Solving Max-Cut to Global Optimality via Feasibility-Preserving Graph Neural Networks

Hao Chen[1*] **Chendi Qian**[2*] **Christopher Morris**[2] **Andrea Lodi**[3] **Can Li**[1]
[1]Davidson School of Chemical Engineering, Purdue University
[2]Faculty of Computer Science, RWTH Aachen University
[3]Jacobs Technion-Cornell Institute, Cornell Tech
{chen4433, canli}@purdue.edu
{chendi.qian@log, morris@cs}.rwth-aachen.de
andrea.lodi@cornell.edu

Abstract

Exact solutions to hard combinatorial optimization problems often rely on strong convex relaxations, but repeatedly solving these relaxations inside a branch-and-bound algorithm can be prohibitively expensive. We focus on the *Max-Cut problem* (Max-Cut), where branch-and-bound commonly uses semidefinite programming (SDP) relaxations to bound subproblems. We propose a Max-Cut-specific *graph neural network* as a principled, lightweight neural proxy for these SDP solvers that can be plugged directly into an exact branch-and-bound framework. The proposed architecture has update steps of complexity $\mathcal{O}(n^2 + ne)$, and predicts both primal- and dual-feasible SDP solutions. The primal SDP solutions yield feasible Max-Cut solutions via the Goemans–Williamson algorithm. In addition, it is trained self-supervisedly without requiring solved SDP relaxations as labels. Empirically, we show that our architecture can substantially reduce the cost of bounding in exact Max-Cut solving by up to $10.6\times$ compared with using the state-of-the-art SDP solver Mosek. Our work highlights the potential of learned, validity-preserving surrogates for accelerating exact optimization over structured convex relaxations.

1 Introduction

Combinatorial optimization (CO) sits at the intersection of optimization, operations research, discrete mathematics, and computer science, with many high-impact real-world applications, including routing, scheduling, and network design [45]. In practice, however, optimization instances are rarely arbitrary; they typically arise from structured, application-specific distributions. Classical exact solvers are largely distribution-agnostic and therefore do not explicitly exploit recurring structure in the instances they encounter [10].

This observation has motivated a growing line of work on combining exact optimization with data-driven prediction. The central idea is to leverage machine learning to infer useful algorithmic information from previously seen instances while preserving the underlying solver's correctness guarantees. In this context, *graph neural networks* (GNNs), and *message-passing neural networks* (MPNNs) [74, 32] in particular are a natural choice, i.e., they are permutation-equivariant, naturally represent variable-constraint interactions, and can process instances of varying size. Here, prior work has shown encouraging results for linear programming [19, 70] and mixed-integer linear programming, e.g., in branching [21, 31, 61], primal solution prediction [43], and cut selection [23, 53, 65]; see Cappart et al. [15] for thorough survey.

[*]Equal contribution

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Many practically important optimization problems, however, go beyond the linear setting. In particular, *semidefinite programming* (SDP) relaxations play a central role in hard discrete optimization problems, yet repeatedly solving SDPs inside an exact branch-and-bound algorithm remains computationally demanding [79, 84, 81, 37, 72]. Here, we study the *Max-Cut problem* (Max-Cut), a canonical NP-complete problem with numerous applications, e.g., in statistical physics and circuit layout design [9, 66]. Exact Max-Cut can be solved via branch and bound, where each search-tree node is bounded by solving the SDP relaxation of the corresponding subproblem. Although these SDP bounds are often strong, recomputing them throughout the search is typically the primary computational bottleneck, even with state-of-the-art SDP solvers such as Mosek [6].

Contributions To address this, we replace expensive SDP solvers with a learned neural proxy that remains compatible with exact optimization. Building on recent principled progress on solving SDPs with expressive GNNs [69], we propose a sparsity-aware Max-Cut-specific GNN architecture that predicts feasible primal and dual solutions to the SDP relaxation and integrates directly into an exact branch-and-bound framework. Unlike most prior learning-to-optimize approaches, which learn policies to *guide* branch and bound while leaving the bounding step to a conventional solver [31, 43, 65], our method targets the bounding computation itself. That is, by learning valid bounds rather than relying solely on heuristic decisions, we preserve the exact algorithm's performance guarantees while substantially reducing overall computation time; see Figure 1 for an overview of our proposed framework.

Concretely, our main contributions are as follows.

1. We propose a Max-Cut-specific GNN architecture with update step complexity $\mathcal{O}(n^2 + ne)$, where n denotes the number of vertices and e the number of edges, reducing the $\mathcal{O}(n^3)$ bottleneck of prior work [69].
2. Our architecture predicts both primal-feasible and dual-feasible solutions for the Max-Cut SDP relaxation. We enforce feasibility by construction, so the predicted dual solution yields a valid upper bound and can be directly integrated into branch-and-bound.
3. We train the architecture in a self-supervised fashion directly on Max-Cut instances, without requiring solved SDP relaxations for supervision.
4. We integrate the learned predictor into an exact branch-and-bound framework for Max-Cut and show speed-ups up to $10.6\times$ over the same framework using the state-of-the-art interior-point method solver Mosek [6]. To the best of our knowledge, this is the first branch-and-bound implementation that uses a neural network solely to evaluate the problem relaxation.

1.1 Related work

GNNs MPNNs [32, 74] have been extensively studied. Spatial MPNNs [11, 26, 36, 80, 86] follow the message-passing paradigm introduced by Gilmer et al. [32]. However, standard MPNNs are fundamentally limited by the expressivity of the 1-*dimensional Weisfeiler–Leman (*1-WL*) test* [86, 58], motivating various more expressive architectures, most notably *higher-order GNNs* [58, 59, 57, 60].

GNN for (combinatorial) optimization In combinatorial optimization, GNNs have demonstrated significant potential. For instance, in *linear programs* (LPs) and *mixed-integer linear programs* (MILPs), GNNs play a critical role by encoding the problem structure into a constraint-variable bipartite graph [31, 24, 19, 43, 70]. Theoretically, a growing body of literature has analyzed the expressivity required for GNNs to approximate the solutions of LPs [19, 20, 70, 48, 50, 51], quadratic programs (QPs) [22, 85, 67], second-order cone programs (SOCPs) [52], and, most recently, linear semidefinite programs [69]. Closely related is Yau et al. [87], which aligns GNNs with low-rank SDP relaxations to generate rounded heuristic solutions for Max-CSP. In contrast, we explicitly approximate the continuous SDP relaxation to accelerate exact Max-Cut branch-and-bound algorithms.

Neural branching Branch and bound [2, 1] is the foundational algorithm for exactly solving MILPs. To accelerate it, early machine learning methods learned surrogate scores [4] or ranking functions [42] for branching. More recently, Gasse et al. [31] trained an MPNN to directly imitate solver-branching behavior, replacing the costly strong-branching score computation. Subsequent works by Zarpellon

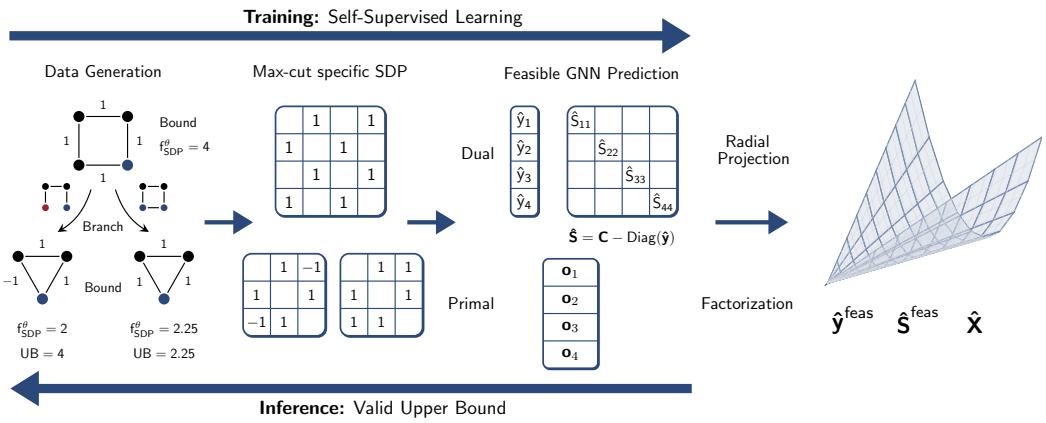

Figure 1: **Overview of our neural SDP surrogate for Max-Cut. (1) Data Generation:** Subgraphs are generated dynamically from *random* branch-and-bound search trees. **(2) GNN Prediction:** MC-MPNN predicts unconstrained primal and dual features. **(3) Feasibility Guarantee:** A radial projection shifts the $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{S}}$ to the feasible region, while the primal matrix is factorized by construction. *The resulting valid bounds are used to aggressively prune the search tree during exact inference.*

et al. [88], Lin et al. [56] have extended this by incorporating richer topological information from the search tree. Theoretically, Chen et al. [21] and Zhou et al. [89] investigated the expressivity required for effective variable selection and concluded that 1-WL-equivalent architectures are inherently inadequate. For a comprehensive survey, please refer to [75, 18]. Despite these advances, existing methods predict which variable to branch on while still relying on classical LP solvers, and no fully neural branch-and-bound algorithm has been developed.

Learning for constrained optimization Another closely related line of work focuses on learning near-optimal solutions directly. Various methods have been proposed to ensure feasible or close-to-feasible predictions for constrained optimization problems. A naive approach is to incorporate penalty functions for constraint violations [28, 63, 70, 76]. More advanced methods include gradient descent correction [25], gauge function [49, 78], homomorphic projection [54, 55], decision rule [30], differentiable projection layer [5, 3, 17], and search in feasible region [68, 71]. In parallel with these structural enforcements, several works have investigated primal-dual Lagrangian methods to ensure feasibility [29, 64, 77, 46]. For Max-Cut SDP relaxations, Yau et al. [87] enforce primal feasibility via a low-rank factorization combined with normalization. Drawing inspiration from Yau et al. [87]'s architectural guarantees and Tanneau and Van Hentenryck [77] 's dual perspectives, our work develops a feasible, self-supervised learning framework tailored for Max-Cut SDPs.

Global solvers for Max-Cut Over the years, researchers have developed many SDP-based branch-and-bound methods to solve Max-Cut problems to global optimality. These methods strengthen the basic SDP relaxation in [33] by incorporating cutting planes (cuts), known as odd-cycle inequalities [38]. Solvers such as BiqMac [73], BiqBin [35], BiqCrunch [47], and MADAM [40] follow this approach. However, the number of triangle inequalities, the lowest-order odd-cycle inequalities, scales as $\mathcal{O}(n^3)$, making it impractical to include them all explicitly. Consequently, these methods rely on first-order algorithms, such as bundle methods, combined with heuristics that selectively add a small subset of cuts. In contrast, our proposed method does not incorporate such cuts and is therefore *not directly comparable* to these solvers in their most advanced configurations. To provide a fair assessment, we benchmark against these solvers both with and without cutting planes enabled in Section D.6.

1.2 Background

The Max-Cut problem Let $G := (V, E)$ be an undirected graph with nodes V ($|V| := n$) and edge weights $w_{ij} \in \mathbb{R}$ for $\{i, j\} \in E$. The Max-Cut problem seeks to partition the nodes into two disjoint sets to maximize the sum of the weights of the edges crossing the cut. By assigning a binary variable $x_i \in \{-1, 1\}$ to each node to indicate its partition, the problem can be formulated as the quadratic

integer program

$$\max_{\boldsymbol{x}\in\{-1,1\}^n} \quad \frac{1}{2}\sum_{\{i,j\}\in E} w_{ij}(1-x_ix_j) = \frac{1}{4}\boldsymbol{x}^\intercal \boldsymbol{L}\boldsymbol{x}, \tag{1}$$

where $\boldsymbol{L}$ is the graph Laplacian. Alternatively, using the adjacency matrix as the coefficient matrix $\boldsymbol{C}$, this is equivalent to minimizing $\boldsymbol{x}^\intercal \boldsymbol{C}\boldsymbol{x}$.

Semidefinite relaxation Because the integer formulation is NP-complete, a standard practice is to utilize its (SDP) relaxation [33]. By relaxing the scalar $x_i \in \{-1, 1\}$ to a unit vector $\boldsymbol{v}_i \in \mathbb{R}^n$ such that $\|\boldsymbol{v}_i\| = 1$, we define a positive semidefinite (PSD) matrix $\boldsymbol{X} \in \mathbb{S}^n_+$ where $X_{ij} = \boldsymbol{v}_i^\intercal \boldsymbol{v}_j$. The primal and dual formulations of the Max-Cut SDP relaxation are

$$\max_{\boldsymbol{X}\in\mathbb{S}^n_+} \quad \langle \boldsymbol{L},\, \boldsymbol{X}\rangle \tag{2a}$$

$$\text{s.t.} \quad \operatorname{diag}(\boldsymbol{X}) = \mathbf{e}, \tag{2b}$$

$$\min_{\boldsymbol{y}\in\mathbb{R}^n, \boldsymbol{S}\in\mathbb{S}^n_+} \quad \langle \boldsymbol{e},\, \boldsymbol{y}\rangle \tag{2c}$$

$$\text{s.t.} \quad \boldsymbol{L} - \operatorname{Diag}(\boldsymbol{y}) + \boldsymbol{S} = 0. \tag{2d}$$

To analyze this theoretically, we can view Equation (2b) as a specific instance of the standard linear SDP form, $\min_{\boldsymbol{X}\in\mathbb{S}^n_+} \langle \boldsymbol{C},\, \boldsymbol{X}\rangle$ s.t. $\mathcal{A}(\boldsymbol{X}) = \boldsymbol{b}$. Here, $\boldsymbol{C}$ is the objective matrix, and the linear operator $\mathcal{A}\colon \mathbb{S}^n \to \mathbb{R}^m$ maps the matrix to m constraints via $\mathcal{A}(\boldsymbol{X}) = \big(\langle \boldsymbol{A}_1,\, \boldsymbol{X}\rangle, \ldots, \langle \boldsymbol{A}_m,\, \boldsymbol{X}\rangle\big)^\intercal$. For Max-Cut, the constraints are strictly uniform and diagonal (i.e., $X_{ii} = 1$), meaning $m = n$ and the constraint matrices $\boldsymbol{A}_k$ are simply single-entry indicator matrices.

Expressive GNN for Max-Cut SDP The 1-WL test [82, 13] iteratively refines a node's color based on the multiset of its neighbors' colors via an injective function, i.e., $\boldsymbol{c}_v^t \coloneqq \mathsf{hash}(\boldsymbol{c}_v^{t-1}, \{\!\{\boldsymbol{c}_u^{t-1} \mid u \in N(v)\}\!\})$, where $\{\!\{\ldots\}\!\}$ denotes a multiset. MPNNs are recognized as the neural instantiation of the 1-WL test [86, 58]. In each layer $t > 0$, an MPNN updates a node's feature $\boldsymbol{h}_v^t$ by aggregating messages from its local neighborhood $N(v)$ as

$$\boldsymbol{m}_v^t = \mathsf{AGG}^t\Big(\Big\{\!\!\Big\{\big(\boldsymbol{h}_v^{t-1}, \boldsymbol{h}_u^{t-1}, w_{vu}\big) \mid u \in N(v)\Big\}\!\!\Big\}\Big), \quad \boldsymbol{h}_v^t = \mathsf{UPD}^t\big(\boldsymbol{h}_v^{t-1}, \boldsymbol{m}_v^t\big),$$

where AGG^t and UPD^t are parameterized functions, e.g., neural networks. For higher expressivity, the 1-WL extends to k-WL and k-FWL; see Section A.1 for more detailed background of WL hierarchy.

As shown in Qian and Morris [69], representing SDPs necessitates higher-order expressivity equivalent to the 2-FWL test; thereby, the VC-2-FWL framework has been established. It initializes variable embeddings $\boldsymbol{v}_{ij}^0$ from the objective $\boldsymbol{C}$ and constraint embeddings $\boldsymbol{c}_k^0$ from the bounds $\boldsymbol{b}$ as

$$\boldsymbol{v}_{ij}^0 \coloneqq \mathsf{init}_{\mathsf{v}}(C_{ij}, \mathbb{I}_{i=j}), \quad \boldsymbol{c}_k^0 \coloneqq \mathsf{init}_{\mathsf{c}}(b_k) \tag{3}$$

and applies the following joint update:

$$\boldsymbol{v}_{ij}^t \coloneqq \mathsf{hash}\Big(\boldsymbol{v}_{ij}^{t-1}, \big\{\!\!\big\{\{\!\{\boldsymbol{v}_{uj}^{t-1}, \boldsymbol{v}_{iu}^{t-1}\}\!\} \mid u \in [n]\big\}\!\!\big\}, \big\{\!\!\big\{(A_{k,ij}, \boldsymbol{c}_k^{t-1}) \mid k \in N(ij)\big\}\!\!\big\}\Big)$$

$$\boldsymbol{c}_k^t \coloneqq \mathsf{hash}\Big(\boldsymbol{c}_k^{t-1}, \big\{\!\!\big\{\big(A_{k,ij}, \boldsymbol{v}_{ij}^{t-1}\big) \mid (i,j) \in N(k)\big\}\!\!\big\}\Big).$$

The update step has a computational complexity of $\mathcal{O}(n^3 + \operatorname{nnz}(\mathcal{A}))$ per update step for general linear SDP, motivating a sparsity-aware variant; see Section 2.1.

Branch and Bound To solve the Max-Cut in (1) to global optimality, the *branch-and-bound* (B&B) method is typically used [73]. The branch step recursively decomposes a graph into two smaller ones by assigning a vertex to one partition or the other. As shown in Figure 1, the vertices can be contracted to produce smaller equivalent graphs with updated edge weights w_{ij}. Most standard Max-Cut solvers, such as BiqMac [73], follow this contraction-based approach to recursively reduce the graph size by one. The bound step solves the SDP relaxation of the resulting subproblems to provide an upper bound f^*_{SDP}, while the best feasible cut $\boldsymbol{x} \in \{-1, 1\}^n$ gives a lower bound f_{LB}. The Goemans–Williamson algorithm [33], based on the primal SDP solution, typically yields good feasible Max-Cut solutions. Hence, the B&B method progressively closes the gap, and a B&B node can be safely pruned whenever $f^*_{\mathrm{SDP}} < f_{\mathrm{LB}} + 1$.

2 A principled GNN architecture to learn feasible bounds

In this work, we demonstrate that by exploiting the uniform diagonal structure of the Max-Cut relaxation, we can safely bypass this general formulation. We propose a highly optimized, sparsity-aware architecture that preserves the necessary higher expressivity while dramatically reducing the computational bottleneck for branch-and-bound integration.

2.1 Max-Cut-specific GNN

Here, we propose a Max-Cut-specific algorithm and its corresponding neural architecture. To that end, we first observe that, for the Max-Cut problem, the constraints depend uniformly on the diagonals. This matches the diagonal indicator at variable-node initialization in Equation (3), meaning the constraints are uniquely and sufficiently marked. Consequently, we can safely drop the corresponding constraint nodes. Removing redundant constraint nodes yields our first specific architecture, Max-Cut Weisfeiler–Leman (MC-WL). The algorithm initializes variables following Equation (3), with the color update step defined following, where $\oplus$ denotes any choice of injective and order-invariant aggregation function, e.g., multiset hashing or element-wise summation in neural networks:

$$\mathsf{MC\text{-}WL}\colon \quad \boldsymbol{v}_{ij}^{t} \coloneqq \mathsf{hash}\Big(\boldsymbol{v}_{ij}^{t-1}, \big\{\!\!\big\{ \boldsymbol{v}_{uj}^{t-1} \oplus \boldsymbol{v}_{iu}^{t-1} \mid u \in [n] \big\}\!\!\big\}\Big).$$

In addition, the objective matrix $\boldsymbol{C}$ is often sparse. In fact, based on the proof of the expressivity of the VC-2-FWL, the variable update can be reformulated to be sparsity-aware by incorporating $\boldsymbol{C}$ directly into the message passing, resulting in a variant:

$$\begin{aligned}\delta\text{-}\mathsf{MC\text{-}WL}\colon \quad \boldsymbol{v}_{ij}^{t} \coloneqq \mathsf{hash}\Big(\boldsymbol{v}_{ij}^{t-1}, &\big\{\!\!\big\{ (\boldsymbol{v}_{iu}^{t-1}, C_{uj}) \mid u \in [n], C_{uj} \neq 0 \big\}\!\!\big\} \\ &\oplus \big\{\!\!\big\{ (\boldsymbol{v}_{uj}^{t-1}, C_{iu}) \mid u \in [n], C_{iu} \neq 0 \big\}\!\!\big\}\Big). \end{aligned} \tag{4}$$

The connection of the multisets via the order-invariant $\oplus$ is to guarantee symmetry $\boldsymbol{v}_{ij}^{t} = \boldsymbol{v}_{ji}^{t}$.

To determine the running-time complexity of δ-MC-WL, we analyze its aggregation steps. The initialization and update is on every index $(i, j) \in [n]^2$, taking $\mathcal{O}(n^2)$ time. Furthermore, the sparse attribute aggregations filter exclusively for the non-zero entries of $\boldsymbol{C}$. For a given column j, let d_j be its number of non-zero entries. Computing the multiset $\big\{\!\!\big\{ (\boldsymbol{v}_{iu}^{t-1}, C_{uj}) \mid u \in [n], C_{uj} \neq 0 \big\}\!\!\big\}$ for all n rows requires $\mathcal{O}(n \cdot d_j)$ operations. Summing this cost across all columns yields $\mathcal{O}(n \sum d_j) = \mathcal{O}(ne)$, where e is the total number of non-zero entries. This concludes the overall running-time complexity per update step, which is $\mathcal{O}(n^2 + ne)$.

The following results show that δ-MC-WL is expressive enough to represent the optimal solution of a Max-Cut instance.

Theorem 2.1. *Let $\boldsymbol{X}^* \in \mathbb{S}_+^n$ and $\boldsymbol{y}^* \in \mathbb{R}^n$ denote the optimal primal and dual solutions for a given Max-Cut SDP relaxation. For any indices (i, j) and (p, q), if the stable colors produced by δ-MC-WL satisfy $\boldsymbol{v}_{ij}^{\infty} = \boldsymbol{v}_{pq}^{\infty}$, then the corresponding primal entries satisfy $X_{ij}^* = X_{pq}^*$. Furthermore, if the diagonal stable colors satisfy $\boldsymbol{v}_{kk}^{\infty} = \boldsymbol{v}_{ll}^{\infty}$, then the dual solution satisfies $y_k^* = y_l^*$.*

Building on Qian and Morris [69], we can neuralize the algorithms defined above into specific neural network architectures. The variable features are all initialized as $\boldsymbol{h}_{ij}^{0} \coloneqq \mathsf{INIT}(C_{ij}, \mathbb{I}_{i=j}) \in \mathbb{R}^d$, where INIT is a parameterized function.

First, translating MC-WL into the MC-MPNN architecture, we encode the pairwise $\oplus$ operation with a MAP function and the outer multiset aggregation with a neural network MSG, using sum aggregation for both operations:

$$\mathsf{MC\text{-}MPNN} \quad \boldsymbol{h}_{ij}^{t} \coloneqq \mathsf{UPD}^{t}\left(\boldsymbol{h}_{ij}^{t-1}, \sum_{u \in [n]} \mathsf{MSG}^{t}\Big(\mathsf{MAP}^{t}\left(\boldsymbol{h}_{uj}^{t-1}\right) + \mathsf{MAP}^{t}\left(\boldsymbol{h}_{iu}^{t-1}\right)\Big)\right).$$

To make the architecture objective-aware, we neuralize δ-MC-WL by embedding the matrix entries C directly into the message passing. Merging both multisets after the MSG function results in the

δ-MC-MPNN update:

$$\delta\text{-MC-MPNN} \quad \boldsymbol{h}_{ij}^{t} \coloneqq \mathsf{UPD}^{t}\left(\boldsymbol{h}_{ij}^{t-1}, \sum_{u\colon C_{iu}\neq 0} \mathsf{MSG}^{t}\big(\boldsymbol{h}_{uj}^{t-1}, C_{iu}\big) + \sum_{u\colon C_{uj}\neq 0} \mathsf{MSG}^{t}\big(\boldsymbol{h}_{iu}^{t-1}, C_{uj}\big)\right).$$

In Section B.3 we prove that the above architectures can be expressive enough to approximate the optimal solution of a Max-Cut SDP.

2.2 Enforce feasibility

Let $\boldsymbol{h}_{ij}^{T}$ denote the final feature tensor produced after T GNN layers. Because the variables operate in matrix space, we first extract node-level representations $\boldsymbol{h}_i$ by aggregating the symmetric pairwise embeddings across rows (or columns). From these node features, we use two independent multi-layer perceptron (MLP) heads to produce primal and dual predictions.

For the primal variable, an MLP projects $\boldsymbol{h}_i$ into a lower-dimensional embedding, which is subsequently ℓ_2-normalized to yield a unit vector $\boldsymbol{o}_i \in \mathbb{R}^r$ such that $\|\boldsymbol{o}_i\|_2 = 1$. We then construct the primal matrix via the inner product $\hat{X}_{ij} \coloneqq \boldsymbol{o}_i^{\intercal}\boldsymbol{o}_j$, which structurally guarantees positive semidefiniteness. This prediction is trained by directly optimizing Equation (2a).

For the dual problem, a separate MLP maps $\boldsymbol{h}_i$ to an unconstrained scalar prediction $\hat{y}_i \in \mathbb{R}$. However, for the Max-Cut SDP relaxation (2), dual feasibility requires the slack matrix to be positive semidefinite, i.e., $\boldsymbol{S} \in \mathbb{S}_{+}^{n}$. The raw network outputs $\hat{\boldsymbol{y}}$ typically violate this condition. To enforce dual feasibility, we evaluate the predicted slack matrix using the equality constraint (2d) as $\hat{\boldsymbol{S}} \coloneqq \mathrm{Diag}(\hat{\boldsymbol{y}}) - \boldsymbol{L}$, and apply a radial projection. By uniformly shifting the dual variables by the magnitude of the most negative eigenvalue of $\hat{\boldsymbol{S}}$ [2], we guarantee a valid, feasible dual solution,

$$\hat{\boldsymbol{y}}^{\mathrm{feas}} \coloneqq \hat{\boldsymbol{y}} + \max\{0,\, -\lambda_{\min}(\hat{\boldsymbol{S}})\}\mathbf{e},$$

based on which we can directly optimize with Equation (2c) during training.

Crucially, directly optimizing these objectives aligns seamlessly with our theoretical analysis. To guarantee uniqueness for symmetry analysis, Theorem 2.1 isolates the minimum Frobenius norm solution, ensuring that nodes indistinguishable by δ-MC-WL share identical values. In contrast, our self-supervised learning (SSL) framework optimizes the objective directly without this regularization, allowing for a more flexible optimization search space. Nevertheless, because the predicted solution is parameterized by the structurally refined colorings of δ-MC-WL, it inherently preserves the necessary matrix symmetries within this broader space.

2.3 Branch and bound baseline and GNN-enhanced variants

We consider three B&B solvers for Max-Cut: (i) a vanilla, SDP solver-based B&B solver; (ii) a hybrid B&B solver; and (iii) a neural B&B solver. For completeness, the Section A.2 provides additional background on B&B with the pseudo code of the three solvers shown in Algorithm 1.

In the vanilla solver (i), each node is evaluated by solving the SDP relaxation with Mosek to obtain a valid upper bound. The resulting primal SDP solution is then rounded by the Goemans–Williamson algorithm to obtain a feasible cut, which can be used to potentially update the incumbent lower bound. Nodes are selected using a best-bound-first strategy, and branching is performed on the most fractional variable. This solver serves as our standard SDP-based baseline.

The first GNN-enhanced variant is the hybrid solver (ii) that uses the GNN as an early-fathoming oracle before invoking Mosek. Because the proposed GNN predicts a dual-feasible solution by construction, its output always provides a valid upper bound. If this bound already satisfies the pruning condition, the node is fathomed immediately, and the Mosek evaluation is skipped. Otherwise, Mosek is invoked, and the algorithm proceeds exactly as in the vanilla solver.

The second GNN-enhanced variant is the fully neural solver (iii), in which both primal and dual solutions are produced by the GNN. The predicted dual-feasible solution provides a valid upper bound, while the predicted primal-feasible solution is passed to the Goemans–Williamson algorithm

[2] In practice, the spectral decomposition needs to be done only once per instance and is not the computational bottleneck.

to obtain feasible cuts. Thus, the SDP relaxation is never solved by Mosek. Moreover, because GNN inference is lightweight, many child nodes can be collected, batched as graphs, and evaluated in parallel, enabling substantially faster node enumeration while preserving the validity of the bounds.

3 Experimental study

The ultimate goal of our architecture is to accelerate exact combinatorial optimization. In this section, we rigorously assess the practical utility of our proposed framework by answering the following key questions:

Q1 Do the Max-Cut-specific GNNs, especially sparsity-aware formulations, yield measurable, practical improvements in computational efficiency?

Q2 How much can our neural SDP predictor accelerate exact Max-Cut solving by replacing conventional evaluations of the basic SDP relaxation within a controlled B&B framework?

3.1 Experiment design

Comparison of GNN variants on uniform dataset To address **Q1**, we directly compare the predictive performance of MC-MPNN and δ-MC-MPNN on synthetic graphs. We generate three distinct datasets of Erdős–Rényi (ER) graphs [27] – the most common graphs for Max-Cut benchmarking – varying the number of nodes $N \in \{50, 100\}$ and the edge probabilities $p \in \{0.1, 0.5\}$. Each dataset contains $10\,000$ instances and is partitioned into training, validation, and test sets at an $8{:}1{:}1$ ratio. Because our framework utilizes SSL loss, the training phase is entirely unsupervised and requires no ground-truth labels. While our training dynamics largely follow Qian and Morris [69], we adapt the framework for SSL by directly optimizing the dual objective.

Finally, to evaluate the predictive accuracy of our models, we compute the exact labels for the test set using the Mosek solver [6]. We compare the models using the averaged relative objective gap over the test set: $|(\text{obj} - \text{obj}^*)\,/\text{obj}^*| \cdot 100\%$, where obj is the predicted value and obj^* is the ground-truth.

B&B experiments on Max-Cut dataset For **Q2**, we evaluate the three variants of B&B described in Section 2.3: (i) a vanilla SDP-based B&B solver using Mosek as baseline; (ii) a hybrid B&B solver that uses GNN for early pruning and invokes Mosek if the node is not pruned; (iii) a neural B&B solver in which the GNN produces lower bounds and upper bounds. To justify our choice of Mosek as the underlying SDP solver for these evaluations, we carried out ablation experiments comparing multiple SDP solvers in Section D.1. For the neural solver, we test both sequential evaluation (no batch) and batched evaluation with batch sizes $K \in \{8, 16, 32\}$. In batched evaluation, the solver instead selects up to K active nodes from the priority queue before performing inference. After branching on these nodes, the resulting $2K$ child-node subgraphs are collected into a mini-batch and evaluated together by the GNN. The predicted bounds are then assigned back to the corresponding child nodes, after which the usual pruning and queue-update steps are applied. Ablation studies of different batch sizes are shown in Sections D.5 and D.6. Experiments were conducted on AMD EPYC 7662 CPU and NVIDIA L40S GPU.

In every epoch, we generate 20 ER seed graphs on the fly from the same distribution of a target instance family. To mimic the subproblems encountered during the B&B tree, for each seed graph, we sample 20 *random branching trajectories* until 3 free variables are reached and expand them into $20 \times |N-2|$ subgraphs subject to successive node contractions. The B&B solvers are tested on unseen benchmark instances from Max-Cut libraries reported in [73, 35], including (1) `g05_N` for $N \in \{60, 80, 100\}$ with edge probability 0.5 and edge weights in $\{0, 1\}$, (2) `pm1s_N` for $N \in \{80, 100\}$ with edge probability 0.1 and edge weights in $\{-1, 0, 1\}$; (3) `w01_N` for $N \in \{100\}$ with edge probability 0.1 with integer edge weights in $[-10, 10]$. The model trained within `g05_100` distribution is tested on `g05_60`, `g05_80`, and `g05_100`, while the remaining models are trained and tested on their respective distributions.

3.2 Results

Q1: Comparison of GNN variants Table 1 demonstrates the objective gaps and the computational overheads during training. Notably, δ-MC-MPNN and MC-MPNN exhibit comparable predictive accuracy and training time across all datasets, though δ-MC-MPNN requires slightly less VRAM.

Therefore, we frame our contributions along two parallel tracks: δ-MC-WL establishes the theory that sub-cubic complexity is possible for Max-Cut SDPs, laying the foundation for future sparse-optimized hardware. Meanwhile, MC-MPNN and δ-MC-MPNN both serve as competitive realizations of our theory, emerging as the most practical and efficient choices for deployment.

Table 1: Training computational overhead, measured in seconds per epoch and VRAM usage allocated, alongside predictive performance on the test set, averaged over 3 random seeds. The best results are highlighted in bold.

Target	Model	Problems		
		N=50, p=0.1	N=50, p=0.5	N=100, p=0.1
Obj gap (%)	MC-MPNN	**0.126**$\pm$**0.005**	0.101$\pm$0.001	**0.598**$\pm$**0.005**
	δ-MC-MPNN	0.128$\pm$0.005	**0.099**$\pm$**0.001**	0.628$\pm$0.004
Time sec/epoch	MC-MPNN	6.5	**6.3**	**27.9**
	δ-MC-MPNN	**6.0**	**6.3**	28.2
VRAM (GB)	MC-MPNN	23.6	23.6	82.0
	δ-MC-MPNN	**21.6**	**21.6**	**81.6**

Additional ablation studies, including sensitivity analyses of network depth, out-of-distribution generalization, and normalization strategies (LayerNorm, GraphNorm etc.), are provided in Section D.

Q2: Branch-and-bound experiments We first conduct a controlled experiment to isolate the effect of the GNN-predicted upper bounds. In this experiment, we disable the Goemans–Williamson rounding step in all three B&B solvers (i)–(iii) and initialize the global lower bound with the true optimal Max-Cut value at the root node. This setup removes the influence of primal heuristics and incumbent updates, so differences in performance can be attributed directly to the quality of the upper bounds used for fathoming. The complete heuristic-free B&B statistics for all three solvers are provided in Section D.5. In Table 2, we highlight that the hybrid solver (ii) predicts sufficiently tight upper bounds to identify more than 83% of SDP-prunable nodes during B&B.

Table 2: Average fraction of SDP-prunable nodes pruned by GNN in the B&B without Goemans–Williamson algorithm using the hybrid solver on established instances. The best-performing configuration, MC-MPNN + LayerNorm, is used for the hybrid solver.

Instance Family	Method	% Nodes Pruned by GNN
g05_60	Hybrid	85.5%
g05_80	Hybrid	86.8%
g05_100	Hybrid	83.8%

Having isolated the effect of upper-bound quality, we next evaluate whether the learned bounds remain useful in a full B&B pipeline where incumbent solutions are generated dynamically. For this experiment, we focus on variants (i) and (iii): the vanilla solver and the fully neural solver. In both cases, feasible Max-Cut solutions are obtained via the Goemans–Williamson algorithm, using primal SDP solutions from Mosek for the vanilla solver and from the GNN for the neural solver. The results are reported in Table 3, where the neural solver uses the best-performing configuration selected in the ablation studies in Section D.6: MC-MPNN with LayerNorm and batched inference of size 32.

In this work, our neural proxy is designed to accelerate the evaluation of the basic SDP relaxation. The vanilla solver that solves basic SDP relaxation using Mosek provides an efficient implementation within this scope, making it the primary controlled baseline for evaluating our method. To compare the impact of relaxation strength on end-to-end performance, we consider complementary baselines: BiqCrunch represents a state-of-the-art exact Max-Cut solver employing stronger SDP relaxations

Table 3: Branch-and-bound results on established instances. The #node and time are averages over each instance family. The best-performing configuration, MC-MPNN + LayerNorm with batched inference of size 32, is reported for the neural solver.

Instance	Solver	Completed	Gap (%)	# Nodes Evaluated	Solving Time (s)	Speed-up ($\times$)
g05_60	Vanilla	10/10	0.00	3566	53.1	1.0
	Neural	10/10	0.00	4562	**9.5**	**5.6**
	BiqCrunch (w/o cuts)	10/10	0.00	3918	153.9	0.34
	BiqCrunch (w/ cuts)	10/10	0.00	7	2.3	26.50
	Gurobi	10/10	0.00	10994	66.7	0.80
	BARON	3/10	3.70	220182	$>$ 1706.0	$<$ 0.03
	SCIP	1/10	8.58	19825	$>$ 1744.1	$<$ 0.03
g05_80	Vanilla	10/10	0.00	49128	1170.3	1.0
	Neural	10/10	0.00	53975	**155.7**	**7.5**
	BiqCrunch (w/o cuts)	10/10	0.00	48633	3746.3	0.31
	BiqCrunch (w/ cuts)	10/10	0.00	49	22.0	53.20
	Gurobi	10/10	0.00	15691	1139.8	1.03
	BARON	0/10	5.47	191799	$>$ 3600.0	$<$ 0.33
	SCIP	0/10	24.15	14144	$>$ 3600.0	$<$ 0.33
g05_100	Vanilla	10/10	0.00	1087290	40930.1	1.0
	Neural	10/10	0.00	1272801	**4509.3**	**9.1**
	BiqCrunch (w/ cuts)	10/10	0.00	390	270.7	151.20
	Gurobi	0/10	6.96	406902327	$>$ 43200.0	$<$ 0.95
pm1s_80	Vanilla	10/10	0.00	25656	609.5	1.0
	Neural	10/10	0.00	27809	**80.8**	**7.5**
pm1s_100	Vanilla	10/10	0.00	254408	10382.5	1.0
	Neural	10/10	0.00	302314	**1058.2**	**9.8**
w01_100	Vanilla	10/10	0.00	269407	11348.6	1.0
	Neural	10/10	0.00	305720	**1073.9**	**10.6**

with additional triangle inequalities, while Gurobi, BARON, and SCIP provide alternatives employing weaker LP-based relaxations.

Overall, the neural solver evaluates slightly more nodes than the vanilla SDP-based solver, reflecting the fact that its learned bounds are somewhat less tight than the exact SDP solver. However, this modest increase in tree size is more than offset by the much lower cost of each node evaluation. This per-node cost reduction is driven by two complementary effects. First, the GNN replaces repeated SDP solves with lightweight inference. Second, batching enables multiple GNNs to be evaluated together in a single mini-batch, which improves hardware utilization. The speed-ups are consistent across all tested instance families and become more pronounced on larger instances, where SDP solves are increasingly expensive. The additional ablation studies of the two complementary effects are provided in Sections D.5 and D.6.

The strong performance of the model trained on g05_100-in-distribution graphs when tested on g05_60 and g05_80 suggests that the learned SDP proxy effectively transfers across graph sizes within the same instance family. This generalization is nontrivial because, although the training data contain contracted subgraphs of comparable sizes, they are typically denser and no longer have purely binary edge weights. A more detailed analysis of generalization is provided in Section D.4.

4 Conclusion

We introduced a feasibility-preserving GNN proxy for the semidefinite relaxations arising in exact Max-Cut branch and bound. By exploiting the sparse structure of the Max-Cut SDP, our architecture avoids the cubic bottleneck of more general GNNs for SDP while retaining the expressivity needed to represent optimal primal and dual solutions. The resulting model predicts primal-feasible and dual-feasible SDP solutions by construction, enabling valid upper bounds and self-supervised training without requiring solved SDP instances as labels. Empirically, our results show that learned SDP bounds can be sufficiently accurate to substantially reduce the cost of bounding in branch and bound. In particular, the proposed neural solver accelerates SDP-based exact Max-Cut solving by up to $10.6\times$ compared with the same framework using Mosek, while preserving correctness via dual feasibility. These findings suggest that learned solvers need not be limited to heuristic guidance, such

as branching or cut selection, but can instead replace core relaxation computations when feasibility and validity are enforced through a tailored GNN architecture.

Limitations and possible future directions The proposed framework exhibits two primary limitations. First, its reliance on generating and training on in-distribution data restricts the model's ability to generalize to out-of-distribution graphs. To address this, we plan to explore foundation models pretrained across diverse graph classes. Second, although our neural solver effectively accelerates exact combinatorial optimization, it does not yet consistently surpass state-of-the-art classical exact solvers such as BiqCrunch [47], which have benefited from decades of research on Max-Cut and derive much of their strength from advanced branch-and-cut paradigms. To bridge this performance gap, we aim to investigate tighter integration between neural SDP solvers and classical cutting-plane methods.

Acknowledgments and Disclosure of Funding

C.L. would like to acknowledge the financial support from the Office of Naval Research (ONR) award N000142412641. A.L. is grateful for the support of ONR, award N00014-24-1-2645. C.M. and C.Q. are partially funded by a DFG Emmy Noether grant (468502433) and the RWTH Junior Principal Investigator Fellowship under Germany's Excellence Strategy.

References

[1] T. Achterberg. *Constraint integer programming*. PhD thesis, 2007. 2

[2] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005. 2

[3] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 2019. 3

[4] A. M. Alvarez, Q. Louveaux, and L. Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017. 2

[5] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*. PMLR, 2017. 3

[6] M. ApS. Mosek optimizer api for python. *Version*, 9(17):6–4, 2022. 2, 7, 23

[7] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 25

[8] L. Babai. Lectures on graph isomorphism. University of Toronto, Department of Computer Science. Mimeographed lecture notes, October 1979, 1979. 16

[9] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3): 493–513, 1988. doi: 10.1287/opre.36.3.493. 2

[10] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. 1

[11] X. Bresson and T. Laurent. Residual gated graph ConvNets. *arXiv preprint arXiv:1711.07553v2*, 2017. 2

[12] S. Burer and R. D. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical programming*, 95(2):329–357, 2003. 23

[13] J.-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992. 4, 16

[14] T. Cai, S. Luo, K. Xu, D. He, T.-y. Liu, and L. Wang. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning*, 2021. 25

[15] Q. Cappart, D. Chételat, E. B. Khalil, A. Lodi, C. Morris, and P. Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023. 1

[16] A. Chambolle and T. Pock. On the ergodic convergence rates of a first-order primal–dual algorithm. *Mathematical Programming*, 159(1):253–287, 2016. 21

[17] B. Chen, P. L. Donti, K. Baker, J. Z. Kolter, and M. Bergés. Enforcing policy feasibility constraints through differentiable projection for energy optimization. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, pages 199–210, 2021. 3

[18] X. Chen, J. Liu, and W. Yin. Learning to optimize: A tutorial for continuous and mixed-integer optimization. *Science China Mathematics*, 67(6):1191–1262, 2024. 3

[19] Z. Chen, J. Liu, X. Wang, and W. Yin. On representing linear programs by graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023. 1, 2

[20] Z. Chen, J. Liu, X. Wang, and W. Yin. On representing mixed-integer linear programs by graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023. 2

[21] Z. Chen, J. Liu, X. Chen, X. Wang, and W. Yin. Rethinking the capacity of graph neural networks for branching strategy. *Advances in Neural Information Processing Systems*, 2024. 1, 3

[22] Z. Chen, X. Chen, J. Liu, X. Wang, and W. Yin. Expressive power of graph neural networks for (mixed-integer) quadratic programs. In *Forty-second International Conference on Machine Learning*, 2025. 2

[23] A. Deza and E. B. Khalil. Machine learning for cutting planes in integer programming: A survey. In *IJCAI*, 2023. 1

[24] J.-Y. Ding, C. Zhang, L. Shen, S. Li, B. Wang, Y. Xu, and L. Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *AAAI*, 2020. 2

[25] P. Donti, D. Rolnick, and J. Z. Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021. 3

[26] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. *arXiv preprint arXiv:1509.09292*, 2015. 2

[27] P. Erdős and A. Rényi. On random graphs i. *Publ. math. debrecen*, 6(290-297):18, 1959. 7

[28] F. Fioretto, T. W. Mak, and P. Van Hentenryck. Predicting ac optimal power flows: Combining deep learning and lagrangian dual methods. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 630–637, 2020. 3

[29] F. Fioretto, P. Van Hentenryck, T. W. Mak, C. Tran, F. Baldo, and M. Lombardi. Lagrangian duality for constrained deep learning. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 118–135. Springer, 2020. 3

[30] G. E. C. Flores, H. Chen, and C. Li. Enforcing hard linear constraints in deep learning models with decision rules. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. 3

[31] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. *Exact combinatorial optimization with graph convolutional neural networks*. Curran Associates Inc., Red Hook, NY, USA, 2019. 1, 2

[32] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017. 1, 2

[33] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42 (6):1115–1145, 1995. 3, 4

[34] M. Grohe. *Descriptive complexity, canonisation, and definable graph structure theory*. Cambridge University Press, 2017. 16

[35] N. Gusmeroli, T. Hrga, B. Lužar, J. Povh, M. Siebenhofer, and A. Wiegele. Biqbin: a parallel branch-and-bound solver for binary quadratic problems with linear constraints. *ACM Transactions on Mathematical Software (TOMS)*, 48(2):1–31, 2022. 3, 7, 23, 26

[36] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *NeurIPS*, 2017. 2

[37] Q. Han, C. Li, Z. Lin, C. Chen, Q. Deng, D. Ge, H. Liu, and Y. Ye. A low-rank admm splitting approach for semidefinite programming. *INFORMS Journal on Computing*, 2025. 2

[38] C. Helmberg and F. Rendl. Solving quadratic (0, 1)-problems by semidefinite programs and cutting planes. *Mathematical programming*, 82(3):291–315, 1998. 3

[39] R. A. Horn and C. R. Johnson. *Topics in matrix analysis*. Cambridge university press, 1994. 19, 20

[40] T. Hrga and J. Povh. Madam: a parallel exact solver for max-cut based on semidefinite programming and admm. *Computational Optimization and Applications*, 80:347–375, 2021. ISSN 1573-2894. doi: 10.1007/s10589-021-00310-6. 3

[41] N. Immerman and E. Lander. Describing graphs: A first-order approach to graph canonization. In *Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988*, pages 59–81, 1990. 16

[42] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016. 2

[43] E. B. Khalil, C. Morris, and A. Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. In *AAAI*, 2022. 1, 2

[44] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 23

[45] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 5th edition, 2012. 1

[46] J. Kotary and F. Fioretto. Learning constrained optimization with deep augmented lagrangian methods. *arXiv preprint arXiv:2403.03454*, 2024. 3

[47] N. Krislock, J. Malick, and F. Roupin. Biqcrunch: A semidefinite branch-and-bound method for solving binary quadratic problems. *ACM Trans. Math. Softw.*, 43(4), Jan. 2017. ISSN 0098-3500. doi: 10.1145/3005345. 3, 10, 28

[48] B. Li, L. Yang, Y. Chen, S. Wang, H. Mao, Q. Chen, Y. Ma, A. Wang, T. Ding, J. Tang, and R. Sun. Pdhg-unrolled learning-to-optimize method for large-scale linear programming. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024. 2

[49] M. Li, S. Kolouri, and J. Mohammadi. Learning to solve optimization problems with hard linear constraints. *IEEE Access*, 11:59995–60004, 2023. ISSN 21693536. doi: 10.1109/access.2023.3285199. 3

[50] Q. Li, T. Ding, L. Yang, M. Ouyang, Q. Shi, and R. Sun. On the power of small-size graph neural networks for linear programming. In *NeurIPS*, 2024. 2

[51] Q. Li, M. Ouyang, T. Ding, Y. Wang, Q. Shi, and R. Sun. Towards explaining the power of constant-depth graph neural networks for structured linear programming. In *ICLR*, 2025. 2

[52] R. Li, E. Liang, and M. Chen. On the expressivity of GNN for solving second order cone programming. In *NeurIPS Workshop on GPU-Accelerated and Scalable Optimization*, 2025. 2

[53] S. Li, W. Ouyang, M. B. Paulus, and C. Wu. Learning to configure separators in branch-and-cut. In *NeurIPS*, 2023. 1

[54] E. Liang, M. Chen, and S. Low. Low complexity homeomorphic projection to ensure neural-network solution feasibility for optimization over (Non-)Convex set. In *International Conference on Machine Learning*. PMLR, 2023. 3

[55] E. Liang, M. Chen, and S. H. Low. Homeomorphic projection to ensure neural-network solution feasibility for constrained optimization. *Journal of Machine Learning Research*, 25:1–55, 2024. 3

[56] J. Lin, J. Zhu, H. Wang, and T. Zhang. Learning to branch with tree-aware branching transformers. *Knowledge-Based Systems*, 252:109455, 2022. 3

[57] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. In *NeurIPS*, 2019. 2, 22

[58] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI*, 2019. 2, 4

[59] C. Morris, G. Rattan, and P. Mutzel. Weisfeiler and Leman go sparse: Towards higher-order graph embeddings. In *NeurIPS*, 2020. 2

[60] C. Morris, G. Rattan, S. Kiefer, and S. Ravanbakhsh. SpeqNets: Sparsity-aware permutation-equivariant graph networks. In *ICML*, 2022. 2

[61] V. Nair, S. Bartunov, F. Gimeno, I. von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, R. Addanki, T. Hapuarachchi, T. Keck, J. Keeling, P. Kohli, I. Ktena, Y. Li, O. Vinyals, and Y. Zwols. Solving mixed integer programs using neural networks. *CoRR*, abs/2012.13349, 2020. 1

[62] B. O'Donoghue. Operator splitting for a homogeneous embedding of the linear complementarity problem. *SIAM Journal on Optimization*, 31(3):1999–2023, 2021. 23

[63] X. Pan, M. Chen, T. Zhao, and S. H. Low. Deepopf: A feasibility-optimized deep neural network approach for ac optimal power flow problems. *IEEE Systems Journal*, 17(1):673–683, 2022. 3

[64] S. Park and P. Van Hentenryck. Self-supervised primal-dual learning for constrained optimization. In *AAAI*, 2023. 3

[65] M. B. Paulus, G. Zarpellon, A. Krause, L. Charlin, and C. Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 17584–17600. PMLR, 17–23 Jul 2022. 1, 2

[66] S. Poljak and Z. Tuza. The max-cut problem — a survey. In *Special Year on Combinatorial Optimization*. American Mathematical Society, 1995. 2

[67] C. Qian and C. Morris. Principled data augmentation for learning to solve quadratic programming problems. *arXiv preprint arXiv:2506.01728*, 2025. 2

[68] C. Qian and C. Morris. Towards graph neural networks for provably solving convex optimization problems. *arXiv preprint arXiv:2502.02446*, 2025. 3

[69] C. Qian and C. Morris. On the expressive power of gnns to solve linear sdps. *arXiv preprint arXiv:2604.27786*, 2026. 2, 4, 5, 7, 19, 21, 22, 25

[70] C. Qian, D. Chételat, and C. Morris. Exploring the power of graph neural networks in solving linear optimization problems. In *AISTATS*, 2024. 1, 2, 3

[71] A. Rashwan, K. Briggs, C. Budd, and L. M. Kreusser. Enforcing convex constraints in graph neural networks. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. 3

[72] F. Rendl, G. Rinaldi, and A. Wiegele. A branch and bound algorithm for max-cut based on combining semidefinite and polyhedral relaxations. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 295–309. Springer, 2007. 2

[73] F. Rendl, G. Rinaldi, and A. Wiegele. Solving Max-Cut to optimality by intersecting semidefinite and polyhedral relaxations. *Math. Programming*, 121(2):307, 2010. 3, 4, 7, 23, 26

[74] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008. 1, 2

[75] L. Scavuzzo, K. Aardal, A. Lodi, and N. Yorke-Smith. Machine learning augmented branch and bound for mixed integer linear programming. *Mathematical Programming*, pages 1–44, 2024. 3

[76] B. Tang, E. B. Khalil, and J. Drgona. Learning to optimize for mixed-integer non-linear programming with feasibility guarantees. In *Workshop on Differentiable Learning of Combinatorial Algorithms*, 2025. 3

[77] M. Tanneau and P. Van Hentenryck. Dual lagrangian learning for conic optimization. *NeurIPS*, 2024. 3

[78] J. Tordesillas, J. P. How, and M. Hutter. Rayen: Imposition of hard convex constraints on neural networks. *arXiv preprint arXiv:2307.08336*, 2023. 3

[79] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM review*, 38(1):49–95, 1996. 2

[80] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. 2

[81] Y. Wang, H. Lan, and Y. Ye. A tuning-free primal-dual splitting algorithm for large-scale semidefinite programming. *arXiv preprint arXiv:2402.00311*, 2024. 2, 21

[82] B. Weisfeiler and A. Leman. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series*, 2(9):12–16, 1968. 4, 16

[83] Z. Wen, D. Goldfarb, and W. Yin. Alternating direction augmented lagrangian methods for semidefinite programming. *Mathematical Programming Computation*, 2(3):203–230, 2010. 23

[84] H. Wolkowicz, R. Saigal, and L. Vandenberghe. *Handbook of semidefinite programming: theory, algorithms, and applications*, volume 27. Springer Science & Business Media, 2012. 2

[85] C. Wu, Q. Chen, A. Wang, T. Ding, R. Sun, W. Yang, and Q. Shi. On representing convex quadratically constrained quadratic programs via graph neural networks. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. 2

[86] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. 2, 4

[87] M. Yau, N. Karalias, E. Lu, J. Xu, and S. Jegelka. Are graph neural networks optimal approximation algorithms? *NeurIPS*, 2024. 2, 3

[88] G. Zarpellon, J. Jo, A. Lodi, and Y. Bengio. Parameterizing branch-and-bound search trees to learn branching policies. In *Proceedings of the aaai conference on artificial intelligence*, volume 35, pages 3931–3939, 2021. 3

[89] J. Zhou, Y. Wang, and P. Li. Branching strategies based on subgraph gnns: A study on theoretical promise versus practical reality. *arXiv preprint arXiv:2512.09355*, 2025. 3

A Extended Background

A.1 Higher-Order Weisfeiler-Leman

Let $n \geq 1$ and $[n] \coloneqq \{1, \ldots, n\}$. We denote multisets with $\{\!\{\ldots\}\!\}$. A labeled graph $G = (V, E, l)$ consists of a finite node set V, an edge set E, and a labeling function $l \colon V \to \mathbb{N}$. The neighborhood of a node $v \in V$ is defined as $N(v) \coloneqq \{u \in V \mid (u, v) \in E\}$. Two graphs G and H are isomorphic ($G \simeq H$) if there exists an adjacency- and label-preserving bijection between their node sets. For k-tuples $\boldsymbol{v}, \boldsymbol{w} \in V^k$, we say they share the same atomic type, denoted $\mathsf{atp}(\boldsymbol{v}) = \mathsf{atp}(\boldsymbol{w})$, if the coordinate mapping $v_i \mapsto w_i$ induces a partial isomorphism.

The 1-Weisfeiler-Leman (1-WL) algorithm, also known as color refinement, is a standard heuristic for testing graph isomorphism [82]. It determines if two graphs are non-isomorphic by iteratively updating node colors based on local neighborhoods. Initialized with $\boldsymbol{c}_v^0 \coloneqq l(v)$, the color of node v at iteration $t > 0$ is updated via an injective hash function:

$$\boldsymbol{c}_v^t \coloneqq \mathsf{hash}\big(\boldsymbol{c}_v^{t-1}, \{\!\{\boldsymbol{c}_u^{t-1} \mid u \in N(v)\}\!\}\big).$$

The algorithm terminates when the partition of colors stabilizes, i.e., $\boldsymbol{c}_v^t = \boldsymbol{c}_w^t \iff \boldsymbol{c}_v^{t+1} = \boldsymbol{c}_w^{t+1}$ for all $v, w \in V$, yielding a *stable coloring* $\boldsymbol{c}_v^\infty$. If two graphs yield a differing number of nodes for any stable color, 1-WL successfully distinguishes them [13].

To distinguish more complex symmetries where 1-WL fails, the hierarchy generalizes to k-tuples where $k \geq 2$, yielding the k-dimensional WL (k-WL) and its folklore variant (k-FWL) [8, 41, 13, 34]. Both algorithms initialize tuple colors using their atomic types: $\boldsymbol{c}_{\boldsymbol{v}}^0 \coloneqq \mathsf{atp}(\boldsymbol{v})$. They differ primarily in how they aggregate information when substituting a node $w \in V$ into the j-th position of $\boldsymbol{v}$, denoted as $\phi_j(\boldsymbol{v}, w) \coloneqq (v_1, \ldots, v_{j-1}, w, v_{j+1}, \ldots, v_k)$.

The k-FWL algorithm captures the *joint* configuration of these substitutions across all tuple positions simultaneously:

$$\boldsymbol{c}_{\boldsymbol{v}}^t \coloneqq \mathsf{hash}\Big(\boldsymbol{c}_{\boldsymbol{v}}^{t-1}, \Big\{\!\!\Big\{ \Big(\boldsymbol{c}_{\phi_1(\boldsymbol{v},w)}^{t-1}, \ldots, \boldsymbol{c}_{\phi_k(\boldsymbol{v},w)}^{t-1}\Big) \mid w \in V \Big\}\!\!\Big\}\Big).$$

In contrast, standard k-WL aggregates the multiset of each substitution position *independently*:

$$\boldsymbol{c}_{\boldsymbol{v}}^t \coloneqq \mathsf{hash}\Big(\boldsymbol{c}_{\boldsymbol{v}}^{t-1}, \Big(\{\!\{\boldsymbol{c}_{\phi_1(\boldsymbol{v},w)}^{t-1} \mid w \in V\}\!\}, \ldots, \{\!\{\boldsymbol{c}_{\phi_k(\boldsymbol{v},w)}^{t-1} \mid w \in V\}\!\}\Big)\Big).$$

Both algorithms iterate until the tuple colors stabilize.

We quantify expressive power using refinement relations. An algorithm $\mathcal{A}$ *refines* $\mathcal{B}$, denoted $\mathcal{A} \sqsubseteq \mathcal{B}$, if $\mathcal{A}$ can distinguish all pairs of graphs that $\mathcal{B}$ can distinguish. Strict refinement is denoted as $\mathcal{A} \sqsubset \mathcal{B}$. Because k-FWL aggregates joint neighborhood structures rather than marginal distributions, it is strictly more expressive than k-WL for $k \geq 2$. Specifically, k-FWL is proven to be equivalent in power to $(k{+}1)$-WL [13, 34], establishing the following strict expressivity hierarchy:

$$\cdots \equiv (k+1)\text{-FWL} \sqsubset (k+1)\text{-WL} \equiv k\text{-FWL} \sqsubset k\text{-WL} \equiv (k{-}1)\text{-FWL} \sqsubset \ldots$$

A.2 Branch-and-Bound Method

The branch-and-bound method is a tree-search algorithm for solving combinatorial optimization problems exactly.

As shown in Algorithm 1, the solver starts with the SDP relaxation of the Max-Cut Problem at the B&B root node, and repeatedly selects an active node from the queue to create subproblems. By default, we select the node with the highest upper bound to explore. Each subproblem is generated by branching on an unfixed variable in the current graph. In our implementation, the branching variable is chosen by the most-fractional rule. The branching variable is then fixed to the two possible partitions, producing two child nodes.

The subproblem associated with each child node is relaxed and evaluated as an SDP. This returns an upper bound together with a primal SDP solution. The Goemans–Williamson algorithm then rounds this primal solution to a feasible cut. Specifically, the primal solution is factorized as $X = VV^T$ with row vectors v_i. Random vectors r are sampled to induce a partition, where the variables with

$\{i|v_i^\top r \geq 0\}$ can be assigned to one partition and the remaining variables are assigned to the other. The feasible cut is then further improved by a local search that flips individual variables to increase the lower bound. If the new cut is better, the incumbent will be updated. Since the SDP relaxation provides an upper bound on the best-cut value achievable by the current subproblem, this child node can be safely pruned whenever its SDP optimal objective value is no better than the incumbent lower bound. The pruning condition is $\mathrm{UB}_c < \mathrm{LB} + 1$, as all edge weights are integers. In this case, neither this node nor any of its descendants can yield a better feasible cut. Otherwise, the node remains active and is added back to the queue.

In this work, we consider three branch-and-bound solvers: (i) a vanilla SDP-based B&B solver; (ii) a hybrid B&B solver; and (iii) a neural B&B solver. While the vanilla solver uses Mosek to solve the SDP relaxation at the node-evaluation step, where $\mathrm{UB}_c = f^*_{\mathrm{SDP}}$, the hybrid and neural solvers apply the proposed GNN to predict a dual-feasible solution and generate a valid upper bound such that $f^*_{\mathrm{SDP}} \leq f_{\mathrm{GNN}}$. Therefore, the same pruning condition can be applied directly if $f_{\mathrm{GNN}} < \mathrm{LB}+1$. The hybrid solver uses the GNN for early fathoming. At each node, the GNN is queried first to produce f_{GNN}. If this bound already prunes the node, the exact SDP evaluation with Mosek will be bypassed. Otherwise, Mosek is invoked to compute the tighter bound, f_{SDP}, and the algorithm proceeds exactly as in the vanilla solver. As a result, the hybrid solver preserves the same branch-and-bound tree, but changes the cost of node evaluation. The nodes pruned by GNN avoid solving the SDP with Mosek, whereas surviving nodes incur the cost of both GNN inference and the subsequent Mosek solve. In contrast, the neural solver differs more significantly from the vanilla solver. The Mosek solve is removed entirely from this solver, and the GNN is responsible for the prediction of both primal and dual solutions. Specifically, the neural solver considers $\mathrm{UB}_c = f_{\mathrm{GNN}}$ and uses the predicted primal solution in the Goemans–Williamson algorithm. These changes can alter node priorities in the queue and branching decisions, resulting in a different branch-and-bound tree. Despite this different search trajectory, the neural solver can explore nodes much more efficiently by combining lightweight GNN inference with batched evaluation of child nodes.

B Omitted proofs

In the following, we outline proofs omitted from the main paper.

B.1 Proof of radial projection

Consider the primal problem

$$\begin{aligned} \max_{\boldsymbol{X} \in \mathbb{S}^n_+} \quad & \langle \boldsymbol{L},\, \boldsymbol{X} \rangle \\ \text{s.t.} \quad & \operatorname{diag}(\boldsymbol{X}) = \mathbf{e}. \end{aligned}$$

We add dual variables $\boldsymbol{y} \in \mathbb{R}^n$ and $\boldsymbol{S} \in \mathbb{S}^n_+$, and construct the Lagrangian as

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{S}) \coloneqq -\langle \boldsymbol{L},\, \boldsymbol{X} \rangle + \boldsymbol{y}^\intercal \left(\operatorname{diag}(\boldsymbol{X}) - \mathbf{e}\right) - \langle \boldsymbol{S},\, \boldsymbol{X} \rangle .$$

Using the property that Diag is the adjoint operator of diag, we have $\boldsymbol{y}^\intercal \mathcal{A}(\boldsymbol{X}) = \langle \mathcal{A}^*(\boldsymbol{y}),\, \boldsymbol{X} \rangle$. And the Lagrangian becomes

$$\mathcal{L}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{S}) \coloneqq -\langle \boldsymbol{L},\, \boldsymbol{X} \rangle - \boldsymbol{y}^\intercal \mathbf{e} + \langle \operatorname{Diag}(\boldsymbol{y}),\, \boldsymbol{X} \rangle - \langle \boldsymbol{S},\, \boldsymbol{X} \rangle .$$

Bounding the primal problem requires the supremum of $\mathcal{L}$ over $\boldsymbol{X}$ to be finite, which occurs only if the coefficient of $\boldsymbol{X}$ vanishes, yielding $\boldsymbol{L} - \operatorname{Diag}(\boldsymbol{y}) + \boldsymbol{S} = \mathbf{0}$. The remainder is $-\boldsymbol{e}^\intercal \boldsymbol{y}$, which is to be maximized. Therefore, the dual problem is

$$\begin{aligned} \min_{\boldsymbol{y} \in \mathbb{R}^n, \boldsymbol{S} \in \mathbb{S}^n_+} \quad & \langle \boldsymbol{e},\, \boldsymbol{y} \rangle \\ \text{s.t.} \quad & \boldsymbol{L} - \operatorname{Diag}(\boldsymbol{y}) + \boldsymbol{S} = 0. \end{aligned}$$

Before proving the theory on radial projection, we first prove a simple lemma.

Lemma B.1. *Let $\boldsymbol{M}$ be a square matrix whose smallest eigenvalue is λ_0. Then, for any scalar $c \in \mathbb{R}$, the smallest eigenvalue of $\boldsymbol{M} + c\boldsymbol{I}$ is $\lambda_0 + c$.*

Algorithm 1 branch and bound for Max-Cut

Require: Problem P, relaxation method $\mathcal{R} \in \{\text{VANILLA}, \text{HYBRID}, \text{NEURAL}\}$, top K nodes
$Q \leftarrow \emptyset$ ▷ Queue
$(\text{UB}_r, x_r) \leftarrow \text{EVALUATENODE}(P, \mathcal{R})$ ▷ Root upper bound
$x_r^{\text{heur}} \leftarrow \text{GW-ALGORITHM}(x_r)$
$x_{\text{LB}} \leftarrow x_r^{\text{heur}}$, $\text{LB} \leftarrow x_r^{\text{heur}} L_r x_r^{\text{heur}}$ ▷ Best solution and lower bound
$\text{PUSH}(Q, r)$
while $Q \neq \emptyset$ **do**
  $\mathcal{N} \leftarrow \text{POPMANY}(Q, K)$ ▷ Best (top K, if $\mathcal{R} = \text{NEURAL}$) upper bound first
  $\mathcal{C} \leftarrow [\,]$
  for all $n \in \mathcal{N}$ **do**
    if $\text{LB} + 1 \leq \text{UB}_n$ **then**
      $j^* \leftarrow \text{MOSTFRACTIONAL}(n)$
      $(c^{\text{down}}, c^{\text{up}}) \leftarrow \text{BRANCH}(n, j^*)$
      $\mathcal{C} \leftarrow \text{APPEND}(\mathcal{C}, c^{\text{down}}, c^{\text{up}})$
    end if
  end for
  for all $c \in \mathcal{C}$ **do**
    $(\text{UB}_c, x_c) \leftarrow \text{EVALUATENODE}(c, \mathcal{R})$ ▷ Parallel if $\mathcal{R} = \text{NEURAL}$; sequential otherwise
    $x_c^{\text{heur}} \leftarrow \text{GW-ALGORITHM}(x_c)$ ▷ Parallel if $\mathcal{R} = \text{NEURAL}$; sequential otherwise
    if $x_c^{\text{heur}} L_c x_c^{\text{heur}} > \text{LB}$ **then**
      $x_{\text{LB}} \leftarrow x_c^{\text{heur}}$, $\text{LB} \leftarrow x_c^{\text{heur}} L_c x_c^{\text{heur}}$ ▷ Update lower bound
    end if
    if $\text{UB}_c < \text{LB} + 1$ **then**
      $\text{PRUNENODE}(c)$ ▷ Prune by optimality
    else
      $\text{PUSH}(Q, c)$
    end if
  end for
end while
return $(x_{\text{LB}}, \text{LB})$

Proof. For any eigenvalue-eigenvector pair $(\lambda_i, \boldsymbol{v}_i)$ $\boldsymbol{M}$, it holds that $\boldsymbol{M}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$ by definition. Consequently, $(\boldsymbol{M} + c\boldsymbol{I})\,\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i + c\boldsymbol{v}_i$, and thus every eigenvalue is shifted to $\lambda_i + c$ for the matrix $\boldsymbol{M} + c\boldsymbol{I}$.

Now consider the smallest eigenvalue λ_0, it will be shifted to $\lambda_0 + c$, this completes the proof. □

Proposition B.2. *The radial projection as in Section 2.2 always yields a feasible dual solution.*

Proof. To establish dual feasibility, we must show that the projected variables satisfy all constraints of the dual formulation:

1. $\hat{\boldsymbol{y}}^{\text{feas}} \in \mathbb{R}^n$
2. $\hat{\boldsymbol{S}}^{\text{feas}} \in \mathbb{S}_+^n$
3. $\boldsymbol{L} - \text{Diag}(\hat{\boldsymbol{y}}^{\text{feas}}) + \hat{\boldsymbol{S}}^{\text{feas}} = \boldsymbol{0}$

First, the neural network predicts an unconstrained vector $\hat{\boldsymbol{y}} \in \mathbb{R}^n$. We define the corresponding slack matrix as $\hat{\boldsymbol{S}} \coloneqq \text{Diag}(\hat{\boldsymbol{y}}) - \boldsymbol{L}$, which satisfies the equality constraint (Condition 3) by construction. To correct this, the radial projection shifts the variables as follows:

$$\hat{\boldsymbol{y}}^{\text{feas}} \coloneqq \hat{\boldsymbol{y}} + \max\{0, -\lambda_{\min}(\hat{\boldsymbol{S}})\}\boldsymbol{e}$$

$$\hat{\boldsymbol{S}}^{\text{feas}} \coloneqq \hat{\boldsymbol{S}} + \max\{0, -\lambda_{\min}(\hat{\boldsymbol{S}})\}\boldsymbol{I}$$

Condition 1 trivially holds as $\hat{\boldsymbol{y}}^{\text{feas}}$ remains in $\mathbb{R}^n$. By applying the Lemma with $c = \max\{0, -\lambda_{\min}(\hat{\boldsymbol{S}})\}$, the minimum eigenvalue of $\hat{\boldsymbol{S}}^{\text{feas}}$ becomes $\lambda_{\min}(\hat{\boldsymbol{S}}) + \max\{0, -\lambda_{\min}(\hat{\boldsymbol{S}})\}$, which is 0 if $\lambda_{\min}(\hat{\boldsymbol{S}}) < 0$, and is strictly positive otherwise. Thus, $\hat{\boldsymbol{S}}^{\text{feas}} \succeq \mathbf{0}$, satisfying Condition 2.

Finally, we verify Condition 3 for the projected variables:

$$\begin{aligned}\boldsymbol{L} - \operatorname{Diag}(\hat{\boldsymbol{y}}^{\text{feas}}) + \hat{\boldsymbol{S}}^{\text{feas}} &= \boldsymbol{L} - \operatorname{Diag}\big(\hat{\boldsymbol{y}} + c\boldsymbol{e}\big) + \big(\hat{\boldsymbol{S}} + c\boldsymbol{I}\big) \\ &= \boldsymbol{L} - \operatorname{Diag}(\hat{\boldsymbol{y}}) - c\boldsymbol{I} + \hat{\boldsymbol{S}} + c\boldsymbol{I} \\ &= \boldsymbol{L} - \operatorname{Diag}(\hat{\boldsymbol{y}}) + \hat{\boldsymbol{S}} = \mathbf{0}.\end{aligned}$$

Since all three conditions are satisfied, the radial projection yields a strictly feasible dual solution. □

B.2 Proof of Theorem 2.1

Theorem B.3 (Restatement of Theorem 2.1)**.** *Let $\boldsymbol{X}^* \in \mathbb{S}^n_+$ and $\boldsymbol{y}^*$ denote the optimal primal and dual solutions, respectively, for a given instance of the Max-Cut SDP relaxation. For any indices (i, j) and (p, q), if the stable colors produced by δ-MC-WL satisfy $\boldsymbol{v}^\infty_{ij} = \boldsymbol{v}^\infty_{pq}$, then the corresponding primal entries satisfy $X^*_{ij} = X^*_{pq}$. Furthermore, if the diagonal stable colors satisfy $\boldsymbol{v}^\infty_{kk} = \boldsymbol{v}^\infty_{ll}$, then the dual solution satisfies $y^*_k = y^*_l$.*

We first prove a lemma that our δ-MC-WL refines the spectral property of a symmetric matrix, similar to VC-2-FWL in prior work [69], but different in derivation details.

Lemma B.4. *Let $\boldsymbol{M} \in \mathbb{S}^n$ be a symmetric matrix with spectral decomposition*

$$\boldsymbol{M} = \sum_{k=1}^{m} \lambda_k \boldsymbol{P}_k$$

where $\lambda_1 > \lambda_2 \cdots > \lambda_m$ are the non repeating eigenvalues, and $\boldsymbol{P}_k$ are Frobenius covariants [39, p. 437], i.e., the symmetric matrices describing the projection onto the eigenspace of eigenvalue λ_k.

The stable coloring $\boldsymbol{v}^\infty$ are produced by the multiset δ-MC-WL algorithm:

$$\begin{aligned}\boldsymbol{v}^0_{ij} &\coloneqq \mathsf{hash}\,(M_{ij}, \mathbb{I}_{i=j}) \\ \boldsymbol{v}^t_{ij} &\coloneqq \mathsf{hash}\Big(\boldsymbol{v}^{t-1}_{ij}, \big\{\!\big\{(\boldsymbol{v}^{t-1}_{iu}, M_{uj}) \mid u \in [n], M_{uj} \neq 0\big\}\!\big\} \oplus \big\{\!\big\{(\boldsymbol{v}^{t-1}_{uj}, M_{iu}) \mid u \in [n], M_{iu} \neq 0\big\}\!\big\}\Big)\end{aligned}$$

where $\mathbb{I}_{i=j}$ is a diagonal indicator which takes value 1 if $i = j$ otherwise 0. The following result holds:

$$\boldsymbol{v}^\infty_{ij} = \boldsymbol{v}^\infty_{pq} \implies (\boldsymbol{P}_k)_{ij} = (\boldsymbol{P}_k)_{pq}, \quad \text{for all } k \in [m].$$

Proof. The proof is based on the fact that the spectral projectors $\boldsymbol{P}_k$ are polynomials in $\boldsymbol{M}$ via Sylvester's formula. We say a coloring $\boldsymbol{v}$ refines a matrix $\boldsymbol{A}$, defined as:

$$\boldsymbol{v}_{ij} = \boldsymbol{v}_{pq} \implies A_{ij} = A_{pq}.$$

We proceed by induction to demonstrate that the multiset aggregation effectively simulates matrix multiplication: if the δ-MC-WL coloring at an iteration refines the entries of $\boldsymbol{M}^d$, the next iteration refines $\boldsymbol{M}^{d+1}$. Consequently, the stable coloring refines any polynomial of $\boldsymbol{M}$, and by extension, naturally refines all the specific polynomials defining the spectral projectors.

At initialization $t = 0$, the coloring of $\boldsymbol{v}^0$ refines $\boldsymbol{M}^1$ naturally, as they are injectively defined by $\boldsymbol{M}$. Besides, $\boldsymbol{v}^0$ also refines $\boldsymbol{M}^0 = \boldsymbol{I}$, because of the diagonal indicator $\mathbb{I}_{i=j}$.

Assume that at iteration t, the coloring $\boldsymbol{v}^t$ refines the matrix power $\boldsymbol{M}^d$:

$$\boldsymbol{v}^t_{ij} = \boldsymbol{v}^t_{pq} \implies \big(\boldsymbol{M}^d\big)_{ij} = \big(\boldsymbol{M}^d\big)_{pq}, \text{ for all } i, j, p, q.$$

We show that the next iteration $\boldsymbol{v}^{t+1}$ will refine $\boldsymbol{M}^{d+1}$:

$$\boldsymbol{v}^{t+1}_{ij} = \boldsymbol{v}^{t+1}_{pq} \implies \big(\boldsymbol{M}^{d+1}\big)_{ij} = \big(\boldsymbol{M}^{d+1}\big)_{pq}.$$

Specifically, we have the following derivation for δ-MC-WL:

$$
\begin{aligned}
& \boldsymbol{v}_{ij}^{t+1} = \boldsymbol{v}_{pq}^{t+1} \\
\implies & \left\{\!\!\left\{ (\boldsymbol{v}_{iu}^{t}, M_{uj}) \mid u \in [n], M_{uj} \neq 0 \right\}\!\!\right\} \oplus \left\{\!\!\left\{ (\boldsymbol{v}_{uj}^{t}, M_{iu}) \mid u \in [n], M_{iu} \neq 0 \right\}\!\!\right\} = \\
& \left\{\!\!\left\{ (\boldsymbol{v}_{pu}^{t}, M_{uq}) \mid u \in [n], M_{uq} \neq 0 \right\}\!\!\right\} \oplus \left\{\!\!\left\{ (\boldsymbol{v}_{uq}^{t}, M_{pu}) \mid u \in [n], M_{pu} \neq 0 \right\}\!\!\right\} && \text{(i)} \\
\implies & \left\{\!\!\left\{ ((\boldsymbol{M}^d)_{iu}, M_{uj}) \mid u \in [n], M_{uj} \neq 0 \right\}\!\!\right\} \oplus \left\{\!\!\left\{ ((\boldsymbol{M}^d)_{uj}, M_{iu}) \mid u \in [n], M_{iu} \neq 0 \right\}\!\!\right\} = \\
& \left\{\!\!\left\{ ((\boldsymbol{M}^d)_{pu}, M_{uq}) \mid u \in [n], M_{uq} \neq 0 \right\}\!\!\right\} \oplus \left\{\!\!\left\{ ((\boldsymbol{M}^d)_{uq}, M_{pu}) \mid u \in [n], M_{pu} \neq 0 \right\}\!\!\right\} && \text{(ii)} \\
\implies & \left\{\!\!\left\{ (\boldsymbol{M}^d)_{iu} M_{uj} \mid u \in [n], M_{uj} \neq 0 \right\}\!\!\right\} \oplus \left\{\!\!\left\{ (\boldsymbol{M}^d)_{uj} M_{iu} \mid u \in [n], M_{iu} \neq 0 \right\}\!\!\right\} = \\
& \left\{\!\!\left\{ (\boldsymbol{M}^d)_{pu} M_{uq} \mid u \in [n], M_{uq} \neq 0 \right\}\!\!\right\} \oplus \left\{\!\!\left\{ (\boldsymbol{M}^d)_{uq} M_{pu} \mid u \in [n], M_{pu} \neq 0 \right\}\!\!\right\} && \text{(iii)} \\
\implies & \left(\sum_{u\colon M_{uj} \neq 0} (\boldsymbol{M}^d)_{iu} M_{uj} \right) \oplus \left(\sum_{u\colon M_{iu} \neq 0} (\boldsymbol{M}^d)_{uj} M_{iu} \right) \\
= & \left(\sum_{u\colon M_{uq} \neq 0} (\boldsymbol{M}^d)_{pu} M_{uq} \right) \oplus \left(\sum_{u\colon M_{pu} \neq 0} (\boldsymbol{M}^d)_{uq} M_{pu} \right) && \text{(iv)} \\
\implies & \left(\sum_{u\colon M_{uj} \neq 0} (\boldsymbol{M}^d)_{iu} M_{uj} + \sum_{u\colon M_{uj} = 0} (\boldsymbol{M}^d)_{iu} M_{uj} \right) \\
& \oplus \left(\sum_{u\colon M_{iu} \neq 0} (\boldsymbol{M}^d)_{uj} M_{iu} + \sum_{u\colon M_{iu} = 0} (\boldsymbol{M}^d)_{uj} M_{iu} \right) \\
= & \left(\sum_{u\colon M_{uq} \neq 0} (\boldsymbol{M}^d)_{pu} M_{uq} + \sum_{u\colon M_{uq} = 0} (\boldsymbol{M}^d)_{pu} M_{uq} \right) \\
& \oplus \left(\sum_{u\colon M_{pu} \neq 0} (\boldsymbol{M}^d)_{uq} M_{pu} + \sum_{u\colon M_{pu} = 0} (\boldsymbol{M}^d)_{uq} M_{pu} \right) && \text{(v)} \\
\implies & \left(\sum_{u} (\boldsymbol{M}^d)_{iu} M_{uj} \right) \oplus \left(\sum_{u} (\boldsymbol{M}^d)_{uj} M_{iu} \right) \\
= & \left(\sum_{u} (\boldsymbol{M}^d)_{pu} M_{uq} \right) \oplus \left(\sum_{u} (\boldsymbol{M}^d)_{uq} M_{pu} \right) \\
\implies & (\boldsymbol{M}^d \boldsymbol{M})_{ij} \oplus (\boldsymbol{M} \boldsymbol{M}^d)_{ij} = (\boldsymbol{M}^d \boldsymbol{M})_{pq} \oplus (\boldsymbol{M} \boldsymbol{M}^d)_{pq} \\
\implies & (\boldsymbol{M}^{d+1})_{ij} = (\boldsymbol{M}^{d+1})_{pq}.
\end{aligned}
$$

Among the derivations, (i) holds because it is a partial definition; (i) to (ii) holds because $\boldsymbol{v}^t$ refines $\boldsymbol{M}^d$ by hypothesis; (ii) to (iii) holds because of injectivity, if the injective function of $\mathsf{hash}(a, b) = \mathsf{hash}(c, d)$, then $\mathsf{hash}(a \cdot b) = \mathsf{hash}(c \cdot d)$; (iii) to (iv) also holds because of injectivity; if two multisets of real values are equal, then the sum aggregation of their elements must also be equal; (iv) to (v) is straightforward because it is adding zeros to (iv).

Therefore, $\boldsymbol{v}_{ij}^{t+1} = \boldsymbol{v}_{pq}^{t+1} \implies \left(\boldsymbol{M}^{d+1}\right)_{ij} = \left(\boldsymbol{M}^{d+1}\right)_{pq}$.

By induction, the stable coloring $\boldsymbol{v}^\infty$ refines $\boldsymbol{M}^d$ for all $d \in \mathbb{N}$. Consequently, it also refines any matrix polynomial $p(\boldsymbol{M}) = \sum_{d=0}^{\infty} a_d \boldsymbol{M}^d$. This holds because the expressivity to distinguish the structural features of individual powers $\boldsymbol{M}^d$ implies the ability to distinguish their linear combinations. The real-valued matrix $\boldsymbol{M}$ is symmetric and diagonalizable, therefore we apply spectral decomposition $\boldsymbol{M} = \sum_{k=1}^{m} \lambda_k \boldsymbol{P}_k$, where matrices $\boldsymbol{P}_k$ are given by the Sylvester's formula [39, p. 437]:

$$
\boldsymbol{P}_k = \prod_{j \neq k} \frac{\boldsymbol{M} - \lambda_j \boldsymbol{I}}{\lambda_k - \lambda_j}.
$$

Because each $\boldsymbol{P}_k$ is a polynomial in $\boldsymbol{M}$, the stable coloring $\boldsymbol{v}^\infty$ refines each $\boldsymbol{P}_k$. Thus, it holds for all k that $\boldsymbol{v}_{ij}^\infty = \boldsymbol{v}_{pq}^\infty \implies (\boldsymbol{P}_k)_{ij} = (\boldsymbol{P}_k)_{pq}$. □

Next, we adapt the proof of VC-2-FWL to fit δ-MC-WL, using the PDHG algorithm as a proof technique. As SDPs can have multiple solutions, which complicates analysis, we also adopt the assumption of Qian and Morris [69] that the SDP has a unique solution with the minimum Frobenius norm.

Our work for Max-Cut relaxations is a special case of SDPs, as the $\mathcal{A}$ is a diagonal operation. We define the operator:

$$\text{diag}\colon \mathbb{S}^n \to \mathbb{R}^n, \quad \text{diag}(\boldsymbol{X}) = (X_{11}, X_{22}, \ldots, X_{nn})^\intercal$$

and its adjoint:

$$\text{Diag}\colon \mathbb{R}^n \to \mathbb{S}^n, \quad \text{Diag}(\boldsymbol{y}) = \begin{bmatrix} y_1 & 0 & \cdots & \\ 0 & y_2 & \cdots & \\ & & \ddots & 0 \\ & \cdots & 0 & y_n \end{bmatrix}.$$

The PDHG updates for the Frobenius-regularized SDP are given by Qian and Morris [69]:

$$\begin{aligned} \boldsymbol{X}^{t+1} &\coloneqq \text{Proj}_{\mathbb{S}_+^n}\left[\frac{\boldsymbol{X}^t - \alpha_t \text{Diag}(\boldsymbol{y}^t) - \alpha_t \boldsymbol{C}}{1 + \alpha_t \varepsilon}\right] \\ \boldsymbol{y}^{t+1} &\coloneqq \boldsymbol{y}^t + \beta_t \text{diag}\left(\boldsymbol{X}^{t+1} + \theta_t\left(\boldsymbol{X}^{t+1} - \boldsymbol{X}^t\right)\right) - \beta_t \boldsymbol{b}. \end{aligned} \tag{5}$$

The Proj function projects a symmetric matrix $\boldsymbol{X} \in \mathbb{S}^n$ onto the PSD cone by finding the closest PSD matrix $\boldsymbol{X}_{\mathbb{S}_+^n}$ that minimizes the Frobenius norm distance $\|\boldsymbol{X}_{\mathbb{S}_+^n} - \boldsymbol{X}\|_F$. Given the spectral decomposition of the input matrix $\boldsymbol{X} = \sum_{i=1}^n \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^\top$, where λ_i are the real eigenvalues and $\boldsymbol{v}_i$ are the corresponding orthonormal eigenvectors. The projected matrix is then reconstructed by truncating any negative eigenvalues to zero: $\boldsymbol{X}_{\mathbb{S}_+^n} = \sum_{i=1}^n \max(\lambda_i, 0) \cdot \boldsymbol{v}_i \boldsymbol{v}_i^\top$.

For any $\varepsilon > 0$, the SDP problem has a unique primal solution $\boldsymbol{X}^*$. And if $\varepsilon \to 0$, it reduces to a normal linear SDP. We initialize with $\boldsymbol{X}^0 = \boldsymbol{0}_{n \times n}$ and $\boldsymbol{y}^0 = \boldsymbol{0}_m$.

Now we proceed with our proof.

Proof of Theorem 2.1. We provide a constructive proof by simulating the primal-dual hybrid gradient (PDHG) algorithm [16, 81]. We show that the stable coloring of δ-MC-WL refines every update of the PDHG algorithm, which is proven to converge in Qian and Morris [69], Wang et al. [81].

Our goal is to show that for all $t \geq 0$, the stable δ-MC-WL coloring on variable and constraint nodes refines the intermediate primal-dual solutions of PDHG updates:

$$\begin{aligned} \boldsymbol{v}_{ij}^\infty = \boldsymbol{v}_{pq}^\infty &\implies X_{ij}^t = X_{pq}^t \\ \boldsymbol{v}_{kk}^\infty = \boldsymbol{v}_{ll}^\infty &\implies y_k^t = y_l^t. \end{aligned}$$

Before the inductive step, we establish three key facts derived from the properties of the stable coloring.

Fact 1 Since the stable coloring $\boldsymbol{v}^\infty$ refines the initial coloring $\boldsymbol{v}^0$, and $\boldsymbol{v}^0$ is defined by $\boldsymbol{C}$, it holds that $\boldsymbol{v}_{ij}^\infty = \boldsymbol{v}_{pq}^\infty \implies C_{ij} = C_{pq}$. Furthermore, by applying Lemma B.4 to $\boldsymbol{C}$, if its spectral decomposition is given by $\boldsymbol{C} = \sum_h \lambda_h \boldsymbol{P}_h$, then the projection matrices also respect the coloring: $\forall h\colon \boldsymbol{v}_{ij}^\infty = \boldsymbol{v}_{pq}^\infty \implies (\boldsymbol{P}_h)_{ij} = (\boldsymbol{P}_h)_{pq}$. Besides, we notice that the constraints are uniform for Max-Cut, i.e., $\boldsymbol{b} = \boldsymbol{e}$, therefore, it does not contribute to separation power.

Fact 2 We notice that the **diagonal** and **off-diagonal** elements receive a different set of colors at the initialization, due to the diagonal indicator $\mathbb{I}$, and their color sets remain disjoint throughout the update of δ-MC-WL.

Proof by induction We now prove the theorem by induction on t. At initialization $t = 0$, we have $\boldsymbol{X}^0 = \boldsymbol{0}$ and $\boldsymbol{y}^0 = \boldsymbol{0}$. The implications $\boldsymbol{v}_{ij}^\infty = \boldsymbol{v}_{pq}^\infty \implies X_{ij}^0 = X_{pq}^0$ and $\boldsymbol{v}_{kk}^\infty = \boldsymbol{v}_{ll}^\infty \implies y_k^0 = y_l^0$ hold trivially.

Suppose the hypothesis holds for iteration $t \geq 0$. We show it holds for iteration $t+1$.

1. For diagonal ones

During the primal update

$$\boldsymbol{X}^{t+1} \coloneqq \mathrm{Proj}_{\mathbb{S}^n_+}\left[\frac{\boldsymbol{X}^t - \alpha_t \mathrm{Diag}(\boldsymbol{y}^t) - \alpha_t \boldsymbol{C}}{1+\alpha_t\varepsilon}\right],$$

consider the term inside the PSD projection in:

$$\boldsymbol{Z}^t \coloneqq \frac{\boldsymbol{X}^t - \alpha_t \mathrm{Diag}(\boldsymbol{y}^t) - \alpha_t \boldsymbol{C}}{1+\alpha_t\varepsilon},$$

since $\boldsymbol{X}^t$ and $\mathrm{Diag}(\boldsymbol{y}^t)$ (by hypothesis), $\boldsymbol{C}$ (by Fact 1), are all refined by the coloring $\boldsymbol{v}^\infty$, their linear combination $\boldsymbol{Z}^t$ also is. With Fact 2, one can easily verify this by inspecting the diagonal and off-diagonal elements separately. For off-diagonals $i \neq j$ and $p \neq q$, if $\boldsymbol{v}^\infty_{ij} = \boldsymbol{v}^\infty_{pq}$, then $(\boldsymbol{X}^t - \alpha_t \boldsymbol{C})_{ij} = (\boldsymbol{X}^t - \alpha_t \boldsymbol{C})_{pq}$. And for diagonals, if $\boldsymbol{v}^\infty_{kk} = \boldsymbol{v}^\infty_{ll}$, then $(\boldsymbol{X}^t - \alpha_t \mathrm{Diag}(\boldsymbol{y}^t) - \alpha_t \boldsymbol{C})_{kk} = (\boldsymbol{X}^t - \alpha_t \mathrm{Diag}(\boldsymbol{y}^t) - \alpha_t \boldsymbol{C})_{ll}$.

Then, the update is $\boldsymbol{X}^{t+1} = \mathrm{Proj}_{\mathbb{S}^n_+}(\boldsymbol{Z}^t)$. The PSD projection depends only on the spectral decomposition. Specifically, spectral decomposition is performed on $\boldsymbol{Z}^t$, and the Frobenius covariants corresponding to negative eigenvalues are removed. By Lemma B.4, we know $\forall(i,j,p,q)\colon \boldsymbol{v}^\infty_{ij} = \boldsymbol{v}^\infty_{pq} \implies X^{t+1}_{ij} = X^{t+1}_{pq}$, that is, $\boldsymbol{v}^\infty$ refines $\boldsymbol{X}^{t+1}$.

Finally, in the dual update step, we evaluate the term inside the operator diag:

$$\boldsymbol{W}^{t+1} \coloneqq \boldsymbol{X}^{t+1} + \theta_t\left(\boldsymbol{X}^{t+1} - \boldsymbol{X}^t\right).$$

Since both $\boldsymbol{X}^{t+1}$ and $\boldsymbol{X}^t$ are refined by $\boldsymbol{v}^\infty$, so is $\boldsymbol{W}^{t+1}$. Therefore, $\mathrm{diag}(\boldsymbol{W}^{t+1})$ is naturally refined by $(\boldsymbol{v}^\infty_{11}, \cdots, \boldsymbol{v}^\infty_{nn})^\intercal$. Since $\boldsymbol{y}^t$ (by hypothesis) and $\boldsymbol{b}$ (by Fact 1) are both refined by the coloring $(\boldsymbol{v}^\infty_{11}, \cdots, \boldsymbol{v}^\infty_{nn})^\intercal$, the sum $\boldsymbol{y}^{t+1}$ is refined by the coloring.

In summary, we have proven by induction that if the stable coloring $\boldsymbol{v}^\infty$ refines $\boldsymbol{X}^t$ and $\boldsymbol{y}^t$ in the PDHG algorithm, they also refine $\boldsymbol{X}^{t+1}$ and $\boldsymbol{y}^{t+1}$.

Since PDHG is guaranteed to converge to $\boldsymbol{X}^*$ and $\boldsymbol{y}^*$, this completes the proof. □

B.3 Proof of Neuralization

Proposition B.5. *There exists a set of parameters for the functions* INIT, UPD, MSG, *and* MAP *such that the neural architectures* MC-MPNN *and* δ-MC-MPNN *possess maximal expressivity equivalent to their corresponding algorithms.*

The proof follows the universal approximation arguments for multiset functions established in Qian and Morris [69].

Proof. To prove that the neural architectures achieve the exact expressive power of their corresponding WL algorithms, we rely on the universal approximation of injective multiset functions. Following Qian and Morris [69] and Maron et al. [57], let $\mathcal{X}$ be a space of bounded multisets. *There exists a continuous, injective function parameterized by an MLP that maps any bounded multiset to a unique real-valued vector representation.*

Since our architectures are deterministic unrollings of the MC-WL and δ-MC-WL algorithms, they cannot distinguish variables that the WL tests cannot. Thus, we only need to show that the neural networks are at most as expressive as the WL injective hash functions, meaning that the neural network layers can injectively simulate them.

MC-MPNN can be derived from VC-2-FWL [69], the aggregation simply replaces the standard bipartite, variable-to-constraint message passing with variable-to-variable topology. The sum over $u \in [n]$, combined with the MLPs MSG and MAP, injectively encodes the neighborhood multisets, identical to the variable-node update proof in Qian and Morris [69].

For the sparse variant δ-MC-MPNN, the algorithm aggregates over conditionally filtered multisets based on the sparse topology of $\boldsymbol{C}$. The neural formulation handles this by explicitly masking the

messages:

$$\sum_{\substack{u\in[n]\\ C_{iu}\neq 0}} \mathsf{MSG}^t\big(\boldsymbol{h}_{uj}^{t-1}, C_{iu}\big)$$

Because the matrix entries C_{iu} belong to a compact space, the MSG function can be parameterized to injectively map the pair $(\boldsymbol{h}_{uj}^{t-1}, C_{iu})$ to a latent vector. The summation over the non-zero indices perfectly implements the multiset encoding of the sparse neighbors. Finally, the update function UPD^t concatenates the outputs of these independent multiset summations and injectively maps them to the next layer's feature space.

Therefore, there exists a set of weights for which the neural architectures perfectly simulate the injective hash functions of their WL algorithms, establishing equivalent expressive power. □

C Details of experiment design

Comparison of GNN variants on uniform dataset All GNN variants are set to 10 layers with a hidden dimension of 128. The models are trained using the Adam optimizer [44] with default hyperparameters and batch size of 128. Experiments were conducted on a compute cluster equipped with four NVIDIA L40S GPUs. We train for a maximum of 1000 epochs, using a scheduler that decays the learning rate if the averaged dual objective on the validation set fails to improve for 100 consecutive epochs. To prevent overfitting, early stopping is triggered after 200 epochs without validation improvement. We report the results averaged over 3 random initialization seeds.

Despite the sparsity of the underlying graphs, we implement δ-MC-MPNN using dense matrix multiplications, as modern GPUs are heavily optimized for dense operations, and we find they process them significantly faster than batched sparse-matrix alternatives.

GNN training on B&B dataset Training is performed using 4 NVIDIA L40S GPUs for at most 1000 epochs, with early stopping triggered after 100 consecutive epochs without validation improvement. On average, each epoch requires 63 seconds for dynamic data generation and 51 seconds for model optimization, for a total of approximately 114 seconds per epoch. If all 1000 epochs are completed, this corresponds to approximately 31.7 hours of end-to-end wall-clock time, or 126.7 allocated GPU-hours. The VRAM allocation is 23.2 GB in total. Importantly, the resulting g05_100 model is reused without retraining for the g05_60, g05_80, and g05_100 branch-and-bound experiments.

B&B experiments on Max-Cut dataset The GNN used in B&B experiments are set to 6 layers with a hidden dimension of 96. With a batch size of 32, We first train the GNN with dual heads for 1000 epochs under the same protocol, then freeze these parameters and train the primal heads for another 200 epochs with patience 100. In the Section 3, each benchmark instance family contains ten different instances. We solve all instances to global optimality without a timeout and report the average number of evaluated nodes and solving time.

D Additional experiments

D.1 Ablation: Choice of SDP solver

To justify our selection of Mosek [6] as our primary classical solver, we conduct an empirical performance comparison across the Max-Cut instances from Rendl et al. [73], Gusmeroli et al. [35]. Specifically, we evaluate the computational time required to solve the SDP relaxation at the root node using three distinct optimization paradigms: SDPLR, a non-linear approach based on the Burer-Monteiro method [12]; Mosek [6], a state-of-the-art interior-point method; and SCS [62], a first-order solver utilizing the ADMM algorithm [83]. The experiments are conducted on the AMD EPYC 7662 CPU. The solving times are summarized in Table 4.

As demonstrated in Table 4, Mosek consistently outperforms both the first-order and Burer-Monteiro methods by a substantial margin across all evaluated instance sizes and densities. This superior computational efficiency strongly motivates our exclusive use of Mosek for exact SDP evaluations throughout our branch-and-bound framework.

Table 4: Comparison of mean solving times (in seconds) for the root-node SDP relaxation across various Max-Cut instance families.

Instance Family	Mosek	SCS	SDPLR
g05_60	0.0146	0.1040	0.1423
g05_80	0.0235	0.2135	0.2023
g05_100	0.0380	0.4145	0.3018
g05_180	0.1208	2.0893	1.0229
pm1s_80	0.0221	0.1760	0.1521
pm1s_100	0.0347	0.3192	0.1792
pm1s_180	0.1135	1.7188	0.4433
pw01_100	0.0376	0.7706	0.1894
pw01_180	0.1158	3.7741	0.5261
pw05_100	0.0399	0.9922	0.3982
pw05_180	0.1268	7.0904	1.4823
pw09_100	0.0408	1.1123	0.3519
pw09_180	0.1307	6.4214	1.1196
w01_100	0.0373	0.6187	0.1872
w01_180	0.1179	9.5305	0.5013
w05_100	0.0384	1.0688	0.3590
w05_180	0.1230	4.9640	1.3138
w09_100	0.0393	0.9056	0.3445
w09_180	0.1265	4.7360	1.0651

D.2 Ablation: Number of layers

We identify a key structural limitation of δ-MC-MPNN at shallow depths. As shown in Table 5, δ-MC-MPNN yields a significantly higher objective gap than MC-MPNN when utilizing only 4 layers. Although this gap becomes marginal in deeper networks, MC-MPNN consistently performs better. This difference intuitively stems from how each architecture expands its structural receptive field. Because MC-MPNN jointly aggregates pairs of previous embeddings, it simulates matrix multiplication. At layer t, it effectively refines $\boldsymbol{C}^{2^t}$. Conversely, δ-MC-MPNN aggregates features with the static matrix $\boldsymbol{C}$, restricting its refinement to $\boldsymbol{C}^{t+1}$ at layer t. Therefore, while δ-MC-MPNN remains theoretically valuable, the rapid field expansion of MC-MPNN makes it more favorable when network depth is limited.

Table 5: Relative objective gap (%) under varying network depths on the $N = 100, p = 0.1$ ER graph dataset.

Model	# Layers		
	4	6	10
MC-MPNN	**0.931**$\pm$**0.001**	**0.656**$\pm$**0.002**	**0.598**$\pm$**0.005**
δ-MC-MPNN	1.403$\pm$0.004	0.872$\pm$0.002	0.628$\pm$0.004

D.3 Ablation: OOD size generalization

To evaluate the out-of-distribution (OOD) size generalization of our architectures, we test models trained on smaller ER graph instances with $N = 100$ against progressively larger graphs ($N = 150$ and $N = 200$) while maintaining the edge probability $p = 0.1$. As detailed in Table 6, both MC-MPNN and δ-MC-MPNN exhibit a steady increase in the relative objective gap as the graph size scales up, highlighting the inherent difficulty of zero-shot size extrapolation. While both architectures degrade on larger instances, MC-MPNN consistently shows a lower objective gap across all tested sizes, confirming that it is better equipped to generalize to unseen, larger-scale topologies.

Table 6: Size generalization performance (relative objective gap %) on larger graphs. Models were evaluated on $N \in \{150, 200\}$ with fixed edge probability $p = 0.1$.

Dataset	MC-MPNN	δ-MC-MPNN
N=100, p=0.1	0.598±0.005	0.628±0.004
N=150, p=0.1	2.545±0.476	3.201±0.547
N=200, p=0.1	4.089±0.453	4.712±0.466

D.4 Ablation: Variants of normalization and GNNs

To further understand the architectural sensitivities of our model, we conducted an ablation study evaluating the impact of different normalization strategies.

As mentioned in the Qian and Morris [69], they add normalization layers between GNN updates to stabilize training. For a mini-batch of B graphs, variable embeddings are zero-padded to form a uniform tensor of shape $B \times n_{\max} \times n_{\max} \times d$, where $n_{\max}$ is the maximum graph size in the batch and d is the feature dimension. Because the update MLPs contain biases, padded entries can distort normalization statistics. To investigate this, we formalize and ablate three normalization strategies:

The SpatialNorm layer normalizes each feature channel $f \in [d]$ across the entire padded spatial grid. The mean μ_f and variance σ_f^2 are computed as:

$$\mu_f = \frac{1}{n_{max}^2} \sum_{i=1}^{n_{max}} \sum_{j=1}^{n_{max}} h_{ij,f}, \quad \sigma_f^2 = \frac{1}{n_{max}^2} \sum_{i=1}^{n_{max}} \sum_{j=1}^{n_{max}} (h_{ij,f} - \mu_f)^2. \tag{6}$$

Features are then standardized and shifted via learnable parameters $\gamma_f, \beta_f \in \mathbb{R}$:

$$\hat{h}_{ij,f} = \gamma_f \left(\frac{h_{ij,f} - \mu_f}{\sqrt{\sigma_f^2 + \epsilon}} \right) + \beta_f, \tag{7}$$

where ϵ is a numerical stability constant. Crucially, because statistics are aggregated over the fixed $n_{\max} \times n_{\max}$ grid, the presence of zero-padding for graphs where the true size $n < n_{\max}$ inherently skews the normalization.

As a baseline, we use LayerNorm [7], which operates independently on each variable node $\boldsymbol{h}_{ij} \in \mathbb{R}^d$. It computes the mean μ_{ij} and variance σ_{ij}^2 exclusively across the feature dimension d:

$$\mu_{ij} = \frac{1}{d} \sum_{f=1}^{d} h_{ij,f}, \quad \sigma_{ij}^2 = \frac{1}{d} \sum_{f=1}^{d} (h_{ij,f} - \mu_{ij})^2. \tag{8}$$

The vector is then scaled using learnable parameters $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^d$:

$$\hat{\boldsymbol{h}}_{ij} = \boldsymbol{\gamma} \odot \left(\frac{\boldsymbol{h}_{ij} - \mu_{ij}\mathbf{1}}{\sqrt{\sigma_{ij}^2 + \epsilon}} \right) + \boldsymbol{\beta}, \tag{9}$$

Since it is performed per-node, LayerNorm is topology-agnostic. Consequently, it is completely immune to the zero-padding artifacts that degrade SpatialNorm.

We also evaluate GraphNorm [14]. Unlike SpatialNorm, GraphNorm dynamically flattens the tensor to strictly isolate valid (unpadded) nodes. Let $\mathcal{V}_g = \{(i,j) \mid i,j \in [n]\}$ denote the actual node indices for a graph g of size n. GraphNorm computes statistics for each channel f exclusively over this valid set:

$$\mu_{g,f} = \frac{1}{|\mathcal{V}_g|} \sum_{(i,j) \in \mathcal{V}_g} h_{ij,f}, \quad \sigma_{g,f}^2 = \frac{1}{|\mathcal{V}_g|} \sum_{(i,j) \in \mathcal{V}_g} (h_{ij,f} - \mu_{g,f})^2. \tag{10}$$

Valid features are then normalized using a learnable mean-scaling parameter α_f alongside $\gamma_f, \beta_f \in \mathbb{R}$:

$$\hat{h}_{ij,f} = \gamma_f \left(\frac{h_{ij,f} - \alpha_f \mu_{g,f}}{\sqrt{\sigma_{g,f}^2 + \epsilon}} \right) + \beta_f. \tag{11}$$

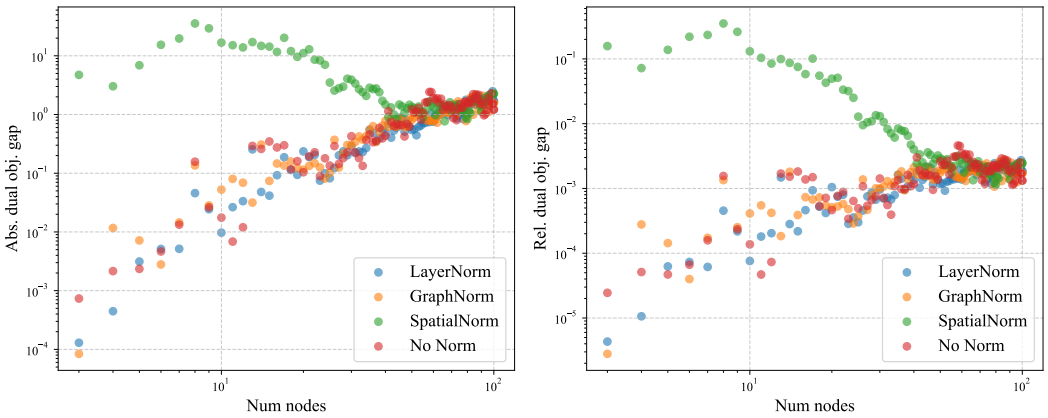

Figure 2: Effect of normalization layers on objective gaps. The absolute (left) and relative (right) dual objective gaps across different numbers of nodes in the test set are shown in the plots.

By explicitly masking padded entries, GraphNorm calculates the true statistics of each graph, completely isolating it from padding-induced distortions.

We trained the network on the Branch-and-Bound dataset with root nodes of ER graphs ($N = 100, p = 0.5$) across various graph sizes using LayerNorm, GraphNorm, SpatialNorm, and an unnormalized baseline (NoNorm), analyzing both the absolute and relative dual objective gaps on the test set. As illustrated in Figure 2, the techniques exhibit two fundamentally different scaling behaviors. SpatialNorm yields highly suboptimal performance on small graphs (e.g., fewer than 50 nodes) because the excessive zero-padding heavily distorts its spatial statistics. Consequently, it shows a distinct downward trend in error as the node count increases and the padding ratio drops. In contrast, LayerNorm, GraphNorm, and NoNorm display the exact opposite trajectory. All three achieve exceptionally low objective gaps on small graphs but exhibit a steady, monotonic increase in error as the topology expands. Interestingly, the surprisingly competitive performance of NoNorm demonstrates that our proposed architecture is inherently well-conditioned, effectively learning the underlying SDP mapping without strictly requiring intermediate normalization to stabilize training. Ultimately, despite these divergent initial trajectories, all configurations tightly converge to a similar performance baseline as the graph size approaches 100 nodes and padding vanishes.

To complement our theoretical analysis, we empirically contrast the MC-MPNN and δ-MC-MPNN variants across different normalization layers. After training on the Branch-and-Bound dataset, we assess their generalization on test instances of varying sizes. As shown in Figure 3, the scaling trend remains highly consistent between the two architectures. However, δ-MC-MPNN demonstrates comparatively weaker convergence on larger graphs, resulting in worse performance than its MC-MPNN counterpart, which is consistent with our ablation of the number of layers.

We further tested the network with LayerNorm, GraphNorm, and NoNorm , as well as MC-MPNN and δ-MC-MPNN in established Max-Cut datasets [73, 35]. Instance families cover a diverse range of graph sizes, densities, and edge weights: (1) `g05_n` (edge probability 0.5) for $n \in \{60, 80, 100, 180\}$ with edge weights in $\{0, 1\}$; (2) `pm1s_n` (edge probability 0.1) for $n \in \{80, 100, 180\}$ with edge weights in $\{-1, 0, 1\}$; (3) `w0d_n` for $n \in \{100, 180\}$ and $d \in \{1, 5, 9\}$ (edge probability 0.1, 0.5, 0.9) with integer edge weights in $[-10, 10]$; (4) `pw0d_n` for $n \in \{100, 180\}$ and $d \in \{1, 5, 9\}$ (edge probability 0.1, 0.5, 0.9) with integer edge weights in $[0, 10]$. Notably, among these families, only `g05_100` aligns with the training distribution. The proposed GNN is trained on graphs with $n = 100$ nodes, edge probability 0.5, and edge weights in $\{0, 1\}$, together with contracted subgraphs generated along random branching trajectories. As a result of contraction, these smaller subgraphs are typically denser than 0.5 and no longer uniformly weighted in $\{0, 1\}$.

We evaluated the out-of-distribution generalization of the proposed GNN by measuring the objective gap of the SDP relaxation of these instances in Figure 4. LayerNorm and NoNorm consistently achieves better objective gap on in-training-distribution instances (`g05_100`) and near-training-distribution instances (`g05_60`, `g05_80`). Similarly, the dense MC-WL variant outperforms the δ-MC-WL variant on these same families. However, LayerNorm exhibits significantly worse gener-

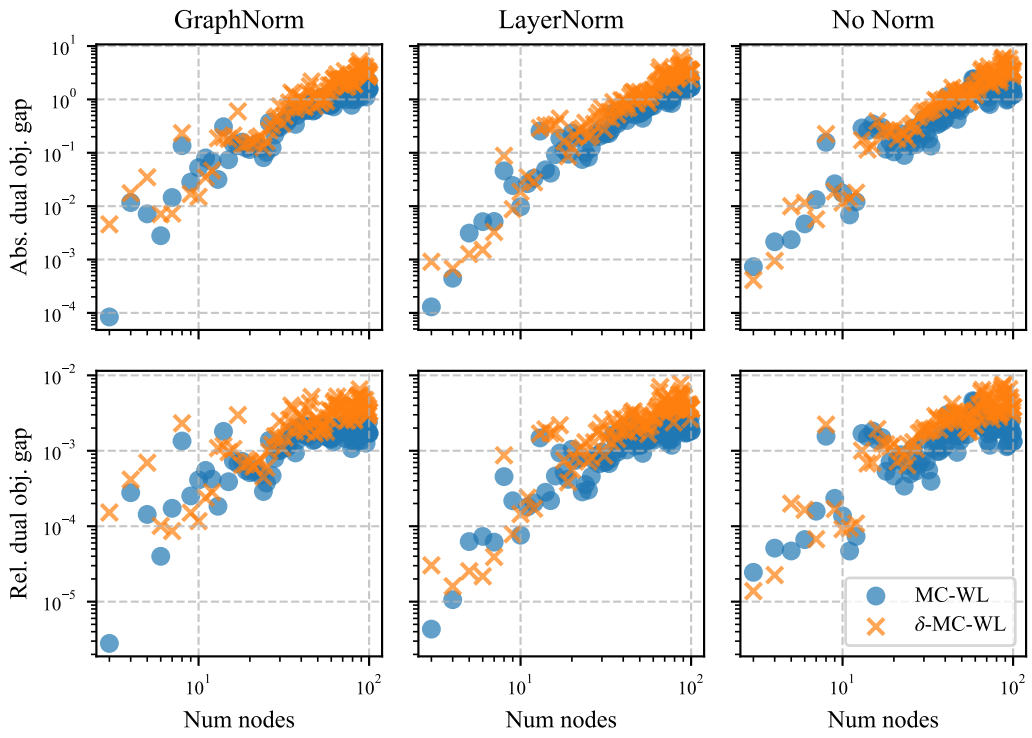

Figure 3: Performance of MC-MPNN and δ-MC-MPNN under different normalizations. Absolute (top) and relative (bottom) dual objective gaps evaluated on the test set.

Table 7: Relative objective gap (%) on SDP relaxations of in-distribution test instances. Models (LayerNorm and MC-WL) are trained within the same distribution and its subproblems.

Instance Family	Objective Gap
g05_100	0.20
pm1s_100	0.17
pm1s_80	0.19
w01_100	0.20

alization than GraphNorm in out-of-distribution instances (pm1s_n, pw0d_n, w0d_n). In particular, LayerNorm does not generalize when only the graph size differs (g05_180), while all other configurations can result in reasonably small objective gaps. However, if the GNN with LayerNorm and MC-WL is trained within a distribution where graphs are sparse and edge weights are sampled from a broader range, it still generalizes well to larger graphs as shown in Figure 5. Under GraphNorm, the generalization of MC-WL variant is slightly better when the edge-weight distribution is relatively narrow (g05, pm1s, pwd), and slightly worse when the edge-weight distribution is wider (wd). Across all configurations, the smallest objective gaps are typically achieved in instance families with edge probability 0.5 (g05_n, pw05_n, w05_n), followed by those with edge probability 0.9 (pw09_n, w09_n). In contrast, sparse instances with edge probability 0.1 (pm1s_n, pw01_n, w01_n) have the worst objective gaps. This trend aligns with the training distribution, which consists of 100-node 50% density graphs and contracted subgraphs that are usually denser than 50%. For completeness, in Table 7 we report the objective gaps on in-distribution test instances, where the test graphs follow the same distributions as those used to train the respective GNNs.

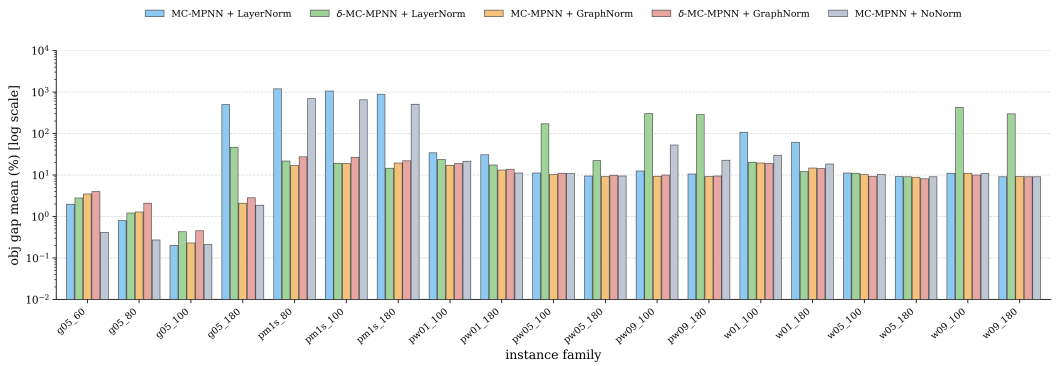

Figure 4: Effect of normalization layers and layer variants on objective gaps, trained within the distribution of `g05_100` and subproblems. The average relative dual objective gaps across different instance families are shown in the plots.

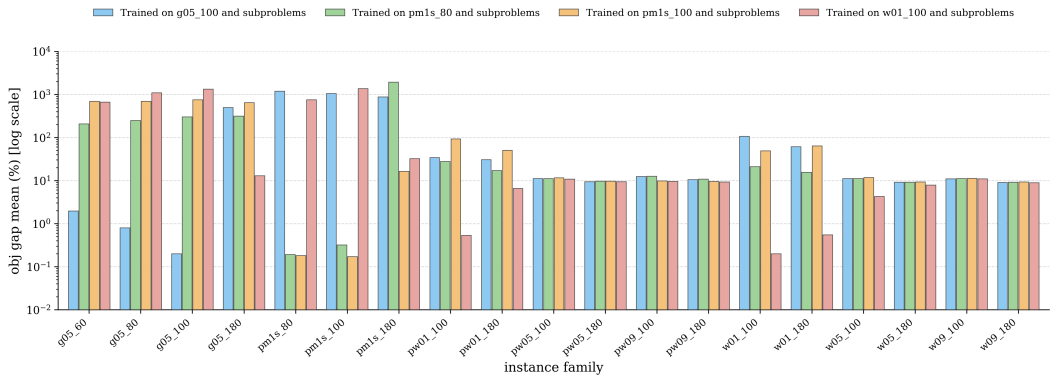

Figure 5: Effect of training distribution on objective gaps, with LayerNorm and MC-WL. The average relative dual objective gaps across different instance families are shown in the plots.

D.5 Ablation: Branch-and-bound experiments without Goemans–Williamson algorithm

We consider B&B experiments without the Goemans–Williamson algorithm to evaluate the impact of learned upper bounds only. By explicitly setting the lower bound to the optimal objective value at the root node, we eliminate the role of primal heuristics in the comparison. The following experiments are performed on the AMD EPYC 7662 CPU and NVIDIA A100 GPU.

D.6 Ablation: Additional branch-and-bound experiments with Goemans–Williamson algorithm

This section reports results of additional branch-and-bound experiments in the standard setting. In particular, we explore and compare the impact of different GNN variants, normalization choices, and top-K selection on the neural solver. The following experiments are performed on the AMD EPYC 7662 CPU and NVIDIA L40S GPU.

As an additional reference, we report results using BiqCrunch [47], a highly optimized state-of-the-art exact solver for Max-Cut. It is crucial to note, however, that the full BiqCrunch algorithm is **not directly comparable** to our proposed framework. BiqCrunch is a sophisticated branch-and-cut solver whose main performance advantage comes from dynamically strengthening the continuous SDP relaxation with separated triangle inequalities. Rather than adding all such inequalities upfront, it iteratively identifies the most violated ones and maintains a compact working set of useful cuts. The resulting strengthened SDP bounds are computed efficiently using a specialized quasi-Newton method. In contrast, the core algorithmic contribution of our work is a neural surrogate designed strictly to accelerate the evaluation of the base SDP relaxation during node enumeration, rather than

Table 8: Branch-and-bound results without Goemans–Williamson algorithm on established instances. The Hybrid and GNN methods utilize the MC-MPNN + LayerNorm model. The best results across all configurations are highlighted.

Instance Family	Method	Top-K	# Nodes Evaluated	Solving Time (s)
g05_60	Vanilla	-	3566	36.4
	Hybrid	1	3566	44.8
	Neural	1	4562	42.5
	Neural	32	**4563**	**15.1**
g05_80	Vanilla	-	49128	796.8
	Hybrid	1	49128	805.0
	Neural	1	53970	507.8
	Neural	32	**53975**	**224.4**
g05_100	Vanilla	-	1087290	25580.0
	Hybrid	1	1087290	26215.0
	Neural	1	1272820	12508.0
	Neural	32	**1272807**	**6116.3**

to replace polyhedral cutting-plane machinery. **Extending our GNN architecture to explicitly incorporate or predict such triangle inequalities remains a promising direction for future work**.

To further distinguish the effect of relaxation strength, we additionally evaluated BiqCrunch with its triangle inequalities disabled. Under this pure B&B setting, BiqCrunch becomes significantly less competitive than both our baseline and our neural solver. Without the benefit of the cuts, its quasi-Newton optimizer is less efficient at solving the basic unstrengthened SDP than the interior-point methods utilized by Mosek. Therefore, the vanilla B&B solver equipped with Mosek remains the most appropriate, direct, and rigorous baseline for evaluating the acceleration provided by our neural SDP surrogate.

Moreover, we include comparisons with three LP-based solvers, Gurobi 11.0.0, BARON 26.5.28 using CPLEX 22.2.0 as its LP/MIP subsolver, and SCIP 10.0.3. We used time limits of 30 minutes for g05_60, 1 hour for g05_80, and 12 hours for g05_100. These comparisons, together with the BiqCrunch results, are consistent with the well-established advantage of SDP relaxations in providing stronger bounds for Max-Cut than LP relaxations, and motivate our choice of an SDP-based vanilla B&B solver as the principal baseline.

Across the near-distribution families g05_60 Table 9 and g05_80 Table 10, as well as the in-distribution g05_100 Table 11 instances, the combination of the MC-MPNN variant and LayerNorm consistently yields the strongest branch-and-bound performance. Notably, the neural solver successfully generalizes to smaller subproblems without requiring retraining, demonstrating robust inference efficiency. The practical computational advantage becomes increasingly pronounced as the graph size grows, the neural solver achieves a substantial $5.58\times$ speedup on g05_60, and reaches an impressive $9.1\times$ acceleration over the vanilla solver on the g05_100 instances.

Interestingly, even though NoNorm can achieve a smaller objective gap than LayerNorm at the initial root node see Figure 4, LayerNorm performs significantly better once the GNN is dynamically deployed deeper inside the branch-and-bound tree. This implies that LayerNorm is highly effective at keeping the feature distributions stable and closer to the training distribution during subsequent branching steps. Furthermore, the batched comparison confirms that batched node evaluation is another major source of acceleration. However, this speedup does not scale linearly with K; while increasing K continues to improve total running time, it yields diminishing returns once K is sufficiently large, e.g., $K = 32$.

Finally, for extended evaluations demonstrating the solver's performance on sparser and out-of-distribution instance families, please refer to the results for pm1s_80 Table 12, pm1s_100 Table 13, and w01_100 Table 14 detailed below.

Table 9: Additional branch-and-bound results on g05_60 instance family. The neural network is trained on graphs in distribution of g05_100. The best results across all neural solver configuration are highlighted.

Method	Top-K	# Nodes Evaluated	Solving Time (s)	Speed-up (×)
Vanilla	-	3566	53.1	1.00
BiqCrunch (w/o cuts)	-	3918	153.9	0.34
BiqCrunch (w/ cuts)	-	7	2.3	26.50
MC-MPNN + LayerNorm	1	4562	34.2	1.55
	8	4562	10.0	5.30
	32	**4562**	**9.5**	**5.58**
δ-MC-MPNN + LayerNorm	1	4617	33.6	1.58
	8	4618	10.3	5.15
	32	4618	9.7	5.46
MC-MPNN + GraphNorm	1	4663	42.2	1.26
	8	4663	11.3	4.69
	32	4663	10.4	5.10
MC-MPNN + NoNorm	1	5291	39.3	1.35
	8	5291	11.8	4.49
	32	5291	11.0	4.82
MC-MPNN + SpatialNorm	1	5211	41.8	1.27
	8	6511	14.7	3.61
	32	9600	19.3	2.75

Table 10: Additional branch-and-bound results on g05_80 instance family. The neural network is trained on graphs in the distribution of g05_100. The best results across all neural solver configurations are highlighted.

Method	Top-K	# Nodes Evaluated	Solving Time (s)	Speed-up (×)
Vanilla	-	49128	1170.3	1.00
BiqCrunch (w/o cuts)	-	48633	3746.3	0.31
BiqCrunch (w/ cuts)	-	49	22.0	53.20
MC-MPNN + LayerNorm	1	53969	444.9	2.63
	8	53973	156.3	7.49
	32	**53975**	**155.7**	**7.52**
δ-MC-MPNN + LayerNorm	1	64437	494.7	2.37
	8	64461	175.2	6.68
	32	64474	179.8	6.51
MC-MPNN + GraphNorm	1	61250	595.7	1.96
	8	61243	197.1	5.94
	32	61251	203.3	5.76
MC-MPNN + NoNorm	1	65593	581.1	2.01
	8	65596	190.5	6.14
	32	65596	189.0	6.19
MC-MPNN + SpatialNorm	1	59094	521.8	2.24
	8	77691	213.4	5.48
	32	120523	314.3	3.72

Table 11: Additional branch-and-bound results on `g05_100` instance family. The neural network is trained on graphs in the distribution of `g05_100`. The best results across all neural solver configurations are highlighted.

Method	Top-K	# Nodes Evaluated	Solving Time (s)	Speed-up (×)
Vanilla	-	1087290	40930.1	1.00
BiqCrunch (w/ cuts)	-	390	270.7	151.20
MC-MPNN + LayerNorm	1	1272822	11791.0	3.47
	32	**1272801**	**4509.3**	**9.08**
δ-MC-MPNN + LayerNorm	32	1660692	5716.0	7.16
MC-MPNN + GraphNorm	32	1326781	5438.9	7.53

Table 12: Additional branch-and-bound results on `pm1s_80` instance family. The neural network is trained on graphs in the distribution of `pm1s_80`. The best results across all neural solver configurations are highlighted.

Method	Top-K	# Nodes Evaluated	Solving Time (s)	Speed-up (×)
Vanilla	-	25656	609.5	1.00
BiqCrunch (w/ cuts)	-	2	1.2	507.92
MC-MPNN + LayerNorm	1	27809	212.5	2.87
	32	**27809**	**80.8**	**7.54**

Table 13: Additional branch-and-bound results on `pm1s_100` instance family. The neural network is trained on graphs in the distribution of `pm1s_100`. The best results across all neural solver configurations are highlighted.

Method	Top-K	# Nodes Evaluated	Solving Time (s)	Speed-up (×)
Vanilla	-	254408	10382.5	1.00
BiqCrunch (w/ cuts)	-	13	11.7	887.39
MC-MPNN + LayerNorm	1	302311	2518.5	4.12
	32	**302314**	**1058.2**	**9.81**

Table 14: Additional branch-and-bound results on `w01_100` instance family. The neural network is trained on graphs in the distribution of `w01_100`. The best results across all neural solver configurations are highlighted.

Method	Top-K	# Nodes Evaluated	Solving Time (s)	Speed-up (×)
Vanilla	-	269407	11348.6	1.00
BiqCrunch (w/ cuts)	-	4	4.1	2767.95
MC-MPNN + LayerNorm	1	305716	2689.8	4.22
	32	**305720**	**1073.9**	**10.57**

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims in the document are properly justified from Section 2 to 4, which include the Max-Cut-specific GNN architecture, GNN-enhanced branch-and-bound framework, and empirical results.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The main document includes a separate subsection to discuss the proposed framework's limitations in terms of generalization on OOD instances and integration with cutting planes, with supporting experimental details in appendix

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper presents several theoretical results, each accompanied by its assumptions and proofs in the respective sections and the appendix.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The proposed method is clearly described, and all the information required for main experiments and ablation studies is provided in the main text and appendix.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The data, code, and instructions needed to reproduce experimental results are provided in the supplementary material.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper provides detailed descriptions of datasets, test instances, and experimental protocols. These include data splits, training hyperparameters and procedures, optimizer settings, and solver configurations are outlined in the main document and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The experimental results of GNN variant comparison report mean and standard deviation over 3 random initialization seeds. For the branch-and-bound experiments, results are averaged across benchmark instances within each instance family.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The computer resources, per-epoch training time, and branch-and-bound solving time are all reported in the main document and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms with the NeurIPS Code of Ethics. The experiments are performed on publicly available problems that do not involve human subjects or sensitive data. Therefore, ethical considerations such as fairness, privacy, and misuse are not applicable in this context.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

Justification: The paper focuses on methodological advances in combinatorial optimization and does not explicitly discuss societal impacts. While the proposed techniques could indirectly benefit applications such as logistics, network design, and resource allocation, we do not analyze potential positive or negative societal impacts in this work.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper poses no such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We use the Mosek solver (academic license) and publicly available instances from BiqMac and BiqBin libraries, along with the open-source BiqCrunch solver. All assets are properly cited and used in accordance with their respective terms of use.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release any new datasets or pretrained models. All assets used are either synthetically generated under controlled settings or drawn from established public benchmark collections.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing and research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: No IRB approval is needed in this work as it does not involve human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: No LLMs were used in the core method development. Any assistance from LLMs was limited to non-substantive editing.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.