat{\mathbf{v}}) - \mathcal{L}_\rho(\tilde{\mathbf{v}}) = \mathcal{H}_\rho(\tilde{\mathbb{E}}[\hat{\mathbf{X}}]) - \mathcal{H}_\rho(\tilde{\mathbb{E}}[\tilde{\mathbf{X}}]) \le \left\langle \nabla\mathcal{H}_\rho(\tilde{\mathbb{E}}[\hat{\mathbf{X}}]), \tilde{\mathbb{E}}[\hat{\mathbf{X}}] - \tilde{\mathbb{E}}[\tilde{\mathbf{X}}]\right\rangle \tag{42}$$

Here the first equality is by definition, and the inequality is by the convexity of $\mathcal{H}_\rho$. Moving on, we observe that $\nabla^2\mathcal{L}_\rho(\hat{\mathbf{v}}) \succeq -\gamma\sqrt{\epsilon'}$ implies $\lambda_{\min}(\nabla\mathcal{H}_\rho(\tilde{\mathbb{E}}[\hat{\mathbf{X}}])) \ge -\gamma\sqrt{\epsilon'}$. This derivation can be found in multiple references such as Bhojanapalli et al. (2018). We adapt the lines of their argument in lemma 3 most relevant to our analysis which we detail here for the sake of completeness.

$$\begin{aligned}
\text{equation 42} &\le -\lambda_{\min}(\nabla\mathcal{H}_\rho(\tilde{\mathbb{E}}[\hat{\mathbf{X}}]))\operatorname{Tr}(\tilde{\mathbb{E}}[\hat{\mathbf{X}}]) - \left\langle \nabla\mathcal{H}_\rho(\tilde{\mathbb{E}}[\hat{\mathbf{X}}]), \tilde{\mathbb{E}}[\tilde{\mathbf{X}}]\right\rangle \\
&\le -\lambda_{\min}(\nabla\mathcal{H}_\rho(\tilde{\mathbb{E}}[\hat{\mathbf{X}}]))\operatorname{Tr}(\tilde{\mathbb{E}}[\hat{\mathbf{X}}]) + \|\nabla\mathcal{H}_\rho(\tilde{\mathbb{E}}[\hat{\mathbf{X}}])\|_F\|\tilde{\mathbb{E}}[\tilde{\mathbf{X}}]\|_F \\
&\le \gamma\sqrt{\epsilon'}\operatorname{Tr}(\tilde{\mathbb{E}}[\mathbf{X}]) + \epsilon'\|\tilde{\mathbf{v}}\|_F \\
&\le \gamma\sqrt{\epsilon'}|\mathcal{P}|2^k + \epsilon'|\mathcal{P}|2^k \le \epsilon
\end{aligned} \tag{43}$$

Here the first inequality follows by a standard inequality of frobenius inner product, the second inequality follows by Cauchy-Schwarz, the third inequality follows by the SOSP conditions on both the min eigenvalue of the hessian and the norm of the gradient, the final two inequalities follow from knowing the main diagonal of $\tilde{\mathbb{E}}[\hat{\mathbf{X}}]$ is the identity and that every vector in $\tilde{\mathbf{v}}$ is a unit vector up to inverse polynomial error. For this last point see the proof in Lemma B.2. Therefore if we set $\epsilon' = poly(\epsilon, |\mathcal{P}|, 2^k)$ we arrive at any ϵ error. Therefore we have established our estimate $\hat{v}$ is approximates the global optimum of the quadratically penalized objective i.e $\mathcal{H}_\rho(\tilde{\mathbb{E}}[\hat{\mathbf{X}}]) - \mathcal{H}_\rho(\tilde{\mathbb{E}}[\tilde{\mathbf{X}}]) \le \epsilon$. To finish our proof, we have to bound the distance between the global optimum of the quadratically penalized objective $\mathcal{H}_\rho(\tilde{\mathbb{E}}[\tilde{\mathbf{X}}])$ and OPTSDP(Λ) the optimum of Optimization 1. This is established for ρ a sufficiently large $poly(\epsilon^{-1}, |\mathcal{P}|, 2^k)$ in Lemma B.2. This concludes our proof that the iterates of Algorithm 2 converge to the solution of the SDP Optimization 1. □

The following Lemma B.2 establishes that

Lemma B.2. *Let Λ be a Max-k-CSP instance, and let OPTSDP(Λ) be the optimum of Optimization 1. Let $\mathcal{L}_\rho(\boldsymbol{v})$ be the quadratically penalized objective*

$$\mathcal{L}_\rho(\boldsymbol{v}) := \sum_{P_z\in\mathcal{P}}\sum_{s\subseteq z} w_s \frac{1}{|\mathcal{C}(s)|}\sum_{g,g'\subseteq s:\zeta(g,g')=s}\langle v_g, v_{g'}\rangle + \rho\left[\sum_{P_z\in\mathcal{P}}\sum_{g,g',h,h'\subseteq z:\zeta(g,g')=\zeta(h,h')}\left(\langle v_g, v_{g'}\rangle - \langle v_h, v_{h'}\rangle\right)^2 + \sum_{v_s\in\boldsymbol{v}}(\|v_s\|^2-1)^2\right] \tag{44}$$

Let $\tilde{v}$ be the argmin of the unconstrained minimization

$$\tilde{v} := \underset{\boldsymbol{v} \in \mathbb{R}^{|\mathcal{P}|^2(2^{2k})}}{\arg\min} \; \mathcal{L}_\rho(\boldsymbol{v})$$

Then we have

$$\mathcal{L}_\rho(\tilde{\boldsymbol{v}}) - \mathit{OPTSDP}(\Lambda) \leq \epsilon$$

for $\rho = poly(\frac{1}{\epsilon}, |\mathcal{P}|, 2^k)$

Proof. We begin the analysis with the generic equality constrained semidefinite program of the form

$$\begin{aligned} \text{Minimize:} \quad & \langle C, X \rangle && && (45)\\ \text{Subject to:} \quad & \langle A_i, X \rangle = b_i && \forall i \in \mathcal{F} && (46)\\ & X \succeq 0 && && (47)\\ & X \in \mathbb{R}^{d \times d} && && (48) \end{aligned}$$

For an objective matrix C and constraint matrices $\{A_i\}_{i \in \mathcal{F}}$ in some constraint set $\mathcal{F}$. We will invoke specific properties of Optimization 1 to enable our analysis. First we define the penalized objective in this generic form

$$\mathcal{H}_\rho(X) := \langle C, X \rangle + \rho \sum_{i \in \mathcal{F}} (\langle A_i, X \rangle - b_i)^2$$

Let $\tilde{X}$ be the minimizer of the penalized problem.

$$\tilde{X} := \underset{X \in \mathbb{R}^{d \times d}}{\arg\min} \mathcal{L}_\rho(X)$$

Let X^* be the minimizer of the constrained problem equation 61. Let τ_i be the error $\tilde{X}$ has in satisfying constraint $\langle A_i, \tilde{X} \rangle = b_i$.

$$\tau_i := |\langle A_i, \tilde{X} \rangle - b_i|$$

We will show that τ_i scales inversely with ρ. That is, $\tau_i \leq poly(|\mathcal{P}|, 2^k, \rho^{-1})$.

Notice that the quadratic penalty on the violated constraints must be smaller than the decrease in the objective for having violated the constraints. So long as the objective is not too sensitive 'robust' to perturbations in the constraint violations the quadratic penalty should overwhelm the decrease in the objective. To carry out this intuition, we begin with the fact that the constrained minimum is larger than the penalized minimum.

$$\mathcal{H}_\rho(X^*) - \mathcal{H}_\rho(\tilde{X}) \leq 0 \tag{49}$$

This implies

$$\langle C, X^* \rangle - (\langle C, \tilde{X} \rangle + \rho \sum_{i \in \mathcal{F}} \tau_i^2) \leq 0 \tag{50}$$

Rearranging LHS and RHS we obtain

$$\rho \sum_{i \in \mathcal{F}} \tau_i^2 \leq \langle C, \tilde{X} - X^* \rangle \tag{51}$$

We know the RHS is upper bounded

$$\rho \sum_{i \in \mathcal{F}} \tau_i^2 \leq \langle C, \tilde{X} - X^* \rangle \leq \sum_{i \in \mathcal{F}} \tau_i poly(k, q) \tag{52}$$

The last line follows from the robustness theorem of Steurer (2010) lemma 3.4 that states for an SDP solution that violates the constraints by a small perturbation changes the objective by a small amount. Then taking Cauchy-Schwarz of the RHS we further bound by

$$\rho \sum_{i \in \mathcal{F}} \tau_i^2 \leq \sqrt{|\mathcal{F}| \sum_{i \in \mathcal{F}} \tau_i^2} poly(k, q)$$

Rearranging left and right hand sides we obtain

$$\sum_{i\in\mathcal{F}} \tau_i^2 \le \rho^{-1} poly(k,q)|\mathcal{F}|$$

which implies $\|\tau\| = poly(|\mathcal{P}|, 2^k, \rho^{-1})$. Moving on, consider the dual feasibility condition

$$C = Q + \sum_{i\in\mathcal{F}} \lambda_i A_i$$

for some $Q \succeq 0$. Then we have

$$\langle C, X^* - \tilde{X}\rangle = \langle Q, X^*\rangle - \langle Q, \tilde{X}\rangle + \sum_{i\in\mathcal{F}} \lambda_i \langle A_i, X^* - \bar{X}\rangle$$

By complementary slackness $\langle Q, X^*\rangle = 0$ so we obtain

$$= -\langle Q, \tilde{X}\rangle + \sum_{i\in\mathcal{F}} \lambda_i \langle A_i, X^* - \bar{X}\rangle$$

By PSD'ness of both Q and $\tilde{X}$ we upper bound by

$$\le \sum_{i\in\mathcal{F}} \lambda_i \langle A_i, X^* - \tilde{X}\rangle = \sum_{i\in\mathcal{F}} \lambda_i (b_i - \langle A_i, \tilde{X}\rangle) \le \sqrt{\sum_{i\in\mathcal{F}} \lambda_i^2}\|\tau\|$$

Where in the first equality we used the fact that $\langle A_i, \bar{X}\rangle = b_i$, and the second inequality is Cauchy-Schwarz. Since we've already established that $\|\tau\| \propto \rho^{-1}$ we must simply bound the size of the dual variables λ_i. To bound the size of λ_i, we separate the constraints A_i into the diagonal constraints $\{F_i\}_{i\in\mathcal{W}}$ and equality constraints $\{G_i\}_{i\in\mathcal{R}}$ where

$$\langle F_i, X\rangle = 1 \quad \forall i \in \mathcal{W} \qquad \langle G_i, X\rangle = 0 \quad \forall i \in \mathcal{R}$$

The dual takes on the following form for $\delta, \eta \in \mathbb{R}$

$$\textbf{Maximize: } \sum_{i\in\mathcal{W}} \delta_i \tag{53}$$

$$\textbf{Subject to: } C - \sum_{i\in\mathcal{W}} \delta_i F_i - \sum_{i\in\mathcal{R}} \eta_i G_i \succeq 0 \tag{54}$$

Where we've split the dual variables $\{\lambda_i\}_{i\in\mathcal{F}}$ into two sets $\{\delta_i\}_{i\in\mathcal{W}}$ and $\{\eta_i\}_{i\in\mathcal{R}}$. Note that the δ_i are polynomially bounded i.e $|\delta_i| \le poly(|\mathcal{P}|, 2^k)$. Assume the contrary, if $\delta_i > poly(|\mathcal{P}|, 2^k)$ then the objective is polynomially unbounded which contradicts dual objective being smaller than primal objective. If $\delta_i < -poly(\mathcal{P}, 2^k)$ then the $i'th$ diagonal coordinate of equation 54 is polynomially unbounded and then e_i is a negative eigenvalue of equation 54 which is a contradiction of PSD'ness. Therefore, the δ_i are polynomially bounded. To demonstrate the $\{\eta_i\}_{i\in\mathcal{R}}$ are polynomially bounded, note that because of linear independence of the constraints plus the minimum singular value being greater than a constant, there exists a setting of the η that is polynomially bounded such that the dual feasibility constraint is satisfied. Since the η do not appear in the objective, finding a setting that satisfies equation 54 suffices.

Constraint matrix is well conditioned. The smallest singular value of $\{A_i\}_{i\in\mathcal{F}}$ is a constant. This is a technical observation about collections of vectors of the form $\{e_1 + e_j\}_{j\in[2,T]}$ where e_i is the $i'th$ standard basis vector. Any unit vector v satisfies $\|\sum_j v_j(e_1 + e_j)\| = (\sum_j v_j)^2 + \sum_j v_j^2 \ge 1$. □

Finally we show it's not hard to generalize our algorithm to alphabets of size $[q]$.

Algorithm 3 SDP Vector Formulation for Max-k-CSP General Alphabet

```
1: procedure SDP VECTOR FORMULATION GENERAL ALPHABET($\Lambda = (\mathcal{V}, \mathcal{P}, q)$)    ▷ SDP Equivalent to UGC optimal
       Minimize: $\sum_{P_z \subset \mathcal{P}} \tilde{\mathbb{E}}_\mu[-P_z(X_z)]$    (55)
2:                                                   ▷ Pseudoexpectation formulation of objective
       Subject to: $\tilde{\mathbb{E}}_\mu[(x_{(i,a)}^2 - x_{(i,a)}) \prod_{(j,b)\in\phi} x_{(j,b)}] = 0 \quad \forall i \in \mathcal{V}, \forall a \in [q], \forall \phi \subseteq \mathcal{D}(P), \forall P \in \mathcal{P}$    (56)
                   $\tilde{\mathbb{E}}_\mu[(\sum_{a\in[q]} x_{ia} - 1) \prod_{(j,b)\in\phi} x_{(j,b)}] = 0 \quad \forall i \in \mathcal{V}, \forall \phi \subseteq \mathcal{D}(P), \forall P \in \mathcal{P}$    (57)
                   $\tilde{\mathbb{E}}_\mu[x_{(i,a)} x_{(i,a')} \prod_{(j,b)\in\phi} x_{(j,b)}] = 0 \quad \forall i \in \mathcal{V}, \forall a \neq a' \in [q], \forall \phi \subseteq \mathcal{D}(P), \forall P \in \mathcal{P}$    (58)
                   $\tilde{\mathbb{E}}[SoS_{2kq}(X_\phi)] \geq 0 \quad \forall \phi \subseteq \mathcal{D}(P), \forall P \in \mathcal{P}$    (59)
                   $\tilde{\mathbb{E}}[SoS_2(\mathbf{X})] \geq 0$    (60)
                                                   ▷ First constraint corresponds to booleanity
3:             ▷ Second constraint corresponds to a variable taking on only one value in the alphabet
4:             ▷ Third constraint corresponds to a variable taking on only one value in the alphabet
5:                                                   ▷ Fourth constraint corresponds to
6:             ▷ Fifth constraint correpsonds to the positivity of every degree two sum of squares of polynomials
7: end procedure
```

Notation for General Alphabet. For any predicate $P \in \mathcal{P}$, let $\mathcal{D}(P)$ be the set of all variable assignment tuples indexed by a set of variables $s \subseteq \mathcal{S}(P)$ and an assignment $\tau \in [q]^{|s|}$. Let $x_{(i,a)}$ denote an assignment of value $a \in [q]$ to variable x_i.

Lemma B.3. *There exists a message passing algorithm that computes in* $poly(1/\epsilon, |\mathcal{P}|, 2^k, \log(\delta^{-1}))$ *iterations a set of vectors* $\mathbf{v} := \{\hat{v}_{(i,a)}\}$ *for all* $(i,a) \in \phi$*, for all* $\phi \subseteq \mathcal{D}(P)$*, for all* $P \in \mathcal{P}$ *that satisfy the constraints of Algorithm 3 to error* ϵ *and approximates the optimum of Algorithm 3 to error* ϵ *with probability* $1 - \delta$

$$\Big|\sum_{P_z \in \mathcal{P}} \tilde{\mathbb{E}}_{\hat{\mu}}[P_z(X_z)] - OPTSDP(\Lambda)\Big| \leq \epsilon$$

where $OPTSDP(\Lambda)$ *is the optimum of Algorithm 3.*

Proof. The proof is entirely parallel to the proof of Theorem B.1. We can write Algorithm 3 entirely in terms of the vector of its cholesky decomposition where once again we take advantage of the fact that SoS degree $2kq$ distributions are actual distributions over subsets of kq variables over each predicate. Given the overparameterized vector formulation, we observe that once again we are faced with equality constraints that can be added to the objective with a quadratic penalty. Perturbed gradient descent induces a message passing algorithm over the constraint graph G_Λ, and in no more than $poly(\frac{1}{\epsilon}, |P|, q^k)$ iterations reaches an (ϵ, γ)-SOSP. The analysis of optimality goes along the same lines as Lemma B.2. For sufficiently large penalty $\rho = poly(\frac{1}{\epsilon}, |P|, q^k)$ the error in satisfying the constraints is ϵ and the objective is robust to small perturbations in satisfying the constraint. That concludes our discussion of generalizing to general alphabets. □

B.1 Neural Certification Scheme

An intriguing aspect of OptGNN is that the embeddings can be interpreted as the solution to a low rank SDP which leaves open the tantalizing possibility that the embeddings can be used to generate a dual certificate i.e a lower bound on a convex relaxation. First we define the primal problem

$$\textbf{Minimize:} \quad \langle C, X\rangle \tag{61}$$

$$\textbf{Subject to:} \quad \langle A_i, X\rangle = b_i \qquad \forall i \in [\mathcal{F}] \tag{62}$$

$$X \succeq 0 \tag{63}$$

Lemma B.4. *Let OPT be the minimizer of the SDP equation 61. Then for any $\tilde{X} \in \mathbb{R}^{N\times N} \succeq 0$ and any $\lambda^* \in \mathbb{R}^{|\mathcal{F}|}$, we define $F_{\lambda^*}(X)$ to be*

$$F_{\lambda^*}(\tilde{X}) := \langle C, \tilde{X}\rangle + \sum_{i\in\mathcal{F}} \lambda_i^*(\langle A_i, \tilde{X}\rangle - b_i)$$

We require SDP to satisfy a bound on its trace $Tr(X) \leq \mathcal{Y}$ for some $\mathcal{Y} \in \mathbb{R}^+$. Then the following is a lower bound on OPT.

$$OPT \geq F_{\lambda^*}(\tilde{X}) - \langle \nabla F_{\lambda^*}(\tilde{X}), \tilde{X}\rangle + \lambda_{min}(\nabla F_{\lambda^*}(\tilde{X}))\mathcal{Y}$$

Proof. Next we introduce lagrange multipliers $\lambda \in \mathbb{R}^k$ and $Q \succeq 0$ to form the lagrangian

$$\mathcal{L}(\lambda, Q, X) = \langle C, X\rangle + \sum_{i\in\mathcal{F}} \lambda_i(\langle A_i, X\rangle - b_i) - \langle Q, X\rangle$$

We lower bound the optimum of OPT defined to be the minimizer of equation 61

$$OPT := \min_{X\succeq 0} \max_{\lambda\in\mathbb{R}, Q\succeq 0} \mathcal{L}(\lambda, Q, X)$$

$$\geq \min_{V\in\mathbb{R}^{N\times N}} \max_{\lambda} \langle C, VV^T\rangle + \sum_{i\in\mathcal{F}} \lambda_i(\langle A_i, VV^T\rangle - b_i)$$

$$\geq \max_{\lambda} \min_{V\in\mathbb{R}^{N\times N}} \langle C, VV^T\rangle + \sum_{i\in\mathcal{F}} \lambda_i(\langle A_i, VV^T\rangle - b_i) \tag{64}$$

$$\geq \min_{V\in\mathbb{R}^{N\times N}} \langle C, VV^T\rangle + \sum_{i\in\mathcal{F}} \lambda_i^*(\langle A_i, VV^T\rangle - b_i) \tag{65}$$

Where in the first inequality we replaced $X \succeq 0$ with VV^T which is a lower bound as every psd matrix admits a cholesky decomposition. In the second inequality we flipped the order of min and max, and in the final inequality we chose a specific set of dual variables $\lambda^* \in \mathbb{R}^{|\mathcal{F}|}$ which lower bounds the maximization over dual variables. The key is to find a good setting for λ^*.

Next we establish that for any choice of λ^* we can compute a lower bound on equation 65 as follows. Let $F_{\lambda^*}(VV^T)$ be defined as the funciton in the RHS of equation 65.

$$F_{\lambda^*}(VV^T) := \langle C, X\rangle + \sum_{i\in\mathcal{F}} \lambda_i^*(\langle A_i, X\rangle - b_i)$$

Then equation 65 can be rewritten as

$$OPT \geq \min_{V\in\mathbb{R}^{N\times N}} F_{\lambda^*}(VV^T) := \langle C, X\rangle + \sum_{i\in\mathcal{F}} \lambda_i^*(\langle A_i, X\rangle - b_i)$$

Now let V^* be the minimizer of equation 65 and let $X^* = V^*(V^*)^T$. We have by convexity that

$$F_{\lambda^*}(X) - F_{\lambda^*}(X^*) \leq \langle \nabla F_{\lambda^*}(X), X - X^*\rangle = \langle \nabla F_{\lambda^*}(X), X\rangle + \langle -\nabla F_{\lambda^*}(X), X^*\rangle \tag{66}$$

$$\leq \langle \nabla F_{\lambda^*}(X), X\rangle - \lambda_{min}(\nabla F_{\lambda^*}(X))\text{Tr}(X^*) \tag{67}$$

$$\leq \langle \nabla F_{\lambda^*}(X), X\rangle - \lambda_{min}(\nabla F_{\lambda^*}(X))N \tag{68}$$

In the first inequality we apply the convexity of F_{λ^*}. In the second inequality we apply a standard inequality of frobenius inner product. In the last inequality we use the fact that $\text{Tr}(X^*) = N$. Rearranging we obtain for any X

$$OPT \geq F_\lambda(X^*) \geq F_{\lambda^*}(X) - \langle \nabla F_{\lambda^*}(X), X\rangle + \lambda_{min}(\nabla F_{\lambda^*}(X))N \tag{69}$$

Therefore it suffices to upper bound the two terms above $\langle \nabla F_{\lambda^*}(X), X\rangle$ and $\lambda_{min}(\nabla F_{\lambda^*}(X))$ which is an expression that holds for any X. Given the output embeddings $\tilde{V}$ of OptGNN (or indeed any set of vectors $\tilde{V}$) let $\tilde{X} = \tilde{V}\tilde{V}^T$. Then we have concluded

$$OPT \geq F_\lambda(X^*) \geq F_{\lambda^*}(\tilde{X}) - \langle \nabla F_{\lambda^*}(\tilde{X}), \tilde{X}\rangle + \lambda_{min}(\nabla F_{\lambda^*}(\tilde{X}))N \tag{70}$$

as desired. □

Up to this point, every manipulation is formal proof. Subsequently we detail how to make an educated 'guess' of the dual variables λ^*. Although any guess will produce a bound, it won't produce a tight bound. To be clear, solving for the optimal λ^* would be the same as building an SDP solver which would bring us back into the expensive primal dual procedures that are involved in solving SDP's. We are designing quick and cheap ways to output a dual certificate that may be somewhat looser. Our scheme is simply to set λ^* such that $\|\nabla F_{\lambda^*}(\tilde{X})\|$ is minimized, ideally equal to zero. The intuition is that if $(\tilde{X}, \lambda^*)$ were a primal dual pair, then the lagrangian would have a derivative with respect to X evaluated at $\tilde{X}$ would be equal to zero. Let $H_\lambda(V)$ be defined as follows

$$H_{\lambda^*}(\tilde{V}) := \langle C, \tilde{V}\tilde{V}^T \rangle + \sum_{i \in \mathcal{F}} \lambda_i^* (\langle A_i, \tilde{V}\tilde{V}^T \rangle - b_i)$$

We know the gradient of $H_\lambda(\tilde{V})$

$$\nabla H_\lambda(\tilde{V}) = 2(C + \sum_{i \in \mathcal{F}} \lambda_i^* A_i)\tilde{V} = 2\nabla F_\lambda(\tilde{V}\tilde{V}^T)\tilde{V}$$

Therefore it suffices to find a setting of λ^* such that $\|\nabla F_\lambda(\tilde{X})\tilde{V}\|$ is small, ideally zero. This would be a simple task, indeed a regression, if not for the unfortunate fact that OptGNN explicitly projects the vectors in $\tilde{V}$ to be unit vectors. This creates numerical problems such that minimizing the norm of $\|\nabla F_\lambda(\tilde{X})\tilde{V}\|$ does not produce a $\nabla F_\lambda(\tilde{X})$ with a large minimum eigenvalue.

To fix this issue, let $R_{\eta,\rho}(V)$ denote the penalized lagrangian with quadratic penalties for constraints of the form $\langle A_i, X \rangle = b_i$ and linear penalty η_i for constraints along the main diagonal of X of the form $\langle e_i e_i^T, X \rangle = 1$.

$$R_{\eta,\rho}(V) := \langle C, VV^T \rangle + \sum_{i \in \mathcal{J}} \rho(\langle A_i, VV^T \rangle - b_i)^2 + \sum_{i=1}^{N} \eta_i(\langle e_i e_i^T, VV^T \rangle - 1)$$

Taking the gradient of $R_{\eta,\rho}(V)$ we obtain

$$\nabla R_{\eta,\rho}(V) := 2CV + \sum_{i \in \mathcal{J}} 2\rho(\langle A_i, VV^T \rangle - b_i)A_i V + \sum_{i=1}^{N} 2\eta_i e_i e_i^T V$$

Our rule for setting dual variables δ_i for $i \in \mathcal{J}$ is

$$\delta_i := 2\rho(\langle A_i, \tilde{V}\tilde{V}^T \rangle - b_i)$$

our rule for setting dual variables η_j for $j \in [N]$ is

$$\eta_j := \frac{1}{2}\|e_j^T(C + \sum_{i \in \mathcal{F}} 2\rho(\langle A_i, VV^T \rangle - b_i)A_i)V\|$$

Then our full set of dual variables λ^* is simply the concatenation (δ, η). Writing out everything explicitly we obtain the following matrix for $\nabla F_{\lambda^*}(\tilde{V}\tilde{V}^T)$

$$\nabla F_\lambda(\tilde{V}\tilde{V}^T) = C + \sum_{i \in \mathcal{F}} \rho(\langle A_i, \tilde{V}\tilde{V}^T \rangle - b_i)A_i + \sum_{j \in [N]} \frac{1}{2}\|e_j^T(C + \sum_{i \in \mathcal{F}} 2\rho(\langle A_i, \tilde{V}\tilde{V}^T \rangle - b_i)A_i)\tilde{V}\| e_i e_i^T$$

Plugging this expression into Lemma B.4 the final bound we evaluate in our code is

$$\begin{aligned} OPT \geq\ & \langle C, \tilde{V}\tilde{V}^T \rangle + \sum_{i \in \mathcal{F}} 2\rho(\langle A_i, \tilde{V}\tilde{V}^T \rangle - b_i)^2 \\ & - \Big\langle C + \sum_{i \in \mathcal{F}} \rho(\langle A_i, \tilde{V}\tilde{V}^T \rangle - b_i)A_i + \sum_{j \in [N]} \frac{1}{2}\|e_j^T(C + \sum_{i \in \mathcal{F}} 2\rho(\langle A_i, \tilde{V}\tilde{V}^T \rangle - b_i)A_i)\tilde{V}\| e_i e_i^T, \tilde{V}\tilde{V}^T \Big\rangle \\ & + \lambda_{min}\Big(C + \sum_{i \in \mathcal{F}} \rho(\langle A_i, \tilde{V}\tilde{V}^T \rangle - b_i)A_i + \sum_{j \in [N]} \frac{1}{2}\|e_j^T(C + \sum_{i \in \mathcal{F}} 2\rho(\langle A_i, \tilde{V}\tilde{V}^T \rangle - b_i)A_i)\tilde{V}\| e_i e_i^T\Big)N \end{aligned} \tag{71}$$

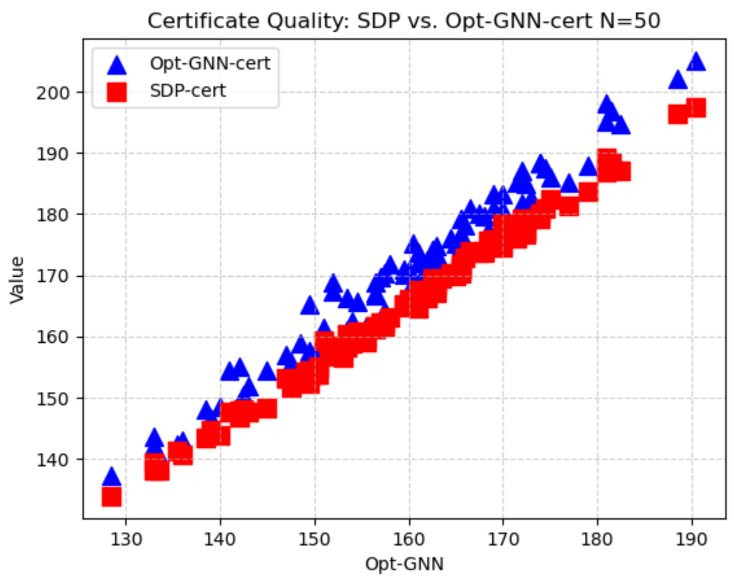

Figure 2: N=50 p=0.1 SDP vs Opt-GNN Dual Certificate

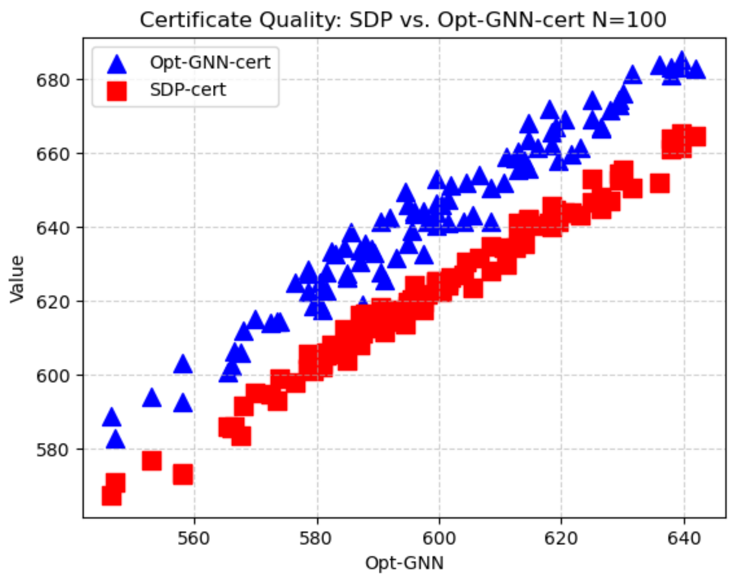

Figure 3: N=100 p=0.1 SDP vs Opt-GNN Dual Certificate

Which is entirely computed in terms of $\tilde{V}$ the output embeddings of OptGNN. The resulting plot is as follows.

Note: The reason for splitting the set of dual variables is because the projection operator onto the unit ball is hard coded into the architecture of the lift network. Satisfying the constraint set via projection is different from the soft quadratic penalties on the remaining constraints and require separate handling.

Max Cut Certificate For Max Cut our dual variables are particularly simple as there are no constraints $\langle A_i, X\rangle = b_i$ for $b_i \neq 0$. The dual variables for Max Cut take on the form for all $i \in [N]$

$$\lambda_i^* = \frac{1}{2}\|\sum_{j\in N(i)} w_{ij}v_j\|$$

It's certainly possible to come up with tighter certification schemes which we leave to future work.

Intuition: Near global optimality one step of the augmented method of lagrange multipliers ought to closely approximate the dual variables. After obtaining a guess for the penalized lagrange multipliers we estimate the lagrange multipliers for the norm constraint by approximating $\nabla R_\lambda(V) = 0$. The alternative would have been to solve the linear system for all the lagrange multipliers at once but this runs into numerical issues and degeneracies explained below. We run out certification procedure which we name Opt-GNN-cert and compare it to the SDP certificate. Note, that mathematically we will always produce a larger (i.e inferior) dual certificate in comparison to the SDP because we are bounding the distance to the SDP optimum with error in the gradients and hessians of the output embeddings of OptGNN. Our advantage is in the speed of the procedure. Without having to go through a primal dual solver, the entire time of producing Opt-GNN-cert is in the time required to feedforward through Opt-GNN. In this case we train an Opt-GNN-MaxCut with 10 layers, on 1000 Erdos-Renyi graphs, with $N = 100$ nodes and edge density $p = 0.1$. We plot the Opt-GNN Max Cut value (an actual integer cut) on the x-axis and in the y-axis we plot the dual certificate value on the same graph where we compare the SDP certificate with the Opt-GNN-cert. See 2 for the $N = 50$ graphs and 3 for the $N = 100$ graphs.

Note of course the dual certificate for any technique must be larger than the cut value outputted by Opt-GNN so the scatter plot must be above the $x = y$ axis of the plot. We see as is mathematically necessary, the Opt-GNN-cert is not as tight as the SDP certificate but certainly competitive and more importantly it is arrived at dramatically faster. Without any attempt at optimizing the runtime, the Opt-GNN feedforward and certification takes no more than 0.02 seconds whereas the SDP takes 0.5 seconds on $N = 100$ node graphs.

B.2 Miscellaneous Lemmas

Theorem B.2 (perturbed-gd Jin et al. (2017)). *Let f be ℓ-smooth (that is, it's gradient is ℓ-Lipschitz) and have a γ-Lipschitz Hessian. There exists an absolute constant c_{max} such that for any $\delta \in (0,1), \epsilon \leq \frac{\ell^2}{\gamma}, \Delta_f \geq f(X_0) - f^*$, and constant $c \leq c_{max}$, $PGD(X_0, \ell, \gamma, \epsilon, c, \delta, \Delta_f)$ applied to the cost fucntion f outputs a (γ^2, ϵ) SOSP with probability at least $1 - \delta$ in*

$$O\Big(\frac{(f(X_0) - f^*)\ell}{\epsilon^2}\log^4(\frac{nk\ell\Delta_f}{\epsilon^2\delta})\Big)$$

iterations.

Definition B.1 ((γ, ϵ)-second order stationary point). *A (γ, ϵ) second order stationary point of a function f is a point x satisfying*

$$\|\nabla f(x)\| \leq \epsilon$$

$$\lambda_{min}(\nabla^2 f(x)) \geq -\sqrt{\gamma\epsilon}$$

C Experiment details

In this section we give further details on our experimental results.

C.1 Hardware

Our training runs used 20 cores of an Intel Xeon Gold 6248 (for data loading and random graph generation) and a NVIDIA Tesla V100 GPU. Our Gurobi runs use 8 threads on a Intel Xeon Platinum

8260. Our KaMIS runs use an Intel Core i9-13900H. Our LwD and DGL-TREESEARCH runs use an Intel Core i9-13900H and an RTX 4060.

Parameter	Generated	**TU-small**	**TU-REDDIT**
Gradient steps	20,000	100,000	100,000
Validation freq	1,000	1,000	2,000
Batch size	16	16	16
Ranks	4, 8, 16, 32	4, 8, 16, 32	4, 8, 16, 32
Layer counts	8, 16	8, 16	8, 16
Positional encodings	RW	LE, RW	RW
Run count	8	16	8

Table 4: Hyperparameter range explored for each group of datasets. For each NN architecture, when training on a dataset, we explored every listed hyperparameter combination in the corresponding column.

C.2 HYPERPARAMETERS

We ran each experiment on a range of hyperparameters. See Table 4 for the hyperparameter listing. For all training runs, we used the Adam optimizer Kingma & Ba (2014) with a learning rate of 0.001. We used Laplacian eigenvector Dwivedi et al. (2020) (LE) or random walk Dwivedi et al. (2021) (RW) positional encoding with dimensionality of half the rank, except for rank 32 where we used 8 dimensions.

Dataset	OptGNN
BA[a] (50,100)	0.998 ± 0.002
BA[a] (100,200)	0.996 ± 0.003
BA[a] (400,500)	0.993 ± 0.003
ER[a] (50,100)	0.998 ± 0.002
ER[a] (100,200)	0.996 ± 0.002
ER[a] (400,500)	0.993 ± 0.001
HK[a] (50,100)	0.998 ± 0.002
HK[a] (100,200)	0.995 ± 0.003
HK[a] (400,500)	0.994 ± 0.003
WC[a] (50,100)	0.998 ± 0.003
WC[a] (100,200)	0.995 ± 0.003
WC[a] (400,500)	0.989 ± 0.003
MUTAG[b]	1.000 ± 0.000
ENZYMES[b]	0.999 ± 0.003
PROTEINS[b]	1.000 ± 0.002
IMDB-BIN[b]	1.000 ± 0.001
COLLAB[b]	0.999 ± 0.002
REDDIT-BIN[c]	1.000 ± 0.001
REDDIT-M-12K[c]	0.999 ± 0.002
REDDIT-M-5K[c]	0.999 ± 0.002

Table 5: Performance of OptGNN compared to Gurobi running under an 8 second time limit, expressed as a ratio. For each dataset, we take the ratio of the integral values achieved by OptGNN and Gurobi 8s on each of the graphs in the test slice. We present the average and standard deviation of these ratios. Here, higher is better. This table demonstrates that OptGNN achieves nearly the same performance, missing on average 1.1% of the cut value in the worst measured case.

C.3 RATIO TABLES

In Table 5 and Table 6 we supply the performance of OptGNN as a ratio against the integral value achieved by Gurobi running with a time limit of 8 seconds. These tables include the standard deviation in the ratio. We note that for Maximum Cut, OptGNN comes within 1.1% of the Gurobi 8s value, and for minimum Vertex Cover, OptGNN comes within 3.1%.

C.4 VERTEX COVER ALTERNATIVE ARCHITECTURES

Table 7 presents the performance of alternative neural network architectures on minimum vertex cover.

Dataset	OptGNN
BA[a] (50,100)	1.001 ± 0.005
BA[a] (100,200)	1.003 ± 0.005
BA[a] (400,500)	1.008 ± 0.011
ER[a] (50,100)	1.010 ± 0.015
ER[a] (100,200)	1.031 ± 0.012
ER[a] (400,500)	1.013 ± 0.006
HK[a] (50,100)	1.002 ± 0.007
HK[a] (100,200)	1.004 ± 0.013
HK[a] (400,500)	1.007 ± 0.011
WC[a] (50,100)	1.014 ± 0.016
WC[a] (100,200)	1.016 ± 0.013
WC[a] (400,500)	1.018 ± 0.007
MUTAG[b]	1.009 ± 0.027
ENZYMES[b]	1.000 ± 0.000
PROTEINS[b]	1.010 ± 0.021
IMDB-BIN[b]	1.002 ± 0.016
COLLAB[b]	1.001 ± 0.003
REDDIT-BIN[c]	1.000 ± 0.002
REDDIT-M-12K[c]	1.000 ± 0.001
REDDIT-M-5K[c]	1.000 ± 0.001

Table 6: Performance of OptGNN compared to Gurobi running under an 8 second time limit, expressed as a ratio. For each dataset, we take the ratio of the integral values achieved by OptGNN and Gurobi 8s on each of the graphs in the test slice. We present the average and standard deviation of these ratios. Here, lower is better. This table demonstrates that OptGNN achieves nearly the same performance, producing a cover on average 3.1% larger than Gurobi 8s in the worst measured case.

Dataset	GAT	GCNN	GIN	GatedGCNN	OptGNN
ER[a] (50,100)	58.78 (20)	64.42 (23)	64.18 (20)	56.17 (14)	**55.25** (21)
ER[a] (100,200)	129.47 (20)	141.94 (17)	140.06 (20)	130.32 (20)	**126.52** (18)
ER[a] (400,500)	443.93 (43)	444.12 (33)	442.11 (31)	440.90 (28)	**420.70** (41)
MUTAG[b]	**7.79** (19)	8.11 (16)	7.95 (20)	**7.79** (17)	**7.79** (18)
ENZYMES[b]	21.93 (24)	25.42 (18)	25.80 (28)	20.28 (14)	**20.00** (24)
PROTEINS[b]	28.19 (23)	31.07 (19)	32.28 (21)	**25.25** (19)	25.29 (18)
IMDB-BIN[b]	17.62 (21)	19.22 (19)	19.03 (23)	16.79 (15)	**16.78** (18)
COLLAB[b]	68.23 (23)	73.32 (17)	73.82 (26)	72.92 (13)	**67.50** (23)

Table 7: Performance of various model architectures compared to OptGNN for selected datasets on Minimum Vertex Cover. Here, lower is better.

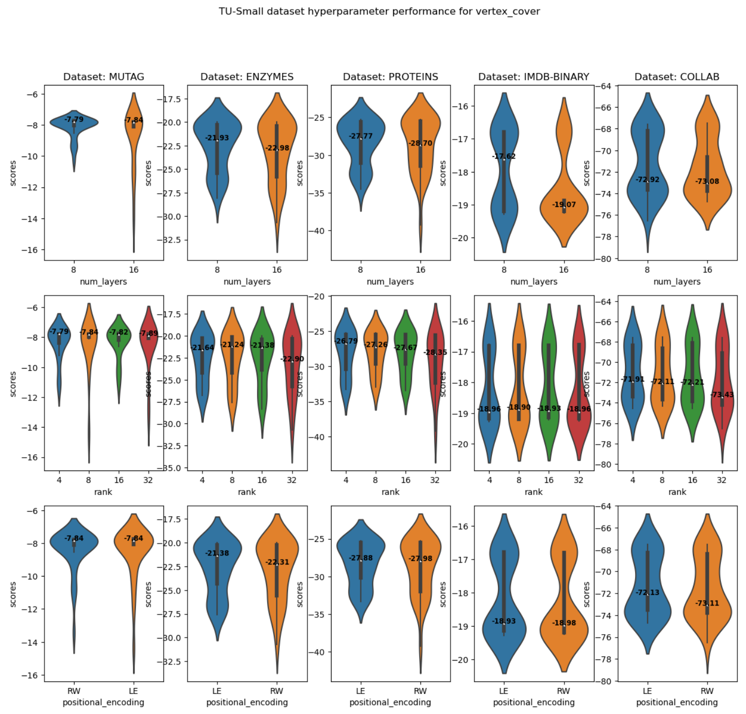

Figure 4: Trends in model performance with respect to the number of layers, hidden size, and positional encoding of the models.

C.5 Effects of hyperparameters on performance

Figure 4, Figure 5, Figure 6, and Figure 7 present overall trends in model performance across hyperparameters.

Train Dataset	MUTAG	ENZYMES	PROTEINS	IMDB-BIN	COLLAB
BA (50,100)	**7.74**	20.12	27.66	17.57	74.15
BA (100,200)	**7.74**	20.35	26.03	16.86	69.29
BA (400,500)	8.05	21.00	26.54	17.34	70.17
ER (50,100)	**7.74**	20.37	28.17	16.86	69.07
ER (100,200)	8.05	21.52	27.72	16.89	68.83
ER (400,500)	7.79	21.55	28.60	16.78	68.74
HK (50,100)	**7.74**	20.42	25.60	17.05	69.17
HK (100,200)	7.84	20.43	27.30	17.01	70.20
HK (400,500)	7.95	20.63	26.30	17.15	69.91
WC (50,100)	7.89	**20.13**	25.46	17.38	70.14
WC (100,200)	7.79	20.30	25.45	17.91	71.16
WC (400,500)	8.05	20.48	25.79	17.12	70.16
MUTAG	**7.74**	20.83	26.76	16.92	70.09
ENZYMES	**7.74**	20.60	28.29	16.79	68.40
PROTEINS	7.89	20.22	**25.29**	16.77	70.26
IMDB-BIN	7.95	20.97	27.06	**16.76**	68.03
COLLAB	7.89	20.35	26.13	**16.76**	**67.52**

Table 8: Models for Vertex Cover trained on ”dataset” were tested on a selection of the TU datasets (ENZYMES, PROTEINS, MUTAG, IMDB-BINARY, and COLLAB). We observe that the performance of the models generalizes well even when they are taken out of their training context.

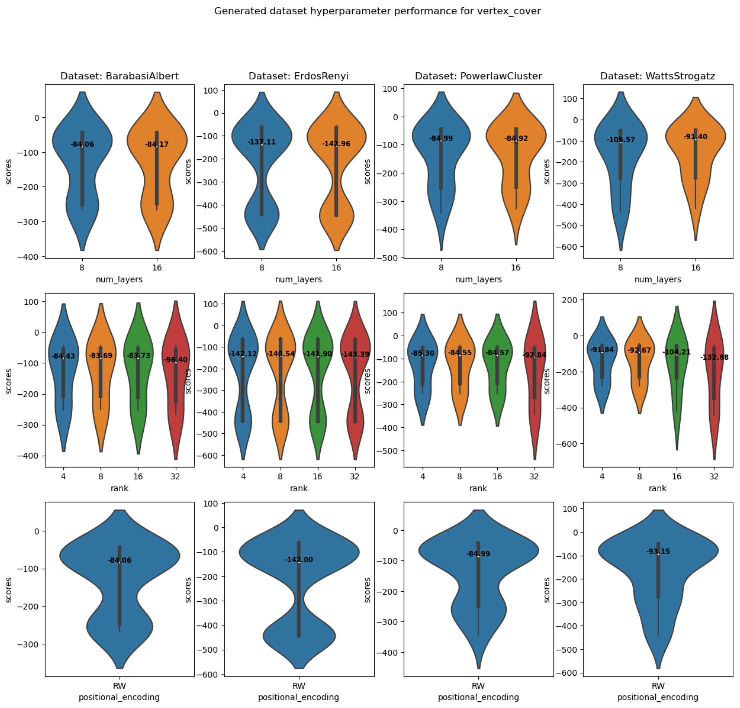

Figure 5: Trends in model performance with respect to the number of layers, hidden size, and positional encoding of the models.

C.6 Generalizability

Models trained on one dataset work quite well on other datasets, suggesting that models have good ability to generalize to examples outside their training distribution. Please see Table 8.

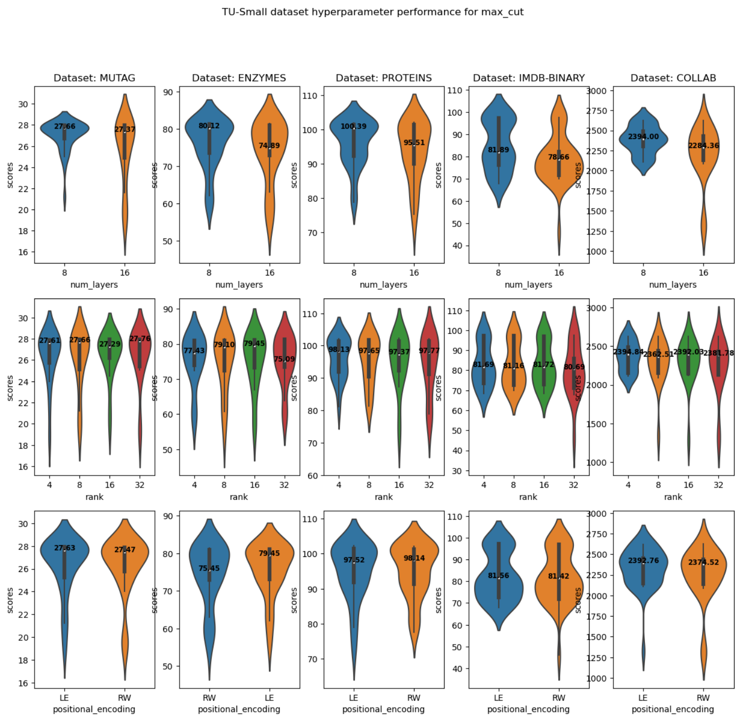

Figure 6: Trends in model performance with respect to the number of layers, hidden size, and positional encoding of the models.

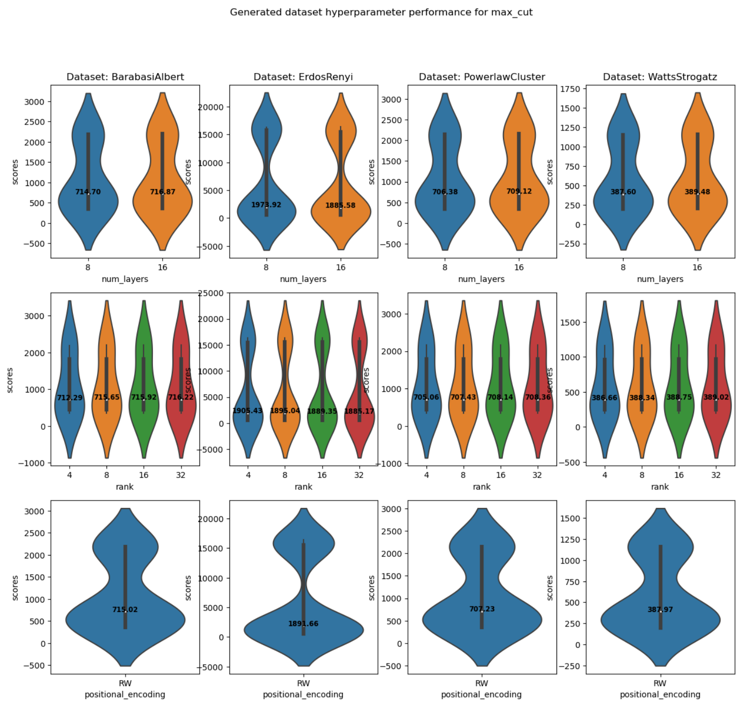

Figure 7: Trends in model performance with respect to the number of layers, hidden size, and positional encoding of the models.