NEURAL COMBINATORIAL OPTIMIZATION ON HYPERGRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Along with AI computing shining in scientific discovery, neural combinatorial optimization (NCO) has shown promise across diverse problems. Yet, existing neural solvers struggle to solve combinatorial optimization problems (COPs) on hypergraphs (e.g., hypergraph max-cut, hypergraph partitioning), due to limited computational frameworks. In this work, we propose **HyperNCO**, an NCO framework for hypergraph COPs. We contribute: **(I)** A unified one-hot encoded polynomial unconstrained binary optimization (PUBO) formulation for modeling hypergraph COPs; **(II)** GPU-accelerated algorithms for one-hot encoded PUBO problems; **(III)** A Gini coefficient-based continuous relaxation annealing strategy to ensure high-quality solutions while preventing convergence to local optima. Experimental results demonstrate that HyperNCO achieves competitive solution quality compared to existing heuristics and state-of-the-art neural solvers, while offering significant advantages in computational efficiency and framework generality.

1 INTRODUCTION

The majority of combinatorial optimization problems (COPs) are NP-hard, making large-scale combinatorial optimization (CO) instances computationally intractable. In the field of neural combinatorial optimization (NCO), advances in AI computing frameworks enabled by GPU acceleration have motivated a surge in exploring end-to-end neural solvers Ye et al. (2023); Tian et al. (2024); Li et al. (2024); Cappart et al. (2023) to address COPs. Among these neural solvers, a class of physics-inspired approaches — such as physics-inspired graph neural networks (PI-GNN) Schuetz et al. (2022a); Ichikawa (2024); Schuetz et al. (2022b) and free-energy machine (FEM) Shen et al. (2025) — employ unified frameworks to solve diverse problems including max-cut, maximum independent set (MIS), and graph partitioning.

To achieve generality, they formulate COPs as quadratic unconstrained binary optimization (QUBO) problems (e.g., max-cut, MIS) or one-hot encoded QUBO problems (e.g., graph coloring, graph partitioning). On this basis, these unconstrained objectives are utilized as loss functions, optimized via straightforward gradient descent in an unsupervised manner. As the NCO framework illustrated in Fig. 1, the QUBO problem is to optimize the following cost function:

$$\min \quad O_{\mathrm{QUBO}} = \sum_{i=1}^{n}\sum_{j=1}^{n} Q_{ij}x_i x_j = \mathbf{x}^\top Q\mathbf{x} \tag{1}$$

where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ is an n-dimensional vector of binary decision variables $x_i \in \{0, 1\}$. The QUBO matrix Q is either symmetric or in upper triangular form.

QUBO aligns with NCO in three aspects: **(I)** Unconstrained cost functions that are well-suited for unsupervised training. Constraints of COPs are incorporated into the cost functions as penalties Glover et al. (2022); **(II)** Activation functions that are well-suited for the relaxation of binary variables $x_i \in [0, 1]$, such as the sigmoid function, enabling gradient-based optimization of the originally binary variables; **(III)** Efficient loss functions $\mathbf{x}^\top Q\mathbf{x}$ in a unified formulation that can be computed via GPU-supported matrix operations. The training pipeline of an NCO framework is illustrated in Fig. 1, incorporating a case study of the MIS problem in Appendix E.

While the efficiency of these unified NCO frameworks has been validated on a variety of QUBO problems for graphs, they face a fundamental limitation in hypergraphs. There, each hyperedge

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

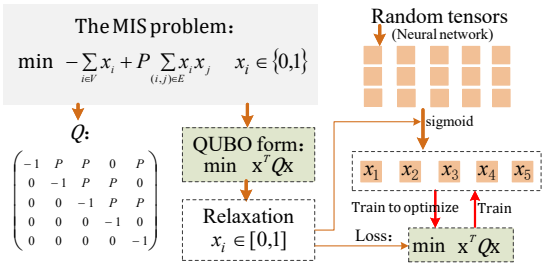


Figure 1: A unified NCO framework can be applied to QUBO problems. Finding MISs on graph $G = (V, E)$ is a typical example of a QUBO problem, where P is the penalty factor.

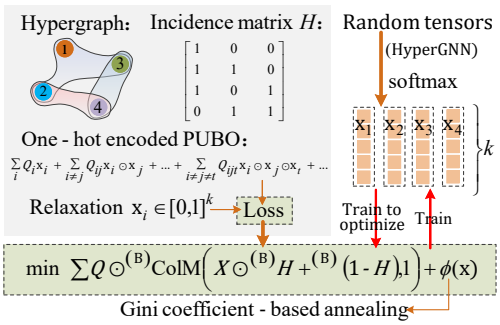


Figure 2: The HyperNCO framework. Please refer to Sec. 4 for detailed explanation.

connects an arbitrary-sized subset of vertices, quadratic terms like $x_i x_j$ cannot represent these high-order interactions (e.g., coloring conflicts within hyperedges). To address this challenge, NCO frameworks such as PI-GNN claim they can be trivially extended to polynomial unconstrained binary optimization (PUBO) problems for hypergraphs (e.g., hypergraph max-cut, hypergraph partitioning). However, this extension introduces multi-variable correlations that fundamentally increase the complexity of the loss function. Formally, the PUBO problem is then to optimize the following cost function:

$$\min \quad O_{\text{PUBO}} = \sum_i Q_i x_i + \sum_{i \neq j} Q_{ij} x_i x_j + \sum_{i \neq j \neq t} Q_{ijt} x_i x_j x_t + \cdots \tag{2}$$

where $x_i \in \{0, 1\}$ are binary decision variables. $Q = [Q_i, Q_{ij}, Q_{ijt}, \cdots]$ are coefficients for monomials in Eq. 2.

Challenge. However, unlike the QUBO loss $\mathbf{x}^\top Q\mathbf{x}$ — which enjoys a unified, GPU-accelerated formulation — the PUBO cost function in Eq. 2 lacks a naturally unified form that is computationally efficient. This limitation significantly constrains the scalability of unsupervised NCO frameworks on large-scale PUBO problems and their multi-label variants (where decision variables $\mathbf{x}_i \in \{0, 1\}^k$ are one-hot vectors), including hypergraph max-cut, coloring, and partitioning. To address this challenge, we present **HyperNCO**, a unified framework for **Hyper**graph **NCO**. We contribute:

1) A Unified One-hot Encoded PUBO Formulation. One-hot encoded PUBO extends PUBO for 2-labeling problems such as hypergraph MIS and max-cut into k-labeling problems. On this basis, we present novel cost functions for modeling several hypergraph COPs (e.g., hypergraph partitioning, max-k-cut, and coloring) via Hadamard products of one-hot vectors.

2) GPU-accelerated Differentiable Optimization. Based on the one-hot encoded PUBO formulation, we propose a training pipeline to enhance the scalability of HyperNCO when applied to large-scale COPs.

3) Gini Coefficient-based Annealing. A continuous relaxation annealing strategy serves a dual purpose: in the high-temperature phase, it broadly searches the solution space to escape local optima; in the low-temperature phase, it enforces the discreteness of solutions.

2 Related Works

As a unified unsupervised NCO solver, PI-GNN Schuetz et al. (2022a) introduced QUBO and PUBO formulation from quantum computing into NCO. Building upon PI-GNN, a series of neural solvers, such as value classification network (VCN) Chen et al. (2024) and continuous relaxation annealing (CRA) Ichikawa (2024), have been proposed. These neural solvers are tailored for binary (2-labeling) problems. To solve k-labeling problems, another line of unsupervised neural solvers, such as GAP Nazi et al. (2019), Deep Modularity Networks (DMN) Tsitsulin et al. (2023), and Potts model-based solvers Schuetz et al. (2022b); Tönshoff et al. (2023), leverage problem-specific cost functions to solve graph partitioning, clustering, and coloring problems.

FEM Shen et al. (2025) is another physics-inspired NCO framework. It leverages the concept of free-energy minimization from statistical physics to solve COPs and can be efficiently implemented using deep learning frameworks (e.g., PyTorch, TensorFlow). FEM is a unified framework, which provides a physical interpretation of the optimization process. It can be viewed as an unsupervised NCO solver featuring the CRA Ichikawa (2024) annealing mechanism. By leveraging QUBO and Potts models, FEM adapts to diverse COPs, including MIS, max-cut, and graph partitioning. However, both PI-GNN and FEM face limitations in solving the hypergraph COPs.

3 Definitions and Notations

Hypergraph. A hypergraph $G = (V, E)$ is defined as a set of nodes $V = \{v_1, v_2, \cdots, v_{|V|}\}$ and a set of hyperedges $E = \{e_1, e_2, \cdots, e_{|E|}\}$, where each hyperedge $e_j \in E$ is a subset of V. A hypergraph G can be represented by an incidence matrix $H \in \{0,1\}^{|V|\times|E|}$, where $H_{ij} = 1$ if $v_i \in e_j$, and $H_{ij} = 0$ otherwise.

Notations. Let $G = (V, E)$ be a hypergraph with vertices $v_i \in V$. We define:

- $\mathbf{x}_i$: the decision variable for vertex v_i, represented as a k-dimensional one-hot vector indicating its discrete state.
- $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^\top \in \mathbb{R}^{n\times k}$: the matrix collecting all state vectors.
- $\sum \mathbf{x}$ (or $\sum X$): the sum of all entries of vector $\mathbf{x}$ (or matrix X), i.e., $\sum_j x_j$ and $\sum_{i,j} X_{ij}$ respectively.
- $\phi(\mathbf{x})$: the Gini coefficient-based penalty term defined in Eq. 9 (Sec. 4.3).

4 Method

As shown in Fig. 2, HyperNCO employs a two-stage framework to solve hypergraph COPs: **(I)** Formulating COPs (e.g., hypergraph max-cut, partitioning, coloring) as computationally efficient one-hot encoded PUBO cost functions; **(II)** These cost functions are relaxed into differentiable loss functions, which are then optimized in an unsupervised manner using gradient-based optimizers.

4.1 One-hot Encoded PUBO

The standard PUBO formalism in Eq. 2 is restricted to binary (2-labeling) problems, such as hypergraph MIS and max-cut. However, many hypergraph COPs—including partitioning and coloring—are inherently multi-valued (k-labeling). To generalize PUBO to these cases, HyperNCO represents each vertex $v_i \in V$ using a k-dimensional one-hot vector $\mathbf{x}_i$, where V is the set of vertices of a hypergraph $G = (V, E)$. In conventional PUBO, constraints over a hyperedge e_j (e.g., distinguishing dominators and dominatees in the MIS problem) are typically expressed via the product of variables $\prod_{v_i\in e_j} x_i$. In our one-hot encoded formalism, we extend the idea using the Hadamard product ($\odot$). For a hyperedge containing vertices $v_1, v_2, \cdots, v_n$, the constraint that all vertices share the same label is captured by the expression:

$$\sum(\mathbf{x}_1 \odot \mathbf{x}_2 \odot \cdots \odot \mathbf{x}_n) = \begin{cases} 1, & \text{same label} \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

which reduces to the standard PUBO product in the binary case, ensuring the generality of HyperNCO.

One-hot Encoded PUBO Formalism. The cost functions of one-hot encoded PUBO problems can be expressed as:

$$\min \quad O_{\mathrm{OH-PUBO}} = \sum \Big(\sum_i Q_i \mathbf{x}_i + \sum_{i\neq j} Q_{ij}\mathbf{x}_i \odot \mathbf{x}_j + \sum_{i\neq j\neq t} Q_{ijt}\mathbf{x}_i \odot \mathbf{x}_j \odot \mathbf{x}_t + \cdots\Big) \tag{4}$$

One-hot encoded PUBO in Eq. 4 is a unified formalism to describe various COPs, including QUBO, PUBO, and k-labeling problems. For example, in hypergraph proper coloring, the Hadamard product $\mathbf{x}_1 \odot \mathbf{x}_2 \odot \mathbf{x}_3$ captures the color conflicts within a hyperedge $e_j = \{v_1, v_2, v_3\}$. Below, we show that such cost functions can be efficiently computed via GPU-accelerated matrix operations.

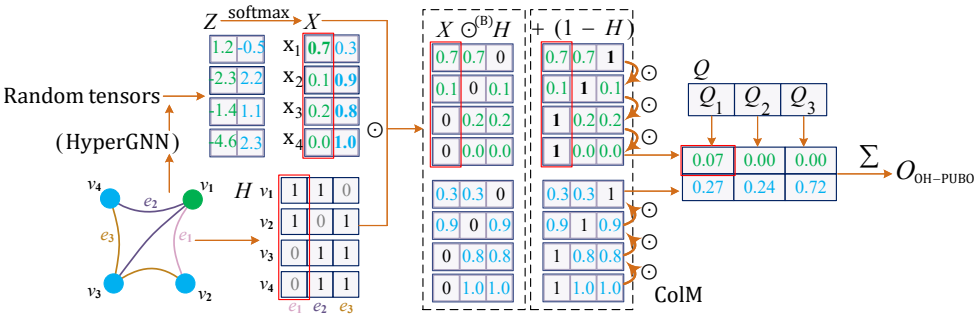


Figure 3: Illustration of the GPU-accelerated algorithm on the hypergraph proper coloring problem.

GPU-accelerated Algorithm. Given a hypergraph $G = (V, E)$, where each hyperedge $e \in E$ corresponds to a monomial (e.g., $Q_{ijt}\mathbf{x}_i \odot \mathbf{x}_j \odot \mathbf{x}_t$) in Eq. 4. Let $H \in \{0,1\}^{|V|\times|E|}$ be the incidence matrix of G. We define the PUBO vector $Q \in \mathbb{R}^{1\times|E|}$ as $Q = [Q_1, Q_2, \cdots, Q_{|E|}]$, where Q_j is the coefficient for the term that corresponds hyperedge e_j. Then,

$$O_{\text{OH-PUBO}} = \sum Q \odot^{(\text{B})} \text{ColM}\big(X \odot^{(\text{B})} H +^{(\text{B})} (1 - H)\big) \tag{5}$$

where $+^{(\text{B})}$ and $\odot^{(\text{B})}$ denote element-wise addition and Hadamard product, respectively, via the broadcasting mechanism[1]. The notation $\text{ColM}(M) = \prod_{i=1}^{|V|} M_{i,j,c}$ denotes the column-wise multiplication along the first dimension (i.e., across rows) of tensor $M \in \mathbb{R}^{|V|\times|E|\times k}$. To efficiently evaluate Eq. 4 on GPUs, we reformulate it into a fully tensorized matrix expression Eq. 5.

A toy example on hypergraph proper coloring is presented in Fig. 3. For the hypergraph $G = (V, E)$ with four vertices v_1, v_2, v_3, v_4 and three hyperedges $e_1 = \{v_1, v_2\}$, $e_2 = \{v_1, v_3, v_4\}$, $e_3 = \{v_2, v_3, v_4\}$, the PUBO vector is $Q = [Q_1, Q_2, Q_3]$. Thus, $O_{\text{OH-PUBO}} = \sum (Q_1\mathbf{x}_1 \odot \mathbf{x}_2 + Q_2\mathbf{x}_1 \odot \mathbf{x}_3 \odot \mathbf{x}_4 + Q_3\mathbf{x}_2 \odot \mathbf{x}_3 \odot \mathbf{x}_4)$. Given the output $X = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4]^\top \in \mathbb{R}^{|V|\times k}$ (softmax-transformed outputs from a neural network layers or random tensors) and incidence matrix H, the broadcasted Hadamard product $X \odot^{(\text{B})} H$ yields a tensor of size $|V| \times |E| \times k$. By adding $(1 - H)$ via broadcasting, entries of H that are 0 are replaced by 1, thereby preparing the tensor for the subsequent column-wise multiplication $\text{ColM}(\cdot)$. Finally, we broadcast the PUBO vector Q to terms $\mathbf{x}_1 \odot \mathbf{x}_2$, $\mathbf{x}_1 \odot \mathbf{x}_3 \odot \mathbf{x}_4$, and $\mathbf{x}_2 \odot \mathbf{x}_3 \odot \mathbf{x}_4$.

Time Complexity. For $X \in \mathbb{R}^{|V|\times k}$, $Q \in \mathbb{R}^{1\times|E|}$, and $H \in \mathbb{R}^{|V|\times|E|}$, the operation in Eq. 4 has time complexity $O(k|V||E|)$. GPU acceleration significantly speeds up the computation. Element-wise operations such as Hadamard product are fully parallelizable. Column-wise reduction such as sum or product over $|V|$ uses parallel reduction techniques, leading to time complexity $O(\log|V|)$. In theory, with sufficient parallel cores, the overall GPU time complexity can approach $O(\log|V|)$. More realistically, with P parallel processing units, the wall-clock time scales as $O\left(\frac{k|V||E|}{P}\right)$.

4.2 COP Instances

We propose novel cost functions for several COPs. Hypergraph partitioning serves as building blocks in numerous applications, including image segmentation Gomes et al. (2017); Chai et al. (2024), storage sharding in distributed databases Kabiljo et al. (2017); Yang et al. (2018), and simulations of distributed quantum circuits Andres-Martinez and Heunen (2019); Gray and Kourtis (2021). Hypergraph coloring has numerous applications in areas such as image segmentation Tuan et al. (2021); Gómez et al. (2007), task scheduling Stollenwerk et al. (2020); Darwish et al. (2021), and resource allocation Wu et al. (2020); Huang et al. (2023); Wang et al. (2020); Mukherjee (2022).

Hypergraph Partitioning. Hypergraph partitioning seeks to partition the node set of a hypergraph into multiple similarly-sized disjoint blocks while reducing the number of hyperedges that span different blocks. Given a hypergraph $G = (V, E)$, the hypergraph partitioning problems can be

[1]Refer to NumPy package for Python.

expressed as:

$$O_{\mathrm{HP}} = \alpha \sum \sum_{e_j \in E} \frac{1}{|e_j|} (\sum_{i=1}^{|e_j|} \mathbf{x}_{ji} - |e_j| \mathbf{x}_{j1} \odot \cdots \odot \mathbf{x}_{j|e_j|}) + \beta \sum_{k=1}^{K} (|P_k| - \frac{|V|}{K})^2 \tag{6}$$

where $\mathbf{x}_{ji}$ are k-dimensional one-hot encoded decision vectors of vertices $v_{ji} \in e_j$. K is the predefined number of partitioned blocks, and $|P_k|$ denotes the number of vertices in partitioned block P_k. α and β are hyperparameters. The one-hot encoded PUBO term $\sum \sum_{e_j \in E} \frac{1}{|e_j|} (\sum_{i=1}^{|e_j|} \mathbf{x}_{ji} - |e_j| \mathbf{x}_{j1} \odot \cdots \odot \mathbf{x}_{j|e_j|})$ represents the number of cut edges. $\sum_{k=1}^{K} (|P_k| - \frac{|V|}{K})^2$ ensures the balancedness.

Hypergraph Max-k-cut. The hypergraph max-k-cut problem involves partitioning the node set of a hypergraph into k disjoint blocks in such a way that the number of hyperedges between vertices in different blocks is maximized. Given a hypergraph $G = (V, E)$, the max-k-cut problem on hypergraphs can be formulated as the one-hot encoded PUBO form:

$$O_{\mathrm{HMkC}} = -\sum \sum_{e_j \in E} \frac{1}{|e_j|} (\sum_{i=1}^{|e_j|} \mathbf{x}_{ji} - |e_j| \mathbf{x}_{j1} \odot \cdots \odot \mathbf{x}_{j|e_j|}) \tag{7}$$

where $\mathbf{x}_{ji}$ are k-dimensional one-hot encoded decision vectors of vertices $v_{ji} \in e_j$.

Hypergraph coloring refers to the assignment of minimal number of colors to vertices such that no hyperedge is monochromatic.

Hypergraph Proper Coloring. Given a hypergraph $G = (V, E)$, the hypergraph proper coloring problem (no hyperedge is monochromatic) is to optimize the cost function:

$$O_{\mathrm{HC}} = \sum_{k=1}^{K_{\max}} y_k + \lambda_1 \sum \sum_{\substack{(v_{j1}, \cdots, v_{j|e|}) \in e \\ e \in E}} \mathbf{x}_{j1} \odot \cdots \odot \mathbf{x}_{j|e|} + \lambda_2 \sum_{i=1}^{|V|} \sum_{k=1}^{K_{\max}} x_{ik}(1 - y_k) \tag{8}$$

where λ_1,$\lambda_2 > 0$ are penalty parameters, $\mathbf{x}_i = (x_{i1}, x_{i2}, \cdots, x_{iK_{\max}})$, $\mathbf{x}_i \in \{0,1\}^{K_{\max}}$ are one-hot vectors that represent the color assignments, variables $y_k \in \{0,1\}$ denote whether the k-th color is used, and $K_{\max}$ is the predefined maximum value of chromatic number. We minimize $\sum_{k=1}^{K_{\max}} y_k$ to optimize the chromatic number. Meanwhile, two types of constraints must be satisfied: **(I)** $\sum_{\substack{(v_{j1}, \cdots, v_{j|e|}) \in e \\ e \in E}} \mathbf{x}_{j1} \odot \cdots \odot \mathbf{x}_{j|e|}$ ensures vertices within the same hyperedge must be assigned more than two colors; **(II)** $\sum_{i=1}^{|V|} \sum_{k=1}^{K_{\max}} x_{ik}(1 - y_k)$ ensures that $x_{ik} \leq y_k$, and thus the k-th color should not be used by any vertex if $y_k = 0$. During training, $\lambda_1 \sum \sum_{\substack{(v_{j1}, \cdots, v_{j|e|}) \in e \\ e \in E}} \mathbf{x}_{j1} \odot \cdots \odot \mathbf{x}_{j|e|}$ can be implemented through the GPU-accelerated algorithm (Eq. 5).

4.3 The Training Pipeline

Our one-hot encoded PUBO formalism is compatible with existing unsupervised NCO frameworks, including PI-GNN and FEM. HyperNCO builds upon these frameworks. To clarify our implementation, we summarize their differences: **(I)** PI-GNN employs GNNs to capture structural features, offering greater representational capacity. In contrast, FEM forgoes computationally expensive neural network layers. Instead, it directly parameterizes the decision variables via the softmax function, enabling training multiple model replicas in parallel to escape local optima; **(II)** PI-GNN lacks an annealing mechanism. Although the CRA algorithm Ichikawa (2024) addressed this limitation, it remains restricted to binary problems.

Model Architecture. Inspired by these frameworks, HyperNCO employs an unsupervised learning architecture as depicted in Fig. 2. To generate decision variables $X = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]^\top$, it adopts optional neural network layers (e.g., HGNN+ Gao et al. (2022) or simple fully connected (FC) layers) incorporating a softmax activation. Notably, as FEM points out, neural network layers consume significant GPU resources and incur computational overhead. Nonetheless, we find that

architectures without these layers fail to scale effectively to large-cardinality hyperedges in problems like hypergraph partitioning and coloring. Therefore, to balance efficiency, we replace them with randomly initialized tensors when dealing with hyperedges of low cardinality. The softmax function then produces k-dimensional vectors as the relaxed representation of $X = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]^\top$.

Training to Optimize the Cost Function. As formalized in Eq. 5, the one-hot encoded PUBO cost function can be computed via efficient GPU-accelerated matrix operations involving the relaxed variables X and the incidence matrix H. HyperNCO minimizes this cost function, driving $X = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n]^\top$ to converge toward optimized solutions. It can leverage deep learning tools (e.g., automatic differentiation in PyTorch or TensorFlow) to optimize the relaxed loss functions via gradient optimizers in an unsupervised manner. Following FEM, HyperNCO employs a batch of m independent replicas, each initialized differently and updated independently, to enhance escape from local optima.

Gini Coefficient-based Annealing. Neural solvers such as PI-GNN face two issues: **(I)** Neural networks are prone to getting stuck in local optima; **(II)** The differentiable relaxation of discrete decision variables leads to soft solutions of continuous values.

To address these challenges, CRA-PI-GNN and FEM employ annealing strategies for binary COPs by incorporating penalty terms into their objective functions. For instance, FEM adopts Shannon entropy, $\gamma \sum_{i=1}^{n} -x_i \log x_i$, as a penalty term for Sigmoid relaxation, which imposes no penalty when x_i is 0 or 1 and yields larger penalties when x_i deviates from 0 and 1. For Softmax relaxation of one-hot encoded decision variables x (where $\sum_{j=1}^{k} x_{ij} = 1$), the Shannon entropy naturally generalizes to $\gamma \sum_{i=1}^{n} \sum_{j=1}^{k} -x_{ij} \log x_{ij}$. However, the gradient of the Shannon entropy function approaches $\pm\infty$ at 0 and 1, which introduces instability during model training. To overcome this limitation, we introduce a new penalty term based on the Gini coefficient, which exhibits a trend of variation similar to Shannon entropy:

$$\phi(\mathbf{x}) = \gamma \sum_{i=1}^{n} (1 - \sum_{j=1}^{k} x_{ij}^2) \tag{9}$$

The Gini coefficient-based penalty term $\phi(\mathbf{x})$ attains its maximum when the variables are uniformly distributed, and minimizes to 0 in a one-hot (discrete) state. Crucially, it is differentiable everywhere.

The annealing strategy enhances the learning process by gradually annealing the parameter γ: **(I)** Initially, γ is set to a negative value ($\gamma < 0$) to smooth the model, thereby facilitating broad search of the solution space and avoiding local optima; **(II)** Subsequently, the value of γ is gradually increased (until $\gamma > 0$) to enforce the discreteness of solutions.

5 EXPERIMENTS

Datasets and Baseline Methods. We evaluate the performance of HyperNCO on publicly available datasets (refer to Appendix. A), including both real-world graphs and hypergraphs. For synthetic datasets, we use DHG library Gao et al. (2022); Feng et al. (2019) to generate hypergraphs. Baseline methods include exact solvers such as Gurobi[2], SCIP Maher et al. (2016) (Please refer to Appendix D for the integer programming formulations of COPs); Neural solvers such as PI-GNN; Problem-specific heuristics such as hMETIS Karypis et al. (1997) and KaHyPar Gottesbüren et al. (2024) (graph/hypergraph partitioning).

Implementation. Experiments were conducted on a workstation equipped with an Intel Core i9-12900K CPU, NVIDIA GeForce RTX 3090 GPUs with 24 GB of memory. For comparison, we include both variants of HyperNCO models based on hypergraph neural networks (HyperGNNs) and an FEM-inspired architecture. For the testing of FEM-inspired HyperNCO model that omits neural network layers, we created 10 replicas to escape local optima.

[2]https://www.gurobi.com/

Table 1: Experimental results on hypergraph partitioning. We compare the performance of a variety of methods including HyperNCO with GAT layers that is denoted as Ours-NN, Ours-FEM that omits neural network layers, PI-GNN, hMETIS, KaHyPar (problem-specific multi-level heuristics), Gurobi, and SCIP (exact solvers for integer programming). Experiments are conducted on 5 graph datasets (BAT, EAT, UAT, DBLP, CiteSeer) and 5 hypergraph datasets (Primary, High, Cora, PubMed, and Cooking200). Three metrics—C (B_1, B_2) (e.g., 192 (2.3%,1.5)) are evaluated as shown in the table: the number of cut edges C, and two balancedness measures B_1 and B_2, where smaller values of B_1 and B_2 indicate better balancedness. These metrics are defined as $B_1 = \frac{\max(S)}{\text{mean}(S)} - 1$ and $B_2 = \sqrt{\frac{1}{k}\sum_{i=1}^{k}\left(|P_i| - \text{mean}(S)\right)^2}$, with $S = \{|P_1|, |P_2|, ..., |P_k|\}$ being the set of partition block sizes. Best results achieved by HyperNCO are highlighted in bold, while the best results among all baseline methods are underlined. For more details please refer to Tab. 5 and Tab. 6 in Appendix B.3 .

Dataset	**Method**	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
BAT	hMETIS	192 (2.3%,1.5)	368 (5.3%,2.1)	465 (6.9%,2.3)	528 (11%,1.9)	593 (9.9%,1.3)
	PI-GNN	192 (2.3%,1.5)	362 (5.3%,1.7)	456 (6.9%,1.3)	526 (6.9%,1.0)	569 (9.9%,1.1)
	SCIP	198 (0%,0)	369 (0%,0)	491 (0%,0)	556 (0%,0)	589 (0%,0)
	Gurobi	198 (0%,0)	361 (0%,0)	461 (0%,0)	522 (0%,0)	583 (0%,0)
	Ours-NN	**191** (2.3%,1.5)	**346** (5.3%,1.7)	**444** (6.9%,1.3)	514 (6.9%,1.2)	567 (9.9%,1.1)
	Ours-FEM	**191** (2.3%,1.5)	**346** (5.3%,1.7)	**444** (6.9%,1.3)	**509** (6.9%,1.2)	**558** (9.9%,1.1)
EAT	hMETIS	946 (2.3%,4.5)	2,034 (3.8%,3.6)	2,565 (5.3%,5.3)	2,990 (6.5%,4.5)	2,983 (46%,22)
	PI-GNN	946 (2.3%,4.5)	2,047 (3.8%,3.6)	2,626 (6.3%,3.7)	2,917 (6.5%,3.5)	2,913 (50%,15)
	SCIP	996 (0%,0)	2,224 (0%,0)	3,065 (0%,0)	3,863 (0%,0)	4,965 (0%,0)
	Gurobi	996 (0%,0)	2,103 (0%,0)	2,604 (0%,0)	3,236 (0%,0)	3,581 (0%,0)
	Ours-NN	**946** (2.3%,4.5)	2,015 (3.8%,3.6)	2,554 (5.3%,3.9)	2,915 (6.5%,3.2)	2,871 (43%,13)
	Ours-FEM	**946** (2.3%,4.5)	**2,014** (3.8%,3.6)	**2,552** (5.3%,3.9)	**2,901** (6.5%,3.2)	**2,870** (43%,13)
UAT	hMETIS	411 (2.0%,12)	1,078 (3.9%,11)	1,783 (4.2%,9.0)	2,568 (5.5%,11)	2,877 (46%,61)
	PI-GNN	410 (2.2%,13)	1,138 (4.1%,12)	1,842 (3.2%,5.5)	2,585 (5.0%,6.0)	2,820 (21%,18)
	SCIP	424 (0%,0)	1,488 (0%,0)	2,643 (0%,0)	8,997 (0%,0)	11,159 (0%,0)
	Gurobi	424 (0%,0)	1,144 (0%,0)	1,841 (0%,0)	2,929 (0%,0)	8,835 (0%,0)
	Ours-NN	403 (2.0%,12)	**1,076** (3.9%,11)	1,749 (3.2%,5.5)	2,499 (4.6%,5.5)	2,719 (20%,18)
	Ours-FEM	**400** (2.0%,12)	**1,076** (3.9%,11)	**1,747** (3.2%,5.5)	**2,480** (4.6%,5.5)	**2,683** (20%,18)
DBLP	hMETIS	84 (0%,0)	136 (0%,0)	166 (0%,0)	186 (0%,0)	223 (0%,0)
	PI-GNN	79 (0%,0)	128 (0%,0)	382 (0%,0)	432 (0.2%,0.7)	399 (8.6%,19)
	SCIP	38 (0%,0)	83 (0%,0)	222 (0%,0)	1,248 (0%,0)	515 (0%,0)
	Gurobi	38 (0%,0)	83 (0%,0)	132 (0%,0)	179 (0%,0)	183 (0%,0)
	Ours-NN	65 (0%,0)	113 (0%,0)	156 (0%,0)	168 (0%,0)	213 (0.8%,2.5)
	Ours-FEM	68 (0%,0)	101 (0%,0)	148 (0%,0)	**164** (0%,0)	207 (0%,0)
CiteSeer	hMETIS	27 (0%,0)	70 (0%,0)	101 (0.2%,0.8)	145 (0%,0)	152 (0%,0)
	PI-GNN	40 (0%,0)	62 (0%,0)	422 (0.2%,0.8)	440 (0.3%,1.7)	536 (0%,0)
	SCIP	25 (0%,0)	27 (0%,0)	49 (0%,0)	140 (0%,0)	1258 (0%,0)
	Gurobi	25 (0%,0)	27 (0%,0)	49 (0%,0)	66 (0%,0)	86 (0%,0)
	Ours-NN	39 (0%,0)	62 (0%,0)	93 (0%,0)	118 (0%,0)	142 (0.3%,0.8)
	Ours-FEM	39 (0%,0)	64 (0%,0)	86 (0%,0)	106 (0%,0)	139 (0.3%,0.8)
Primary	KaHyPar	2,582 (0.8%,1)	4,070 (0%,0)	5,072 (0%,0)	5,190 (0%,0)	6,576 (0%,0)
	SCIP	3,614 (0%,0)	7,792 (0%,0)	8,758 (0%,0)	11,323 (0%,0)	11,420 (0%,0)
	Gurobi	2,594 (0%,0)	4,367 (0%,0)	5,736 (0%,0)	9,581 (0%,0)	8,280 (0%,0)
	Ours-NN	**2,582** (0.8%,1)	4,070 (0%,0)	**5,072** (0%,0)	5,181 (0%,0)	6,484 (0%,0)
	Ours-FEM	**2,582** (0.8%,1)	**4,064** (0%,0)	**5,072** (0%,0)	**5,167** (0%,0)	**6,209** (0%,0)
High	KaHyPar	793 (1.0%,2.5)	828 (1.0%,0.8)	1,530 (0%,0)	1,916 (0%,0)	2,151 (0%,0)
	SCIP	905 (0%,0)	1,520 (0%,0)	6,127 (0%,0)	6,389 (0%,0)	6,558 (0%,0)
	Gurobi	851 (0%,0)	931 (0%,0)	1,791 (0%,0)	2,171 (0%,0)	2,587 (0%,0)
	Ours-NN	784 (1.5%,2.5)	812 (2.8%,2.2)	1,523 (0%,0)	1,903 (0%,0)	2,149 (0%,0)
	Ours-FEM	**772** (1.5%,2.5)	**806** (2.8%,2.2)	**1,424** (0%,0)	**1,897** (0%,0)	**2,123** (0%,0)
Cora	KaHyPar	123 (0%,0)	172 (0.4%,1.7)	201 (0%,0)	237 (0.8%,2.6)	270 (0.6%,2.2)
	SCIP	116 (0%,0)	151 (0%,0)	210 (0%,0)	232 (0%,0)	324 (0%,0)
	Gurobi	116 (0%,0)	151 (0%,0)	179 (0%,0)	211 (0%,0)	227 (0%,0)
	Ours-NN	122 (0%,0)	169 (0%,0)	200 (0%,0)	234 (0%,0)	261 (0%,0)
	Ours-FEM	126 (0%,0)	199 (0%,0)	265 (0%,0)	326 (0%,0)	391 (0%,0)
PubMed	KaHyPar	814 (0%,0)	1,253 (0%,0)	1,476 (0.6%,9.3)	1,864 (0.9%,12)	2,083 (1.1%, 15)
	SCIP	911 (0%,0)	5,463 (0%,0)	6,388 (0%,0)	5,887 (0%,0)	-
	Gurobi	809 (0%,0)	2,923 (0%,0)	4,940 (0%,0)	4,139 (0%,0)	4,700 (0%,0)
	Ours-NN	844 (0%,0)	**1,157** (0%,0)	**1,455** (0.3%,3.1)	**1,855** (0.8%,9.5)	**2,048** (0%,0)
	Ours-FEM	818 (0%,0)	1,202 (0%,0)	1,640 (0.3%,3.1)	1,962 (0.8%,9.5)	2,418 (0%,0)
Cooking 200	KaHyPar	1,093 (0%,0)	1,304 (0%,0)	1,454 (0%,0)	1,507 (0%,1.5)	1,560 (0.7%,7.9)
	SCIP	1,073 (0%,0)	1,978 (0%,0)	2,395 (0%,0)	2,022 (0%,0)	2,036 (0%,0)
	Gurobi	1,002 (0%,0)	1,375 (0%,0)	2,237 (0%,0)	2,203 (0%,0)	2,261 (0%,0)
	Ours-NN	1,283 (0%,0)	1,305 (0%,0)	**1,406** (0%,0)	**1,439** (0%,0.8)	**1,511** (0%,0)
	Ours-FEM	1,232 (0%,0)	1,452 (0%,0)	1,483 (0%,0)	1,506 (0%,0.8)	1,524 (0%,0)

5.1 HYPERGRAPH PARTITIONING

The experimental results are summarized in Tab. 1. Evaluation metrics include the number of cuts and balancedness. The ratio $\frac{\alpha}{\beta}$ in Eq. 6 was fixed to a constant value, and the learning rate η was set to 1^{-4}. The runtime of Gurobi and SCIP was limited to 1 hour.

For graph datasets, we tested HyperNCO integrated with GNNs (denoted as HyperNCO-NN in Tab. 1) such as GCN, GraphSage, and GAT Veličković et al. (2018). Among these, GATs with 2 or 3 layers yielded better performance on graphs. However, the FEM-inspired architecture which omits neural network layers (denoted as HyperNCO-FEM) generally outperformed GAT-based model architecure.

For hypergraph datasets, HyperNCO achieved its best performance when employing $4\sim8$ layers of HGNN+ Gao et al. (2022) incorporating FC layers. More concretely, the FEM-inspired model performed well on graph datasets (BAT, EAT, UAT, DBLP, CiteSeer) and hypergraphs with low hyperedge cardinality (Primary, High), where omitting neural network layers is feasible. In contrast, for hypergraphs with large hyperedge cardinality (Cora, PubMed, Cooking200), HyperNCO required HyperGNNs to effectively capture the complex, high-order interactions among nodes within each hyperedge.

Moreover, almost all best solutions were obtained by HyperNCO or Gurobi. As an exact solver, Gurobi outperformed SCIP. Compared to HyperNCO, Gurobi performs better on small-scale datasets. However, as the data scale and the value of k increased, the solution space grew exponentially, making it challenging for Gurobi to approximate the optimal solution. Additionally, when dealing with nonlinear problems, Gurobi and SCIP need to introduce numerous auxiliary variables to linearize the original problem. The number of auxiliary variables increases exponentially with the cardinality of hyperedges. As a result, they often fail to find the optimal solution within limited time when solving large-scale nonlinear hypergraph COPs.

In summary, **(I)** On small datasets with low degrees (DBLP, CiteSeer, Cora), we may consider using Gurobi to search for optimal solutions; **(II)** For graphs and hypergraphs with small-cardinality hyperedges (BAT, EAT, UAT, Primary, High), we may choose (FEM-inspired) HyperNCO that omits neural network layers to obtain high-quality approximate solutions; **(III)** For hypergraphs with large-cardinality hyperedges (PubMed, Cooking200), HyperGNN-based HyperNCO architecture achieves better performance; **(IV)** However, HyperGNN-based HyperNCO remains constrained by GPU VRAM limitations. Thus, for even larger hypergraphs (e.g., exceeding 100,000 vertices), KaHyPar should be considered as a more scalable alternative as it is a shared-memory framework designed for high-performance computing and capable of partitioning very large hypergraphs.

5.2 HYPERGRAPH COLORING

The performance of HyperNCO on graph coloring and hypergraph proper coloring is evaluated on real-world datasets, as shown in Tab. 2.

For the random tensor $Z \in \mathbb{R}^{|V|\times K_{\max}}$ (which can be generated by neural networks), we apply the softmax function to obtain the decision variables $X = \mathrm{softmax}(Z)$. Concurrently, to generate the color decision variables $\mathbf{y} = (y_1, \cdots, y_{K_{max}})$ for Eq. 8, we use max pooling followed by a sigmoid activation: $\mathbf{y} = \mathrm{sigmoid}\big(\mathrm{maxpooling}(Z^\top)\big)$. The penalty factor is updated adaptively as $\lambda_{t+1} = \lambda_t + \eta' \cdot \mathrm{cons}$, where λ_t and λ_{t+1} represent the penalty factor for the current epoch and the next epoch. cons is the average constraint violation in the current epoch, and η' is the learning rate, controlling the step size for adjustment of each update to λ. The learning rate η is set to 1^{-4}.

For graph datasets (BAT, EAT, UAT, DBLP, CiteSeer), we employ the HyperNCO variant that omits neural network layers. For hypergraph datasets (Primary, High, Cora, PubMed, Cooking200), we use a 2-layer GraphSAGE Hamilton et al. (2017) as the base model for HyperNCO. GraphSAGE is chosen because it demonstrated superior performance compared to other GNN and HyperGNN models across all datasets in our tests. To adapt GraphSAGE for vertex embedding tasks in hypergraph proper coloring, the hypergraphs are transformed into graphs through clique expansion.

Tab. 2 presents the chromatic numbers found and the time (s) taken to find the first valid solution. While Gurobi outperforms SCIP on graph datasets, SCIP shows better performance on hypergraphs. Notably, Gurobi failed to produce a solution for the PubMed and Cooking200 datasets due to exceeding the available memory (RAM). In summary, a simple degree-based greedy algorithm

Table 2: Experimental results on graph coloring and hypergraph proper coloring. Evaluation metrics include chromatic numbers and the runtime (s) (e.g., 16 (3)).

Method	BAT	EAT	UAT	DBLP	CiteSeer	Primary	High	Cora	PubMed	Cooking
Gurobi	16 (3)	24 (605)	57 (614)	6 (21)	6 (298)	30 (605)	23 (276)	7 (604)	-	-
SCIP	18 (2)	29 (9)	64 (25)	8 (16)	8 (22)	29 (12)	20 (424)	4 (13)	5 (15)	3 (36)
	16 (16)	25 (12)	60 (32)	7 (37)	7 (66)	27 (557)	20 (424)	3 (30)	4 (27)	2 (851)
Greedy	16 (1e-4)	24 (6e-4)	58 (1e-3)	6 (9e-4)	6 (1e-3)	29 (0.25)	21 (0.14)	5 (0.03)	10 (17)	6 (0.66)
Ours	17 (0.3)	25 (0.6)	59 (0.9)	7 (4)	6 (20)	31 (20)	**20** (6)	4 (8)	5 (7)	3 (6)

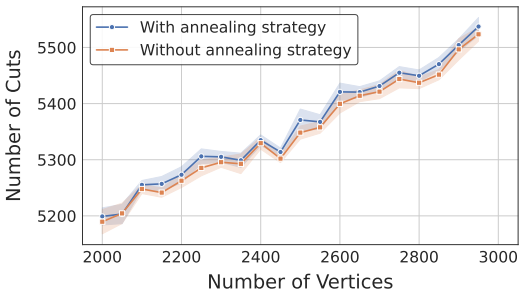


Figure 4: Solution quality of the max-cut problem with or without the annealing strategy across various graph sizes.

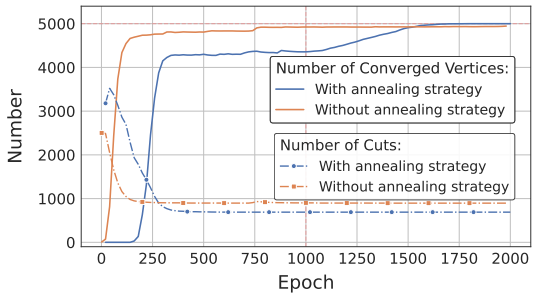


Figure 5: Quality and discreteness of solutions of the graph partitioning problem with or without the annealing strategy.

achieves the best performance in both speed and solution quality (chromatic number) for graph datasets. For hypergraph datasets, HyperNCO can find valid colorings faster but struggles to match the optimal chromatic numbers found by SCIP. It is important to note, however, that SCIP's strong performance is contingent upon the specific objective function formulation introduced in Sec. 4.2.

5.3 Ablation Study

Annealing Strategy. Here, we validate the effectiveness of Gini coefficient-based annealing strategy. As shown in Fig. 4, the numbers of cuts across various scales of graphs for the max-cut problem have been evaluated. HyperNCO is tested on graphs with $2,000 \sim 3,000$ vertices. For each scale, we generate 50 random graphs (where $|V| = |E|$) and report the average number of cuts. The experimental results demonstrate that the annealing strategy effectively prevents the model from becoming trapped in local optima.

The experimental results for graph partitioning (where $k = 4$) are illustrated in Fig. 5. The experiments were conducted on a graph of 5,000 vertices (where $|V| = |E|$). The penalty factor γ was initialized to -0.25, reached 0 after 1000 epochs and continued to increase thereafter with increasing training epochs. As illustrated in Fig. 5, a negative γ value enables the model to escape local optima, thereby discovering higher-quality solutions. Conversely, a positive γ guarantees convergence of all vertex states, ultimately driving all decision variables to discrete values.

6 Conclusions

In this work, we propose HyperNCO, a unified NCO framework for solving hypergraph COPs.

Algorithm Design. We propose one-hot encoded PUBO formalism for modeling various hypergraph COPs. On this basis, we propose a GPU-accelerated and Gini coefficient-based annealing enhanced training pipeline to solve COPs in an unsupervised and end-to-end manner.

Extensive Experiments & Empirical Findings. We propose novel cost functions for hypergraph partitioning/max-k-cut/coloring problems. Although exact solvers like Gurobi can exhaustively search for globally optimal solutions for small-scale datasets, experimental results have demonstrated the performance of HyperNCO on larger problems. Neural solvers hold promise in tackling real-world CO challenges.

AI USE STATEMENT

We have not used generative AI tools for any tasks requiring disclosure. The use of generative AI tools for tasks requiring recommended disclosure is also not applicable to this work. We take full responsibility for the final content of this work, including all text, claims, and artifacts.

REPRODUCIBILITY STATEMENT

In this paper, we present a unified framework for neural combinatorial optimization. To ensure that our work can be easily reproduced and built upon by the research community, we have taken several key steps. First, the source code implementing our framework is available as part of the supplementary materials. The code includes all scripts necessary for training and evaluation. Experimental settings and hyperparameters are available in the supplementary materials. We use publicly available datasets. In terms of theoretical contributions, we include clear explanations of assumptions and complete proofs for all claims made in the paper. Formal derivations and justifications can be found in the paper for further verification. Finally, we also provide details about the hardware and software environment used in our experiments. By providing detailed descriptions, open-source code, and clear theoretical justifications, we aim to make our work on neural combinatorial optimization fully reproducible and accessible to the broader research community.

REFERENCES

Pablo Andres-Martinez and Chris Heunen. Automated distribution of quantum circuits via hypergraph partitioning. *Physical Review A*, 100(3):032308, 2019.

Quentin Cappart, Didier Chételat, Elias B Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

Shurong Chai, Rahul K Jain, Shaocong Mo, Jiaqing Liu, Yulin Yang, Yinhao Li, Tomoko Tateyama, Lanfen Lin, and Yen-Wei Chen. A novel adaptive hypergraph neural network for enhancing medical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 23–33. Springer, 2024.

Ming Chen, Jie Chun, Shang Xiang, Luona Wei, Yonghao Du, Qian Wan, Yuning Chen, and Yingwu Chen. Learning to solve quadratic unconstrained binary optimization in a classification way. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:114478–114509, 2024.

Lamis R Darwish, Mohamed T El-Wakad, and Mahmoud M Farag. Towards sustainable industry 4.0: A green real-time iiot multitask scheduling architecture for distributed 3d printing services. *Journal of Manufacturing Systems*, 61:196–209, 2021.

Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 3558–3565, 2019.

Yue Gao, Yifan Feng, Shuyi Ji, and Rongrong Ji. Hgnn+: General hypergraph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3181–3199, 2022.

Fred Glover, Gary Kochenberger, Rick Hennig, and Yu Du. Quantum bridge analytics i: a tutorial on formulating and using qubo models. *Annals of Operations Research*, 314(1):141–183, 2022.

Vitor Gomes, Pablo Barcellos, and Jacob Scharcanski. Stochastic shadow detection using a hypergraph partitioning approach. *Pattern Recognition*, 63:30–44, 2017.

D Gómez, Javier Montero, Javier Yáñez, and C Poidomani. A graph coloring approach for image segmentation. *Omega*, 35(2):173–183, 2007.

Lars Gottesbüren, Tobias Heuer, Nikolai Maas, Peter Sanders, and Sebastian Schlag. Scalable high-quality hypergraph partitioning. *ACM Transactions on Algorithms*, 20(1):1–54, 2024.

Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, 2021.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.

Jie Huang, Shilong Zhang, Fan Yang, Tao Yu, LV Narasimha Prasad, Manisha Guduri, and Keping Yu. Hypergraph-based interference avoidance resource management in customer-centric communication for intelligent cyber-physical transportation systems. *IEEE Transactions on Consumer Electronics*, 2023.

Yuma Ichikawa. Controlling continuous relaxation for combinatorial optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:47189–47216, 2024.

Igor Kabiljo, Brian Karrer, Mayank Pundir, Sergey Pupyrev, and Alon Shalita. Social hash partitioner: a scalable distributed hypergraph partitioner. *Proceedings of the VLDB Endowment*, 10(11): 1418–1429, 2017.

George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: Application in vlsi domain. In *Proceedings of the 34th annual Design Automation Conference (DAC)*, pages 526–529, 1997.

Qian Li, Tian Ding, Linxin Yang, Minghui Ouyang, Qingjiang Shi, and Ruoyu Sun. On the power of small-size graph neural networks for linear programming. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

Stephen Maher, Matthias Miltenberger, João Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano. PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *Mathematical Software – ICMS 2016*, pages 301–307. Springer International Publishing, 2016. doi: 10.1007/978-3-319-42432-3_37.

Sayan Mukherjee. A grover search-based algorithm for the list coloring problem. *IEEE Transactions on Quantum Engineering*, 3:1–8, 2022.

Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, and Azalia Mirhoseini. A deep learning framework for graph partitioning. *International Conference on Learning Representations (ICLR)*, 2019.

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

Martin JA Schuetz, J Kyle Brubaker, Zhihuai Zhu, and Helmut G Katzgraber. Graph coloring with physics-inspired graph neural networks. *Physical Review Research*, 4(4):043131, 2022b.

Zi-Song Shen, Feng Pan, Yao Wang, Yi-Ding Men, Wen-Biao Xu, Man-Hong Yung, and Pan Zhang. Free-energy machine for combinatorial optimization. *Nature Computational Science*, pages 1–11, 2025.

Tobias Stollenwerk, Stuart Hadfield, and Zhihui Wang. Toward quantum gate-model heuristics for real-world planning problems. *IEEE Transactions on Quantum Engineering*, 1:1–16, 2020.

Ye Tian, Luchen Wang, Shangshang Yang, Jinliang Ding, Yaochu Jin, and Xingyi Zhang. Neural network-based dimensionality reduction for large-scale binary optimization with millions of variables. *IEEE Transactions on Evolutionary Computation*, 2024.

Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any csp: graph neural networks as fast global search heuristics for constraint satisfaction. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 4280–4288, 2023.

Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph clustering with graph neural networks. *Journal of Machine Learning Research*, 24(127):1–21, 2023.

Tran Anh Tuan, Nguyen Tuan Khoa, Tran Minh Quan, and Won-Ki Jeong. Colorrl: reinforced coloring for end-to-end instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16727–16736, 2021.

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

Bowen Wang, Yanjing Sun, Zhi Sun, Long D Nguyen, and Trung Q Duong. Uav-assisted emergency communications in social iot: A dynamic hypergraph coloring approach. *IEEE Internet of Things Journal*, 7(8):7663–7677, 2020.

Kai-Ju Wu, Y-W Peter Hong, and Jang-Ping Sheu. Coloring-based channel allocation for multiple coexisting wireless body area networks: A game-theoretic approach. *IEEE Transactions on Mobile Computing*, 21(1):63–75, 2020.

Wenyin Yang, Guojun Wang, Kim-Kwang Raymond Choo, and Shuhong Chen. Hepart: A balanced hypergraph partitioning algorithm for big data applications. *Future Generation Computer Systems*, 83:250–268, 2018.

Huigen Ye, Hua Xu, Hongyan Wang, Chengming Wang, and Yu Jiang. Gnn&gbdt-guided fast optimizing framework for large-scale integer programming. In *International conference on machine learning (ICML)*, pages 39864–39878. PMLR, 2023.

A Datasets and Baseline Methods

The performance of HyperNCO is evaluated on publicly available datasets shown in Tab. 3, including both real-world graphs and hypergraphs.

Table 3: Summary statistics of ten real-world graph and hypergraph datasets. BAT, EAT, UAT, DBLP, and CiteSeer are graphs. Primary, High, Cora, Pubmed, and Cooking200 are hypergraphs. The number of vertices $|V|$, the number of edges or hyperedges $|E|$, and the size of the hypergraph $\sum_{e\in E}|e|$, have been depicted in the table. Moreover, vertex degrees (denoted as Degree) and hyperedge cardinalities (denoted as Cardinality) are presented in the table.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\sum_{e\in E}\lvert e\rvert$	**Degree**	**Cardinality**
BAT	131	1,003		15.31	2
EAT	399	5,993		30.04	2
UAT	1,190	13,599		22.86	2
DBLP	2,591	3,528		2.72	2
CiteSeer	3,279	4,552		2.78	2
Primary	242	12,704	30,729	126.82	2.42
High	327	7,818	18,192	55.63	2.33
Cora	1,330	1,413	4,370	3.28	3.09
Pubmed	3,824	7,523	33,687	8.81	4.48
Cooking200	7,403	2,750	54,970	7.43	19.99

Baseline methods are summarized as below:

Gurobi. A commercial optimization solver for complex mathematical programming problems, including linear, quadratic, and mixed-integer optimization.

SCIP. An open-source solver for mixed-integer programming, accessed via its Python interface PySCIPOpt Maher et al. (2016).

hMETIS. A widely-used multi-level hypergraph partitioner[3].

KaHyPar. A multi-level algorithm Gottesbüren et al. (2024) that typically outperforms hMETIS, especially under tight imbalance constraints.

B Additional Experimental Results

B.1 Hypergraph Max-k-cut

The evaluation results for the hypergraph max-k-cut problem are presented in Tab. 4, measured by the number of cut edges. The learning rate η was set to 1^{-4}, and the runtime of Gurobi and SCIP was limited to one hour. We employed two variants of HyperNCO models, with and without neural network layers. The HyperNCO-NN variant uses a 2-layer GAT or a 4-layer HGNN+ as its base model.

As shown in Tab. 4, Gurobi outperforms SCIP. HyperNCO achieves better results than Gurobi on four larger-scale datasets, including EAT, UAT, Primary, and High. Conversely, Gurobi outperforms HyperNCO on small-scale datasets, including DBLP and CiteSeer. For Cora, PubMed, and Cooking200 at $k = 2$, both Gurobi and HyperNCO already find the maximum numbers of cuts, therefore, increasing k yields no further improvement in the results.

B.2 Additional Ablation Study

GPU-accelerated Algorithm. We conduct experiments to verify that GPU-acceleration algorithm (Eq. 5) is crucial for addressing hypergraph COPs. To elucidate that the GPU-acceleration algorithm significantly speeds up the training processes, we compare the runtime of the GPU-accelerated loss (Eq. 5) with that of a baseline loss computed via direct hyperedge traversal (Eq. 4). HyperNCO is

[3]https:karypis.github.ioglarossoftwaremetisoverview.html#hmetishypergraphcircuitpartitioning

Table 4: Experimental results on hypergraph max-k-cut.

Method	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	**Method**	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$
			BAT						Primary		
SCIP	655	829	899	929	952	SCIP	7,963	9,811	10,816	11,372	11,667
Gurobi	655	829	902	942	965	Gurobi	8,484	10,293	10,983	11,446	11,752
Ours-NN	650	825	897	933	958	Ours-NN	**8,494**	**10,392**	**11,204**	**11,536**	**11,897**
Ours-FEM	**655**	**829**	**902**	**942**	964	Ours-FEM	7,093	9,306	10,398	10,993	11,399
			EAT						High		
SCIP	3,782	4,475	5,108	5,418	5,587	SCIP	4,943	6,096	6,691	7,054	7,265
Gurobi	3,997	4,853	5,277	5,554	5,640	Gurobi	5,213	6,342	6,842	7,128	7,312
Ours-NN	3,993	4,968	**5,405**	5,592	5,638	Ours-NN	5,046	**6,415**	**6,925**	**7,230**	**7,410**
Ours-FEM	**3,997**	**5,021**	5,403	**5,608**	**5,726**	Ours-FEM	4,544	5,899	6,594	6,977	7,219
			UAT						Cora		
SCIP	7,438	9,825	11,048	11,918	12,164	SCIP	1,400	1,413	1,413	1,413	1,413
Gurobi	8,336	10,281	11,432	12,098	12,407	Gurobi	1,400	1,413	1,413	1,413	1,413
Ours-NN	8,414	10,635	11,611	12,177	12,553	Ours-NN	1,394	**1,413**	**1,413**	**1,413**	**1,413**
Ours-FEM	**8,451**	**10,763**	**11,774**	**12,310**	**12,650**	Ours-FEM	1,096	1,219	1,318	1,352	1,358
			DBLP						PubMed		
SCIP	2,869	3,393	3,505	3,523	3,528	SCIP	7,205	7,520	7,523	7,523	7,523
Gurobi	2,869	3,393	3,505	3,523	3,528	Gurobi	7,251	7,520	7,523	7,523	7,523
Ours-NN	2,827	3,370	3,493	3,520	3,526	Ours-NN	7,162	7,508	7,522	**7,523**	**7,523**
Ours-FEM	2,829	3,371	3,498	3,521	3,527	Ours-FEM	5,431	6,194	6,795	6,962	7,073
			CiteSeer						Cooking200		
SCIP	3,960	4,478	4,545	4,550	4,552	SCIP	1,002	1,375	2,237	2,203	2,261
Gurobi	3,960	4,478	4,545	4,550	4,552	Gurobi	2,750	2,750	2,750	2,750	2,750
Ours-NN	3,924	4,442	4,530	4,543	4,544	Ours-NN	2,749	2,745	2,749	**2,750**	2,748
Ours-FEM	3,912	4,453	4,536	4,545	4,549	Ours-FEM	1,688	2,084	2,260	2,365	2,413

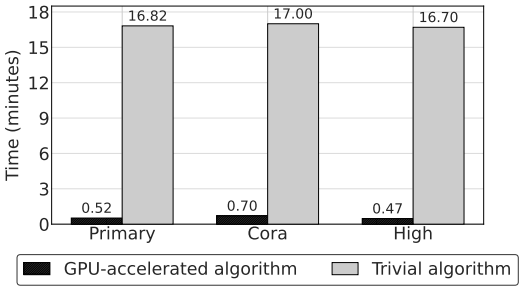


Figure 6: Comparison of time efficiency between the GPU-accelerated algorithm in Eq. 5 and trivial algorithm in Eq. 4.

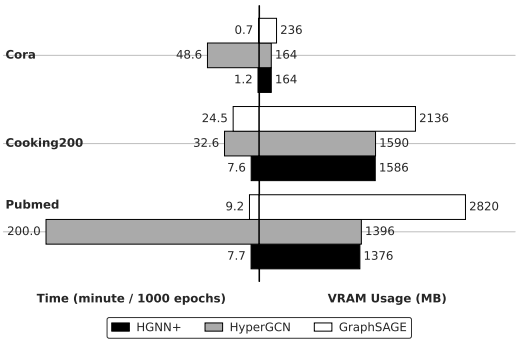


Figure 7: Runtime and VRAM usage of HyperNCO to solve hypergraph partitioning problems when applying HyperGNN models including HGNN+, HyperGCN, and GraphSAGE.

tested on hypergraph partitioning, with each model trained for 1,000 epochs. As shown in Fig. 6, the GPU-accelerated algorithm achieves a speedup of over 20 times, confirming the necessity of implementing the loss functions using matrix operations.

HyperGNN Model Selection. HGNN+ was selected as the base HyperGNN model for HyperNCO. This decision is based on our comparative experiments involving HGNN+, HyperGCN, and UniGCN, where HGNN+ consistently outperformed others in both training speed and VRAM efficiency. To further validate this, we measure the runtime of HGNN+, HyperGCN, and GraphSAGE on three hypergraph datasets (Cora, PubMed, Cooking200) for hypergraph partitioning with $k = 3$. Each model is run for 5,000 epochs. As shown in Fig. 7, HGNN+ is more time-efficient than HyperGCN

and GraphSAGE, while the VRAM usage of HGNN+ is similar to that of HyperGCN, and both of them are superior to GraphSAGE.

B.3 Additional Results

We further compare HyperNCO variants based on FEM, PI-GNN, and PI-GNN+Gini annealing.

PI-GNN was originally designed solely to solve QUBO problems. To compare their performance, we extended it using HGNN+. The difference between Ours-NN and PI-GNN in the table lies in whether the Gini annealing mechanism is incorporated.

Hypergraph Partitioning. In Tab. 1, we have provided experimental results on hypergraph partitioning. To further compare the differences among FEM, PI-GNN, and PI-GNN with the Gini coefficient-based annealing mechanism in terms of VRAM usage and runtime, we present the additional results in Tab. 5. The runtime limit for Gurobi and SCIP was set to 1 hour. Moreover, querying the online API of KaHyPar reveals that it runs extremely fast, as it serves as a tailored heuristic for hypergraph partitioning with effective computational speed optimizations.

From Tab. 5, we observe three key findings that are consistent with the conclusions reported in CRA Ichikawa (2024) and FEM Shen et al. (2025):

First, on graphs or hypergraphs with smaller hyperedge cardinalities, FEM outperforms both PI-GNN and PI-GNN+Gini. This is consistent with FEM's observation that neural network architectures can slow down training speed and sometimes degrade solution quality. However, on certain hypergraphs with large hyperedge sizes, FEM fails to consistently outperform PI-GNN+Gini. This suggests that HyperGNN representations are beneficial for handling high-order relationships.

Second, in our experiments, PI-GNN+Gini generally outperforms PI-GNN, which aligns with CRA's findings.

Third, HyperNCO cannot surpass exact solvers like Gurobi when the dataset size is small, and underperforms tailored heuristics on a few instances. Branch-and-Cut solvers (e.g., Gurobi and SCIP) need to search for exact solutions within an exponentially growing solution space, which degrades their performance on large-scale datasets. Furthermore, for non-linear problems such as PUBO, Branch-and-Cut solvers rely on linearization techniques that introduce a substantial number of auxiliary variables. This further causes Gurobi and SCIP to suffer from severe computational overhead when solving non-linear instances.

Moreover, as mentioned in Tab. 1, for HyperNCO-FEM we execute 10 independent runs (replicas/seeds) to avoid local optima. Several solutions generated by HyperNCO-FEM are detailed in Tab. 6. As indicated in the table, executing runs across multiple random seeds effectively prevent the model from getting trapped in local optima.

By leveraging the synthetic data, we conduct additional experiments by varying the hyperedge cardinality under fixed numbers of vertices ($|V|$), hyperedges ($|E| = 6,000$), and partitions ($k = 3$). Increasing the cardinality effectively changes the density of connections within the hypergraph. Experimental results are shown in Tab. 7. Both HyperNCO variants (FEM and PI-GNN+Gini) consistently outperform the baseline KaHyPar across all density and cardinalities. For PI-GNN+Gini, a two-layer HGNN+ architecture was adopted. On ordinary graph data (cardinality is 2), FEM performs better than PI-GNN+Gini. However, as the cardinality increases (denser hyperedge connectivity), PI-GNN+Gini consistently achieves the best performance.

Hypergraph Max-cut. By leveraging the synthetic data of uniform and non-uniform hypergraphs, we further analyze the performance of HyperNCO across varying hypergraph size and densities. The experimental results are presented in Tab. 8, where we compare the performance of different methods on the hypergraph max-cut problem with hyperedge cardinalities from 2 to 4. Note that the reported Gurobi and SCIP values are the best feasible solutions found within the 10-minute time limit.

On larger instances, the exact solvers do not converge to optimality within this budget, which explains why neural methods can achieve competitive or superior results. For PI-GNN+Gini, a two-layer HGNN+ architecture was adopted. The results indicate that FEM performs better on ordinary graphs, whereas adding HGNN+ neural layers enhances model performance on larger-scale datasets and

Table 5: Comparison of the implementations of HyperNCO based on FEM, PI-GNN, and PI-GNN with the Gini coefficient-based annealing mechanism. We evaluated the runtime (s) and VRAM usage (MB) on hypergraph partitioning.

Dataset	**Model**	$k=2$			$k=3$			$k=4$		
		Time	**VRAM**	**Cut**	**Time**	**VRAM**	**Cut**	**Time**	**VRAM**	**Cut**
BAT	Ours-FEM	3	0.24	**191**	4	0.25	**346**	4	0.24	**444**
	PI-GNN	16	58	192	16	58	362	22	58	456
	Our-NN	13	58	**191**	4	58	**346**	16	58	**444**
	Gurobi	-	-	198	-	-	361	-	-	461
	hMETIS	-	-	192	-	-	368	-	-	465
EAT	Ours-FEM	3	2	**946**	3	2	**2,014**	4	2	**2,552**
	PI-GNN	11	47	**946**	13	47	2,047	12	116	2,626
	Our-NN	9	47	**946**	11	47	2,015	10	116	2,554
	Gurobi	-	-	996	-	-	2,103	-	-	2,604
	hMETIS	-	-	**946**	-	-	2,034	-	-	2,565
UAT	Ours-FEM	4	17	**400**	4	17	**1,076**	15	17	**1,747**
	PI-GNN	32	240	410	51	334	1,138	43	334	1,842
	Our-NN	27	240	403	36	334	**1,076**	36	334	1,749
	Gurobi	-	-	424	-	-	1,144	-	-	1,841
	hMETIS	-	-	411	-	-	1,078	-	-	1,783
DBLP	Ours-FEM	4	78	68	4	78	101	4	78	148
	PI-GNN	52	873	79	114	674	128	134	877	382
	Our-NN	26	873	65	76	674	113	106	877	156
	Gurobi	-	-	**38**	-	-	**83**	-	-	**132**
	hMETIS	-	-	84	-	-	136	-	-	166
CiteSeer	Ours-FEM	4	126	39	4	126	61	4	126	86
	PI-GNN	37	252	40	101	758	62	140	758	422
	Our-NN	31	252	39	111	758	58	100	758	93
	Gurobi	-	-	**25**	-	-	**27**	-	-	**49**
	hMETIS	-	-	27	-	-	70	-	-	101
Primary	Ours-FEM	11	89	**2,582**	15	127	**4,059**	36	167	**5,072**
	PI-GNN	44	1,965	2,594	132	2,029	4,380	191	174	5,080
	Our-NN	45	1,377	**2,582**	111	1,424	4,078	164	183	**5,072**
	Gurobi	-	-	2,594	-	-	4,367	-	-	5,736
	KaHyPar	-	-	**2,582**	-	-	4070	-	-	-
High	Ours-FEM	5	73	**772**	13	105	**806**	8	141	**1,424**
	PI-GNN	81	148	843	91	231	861	327	456	1,689
	Our-NN	70	216	784	81	231	812	433	386	1,523
	Gurobi	-	-	851	-	-	931	-	-	1,791
	KaHyPar	-	-	793	-	-	828	-	-	1,530
Cora	Ours-FEM	4	54	126	10	79	199	12	218	265
	PI-GNN	42	120	131	82	187	169	113	281	200
	Our-NN	32	120	122	82	185	168	56	333	200
	Gurobi	-	-	**116**	-	-	**151**	-	-	**179**
	KaHyPar	-	-	123	-	-	172	-	-	201
PubMed	Ours-FEM	39	830	818	155	1,158	1,189	211	1,688	1,640
	PI-GNN	46	1,106	843	310	1,259	1,177	531	3,870	1,464
	Our-NN	35	1,106	838	182	1,259	**1,139**	412	4,017	**1,455**
	Gurobi	-	-	**809**	-	-	2,923	-	-	4,940
	KaHyPar	-	-	814	-	-	1,253	-	-	1,476
Cooking200	Ours-FEM	111	721	1,232	156	862	1,437	153	1,249	1,483
	PI-GNN	290	2,110	1,283	417	3,540	1,351	2,167	1,571	**1,375**
	Our-NN	290	1,876	1,251	407	3,001	1,305	1,109	1,576	1,406
	Gurobi	-	-	**1,002**	-	-	1,375	-	-	2,237
	KaHyPar	-	-	1,093	-	-	**1,304**	-	-	1,454

Table 6: Experimental results across replicas for FEM in Tab. 1.

Dataset	k	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
Cora	2	128	128	126	126	126	139	126	126	126	126
	3	203	204	233	211	204	199	246	208	209	203
Primary	2	3,067	2,594	2,582	2,582	3,494	2,594	3,149	2,582	2,582	3,712
	3	4,098	4,456	4,204	4,291	4,064	4,456	4,291	4,064	4,291	4,291
High	2	851	861	861	840	922	819	861	840	772	772
	3	806	806	806	806	806	806	806	806	1,547	806
PubMed	2	829	1,069	843	824	828	834	829	818	881	835
	3	1,218	1,276	1,385	1,202	1,279	1,273	1,202	1,694	1,207	1,278
Cooking	2	1,232	1,240	1,290	1,253	1,268	1,277	1,308	1,234	1,235	1,234
200	3	1,452	1,468	1,480	1,476	1,477	1,461	1,467	1,461	1,487	1,468

Table 7: Experimental results on synthetic data for hypergraph partitioning. We compare the implementations of HyperNCO based on FEM and PI-GNN. The cut number (where $k = 3$) is reported in the table below.

$\|V\|$	$\|E\|$	**Cardinality**	**KaHyPar**	**Ours-FEM**	**Our-NN**
200	6,000	2	3,472	**3,438**	3,453
		4	5,523	**5,473**	**5,473**
		6	5,885	5,856	**5,850**
		8	5,958	5,947	**5,943**
300	6,000	2	3,246	**3,220**	3,227
		4	5,472	5,397	**5,382**
		6	5,851	5,812	**5,809**
		8	5,943	5,930	**5,723**
1,000	6,000	2	2,470	2,429	**2,426**
		4	4,986	4,987	**4,963**
		6	5,617	5,609	**5,590**
		8	5,824	5,834	**5,796**

Table 8: Comparison of the implementations of HyperNCO based on FEM, and PI-GNN. Experimental results on hypergraph max-cut is shown in the table. Evaluation metrics include the cut number and runtime (s) (e.g., 3,385 (11)).

Type	**Cardinality**	$\|V\|$	$\|E\|$	**Ours-FEM**	**Our-NN**	**SCIP**	**Gurobi**
Uniform	2	2,000	4,000	**3,385** (11)	3,367 (45)	3,177	3,378
Uniform	2	4,000	8,000	**6,766** (40)	6,757 (94)	5,723	6,718
Uniform	2	6,000	12,000	**10,158** (89)	10,069 (167)	8,534	9,992
Uniform	2	2,000	8,000	6,038 (42)	**6,040** (67)	4,946	5,914
Uniform	2	4,000	16,000	12,052 (162)	**12,067** (164)	9,875	11,859
Uniform	2	6,000	24,000	18,083 (360)	**18,100** (362)	14,787	17,695
Uniform	3	2,000	4,000	3,967 (22)	3,979 (47)	3,723	**3,991**
Uniform	3	4,000	8,000	7,943 (82)	7,940 (94)	7,359	**7,961**
Uniform	3	6,000	12,000	11,915 (182)	11,903 (167)	11,165	**11,949**
Uniform	3	2,000	8,000	7,575 (42)	**7,601** (70)	6,768	7,504
Uniform	3	4,000	16,000	15,173 (164)	**15,182** (165)	13,475	14,983
Uniform	3	6,000	24,000	22,756 (360)	**22,798** (311)	21,053	19,973
Non-uniform	3	2,000	4,000	3,893 (22)	3,893 (47)	3,843	**3,908**
Non-uniform	3	2,000	8,000	7,303 (43)	**7,319** (71)	6,574	7,200
Non-uniform	4	2,000	8,000	7,956 (73)	**7,961** (85)	5,206	7,940
Non-uniform	4	6,000	24,000	23,853 (320)	**23,874** (718)	22,627	23,785

hypergraphs. Although neural solvers demonstrate a runtime advantage on most datasets, Gurobi still achieves better results on certain instances.

C LIMITATION

Table 9: The VRAM usage of HyperNCO on hypergraph partitioning, where $k = 3$.

$\|V\|$	$\|E\|$	**VRAM (GB)**
10,000	40,000	16.4
15,000	60,000	41.8

As we have discussed in Sec. 4.1, the VRAM usage of HyperNCO depends on a dense tensor of shape $|V| \times |E| \times k$. Thus, Evaluating the limits of the $|V| \times |E| \times k$ is crucial. To address this, we have included additional validation of HyperNCO's scalability as shown in Tab. 9.

Since we use two NVIDIA RTX 3090 GPU with 24 GB of VRAM, $|V| = 15,000$, $|E| = 60,000$, $k = 3$ represents the maximum problem scale that HyperNCO can handle. Scaling to larger problem sizes requires more VRAM.

D INTEGER PROGRAMMING FORMULATIONS FOR GUROBI AND SCIP

D.1 HYPERGRAPH PARTITIONING

$$
\begin{aligned}
\min_{x,z,y,s_{\min},s_{\max}} \quad & \sum_{e\in E} y_e \\
\text{s.t.} \quad & \sum_{k=1}^{K} x_{v,k} = 1, \quad \forall v \in V \quad \text{(Each node belongs to exactly one partition.)} \\
& z_{e,k} \le x_{v,k}, \quad \forall e \in E, \forall k \in \{1,\dots,K\}, \forall v \in e \\
& \sum_{v\in e} x_{v,k} \ge |e| \cdot z_{e,k} - (|e|-1), \quad \forall e \in E, \forall k \in \{1,\dots,K\} \\
& \quad (z_{e,k} = 1 \text{ if and only if all nodes in the hyperedge belong to partition } k.) \\
& \sum_{k=1}^{K} z_{e,k} + y_e \ge 1, \quad \forall e \in E \\
& \sum_{k=1}^{K} z_{e,k} \le 1 - y_e, \quad \forall e \in E \quad \text{(Whether hyperedge } e \text{ is cut)} \\
& s_{\min} \le \sum_{v\in V} x_{v,k} \le s_{\max}, \quad \forall k \in \{1,\dots,K\} \\
& s_{\max} - s_{\min} \le \delta \quad \text{(The balancedness)} \\
& x_{v,k}, z_{e,k}, y_e \in \{0,1\} \\
& s_{\min}, s_{\max} \in \mathbb{Z}_{\ge 0}
\end{aligned}
\tag{10}
$$

where the balance tolerance δ is set to 1.

- $x_{v,k} \in \{0,1\}$: 1 if node v is assigned to partition k; 0 otherwise.
- $z_{e,k} \in \{0,1\}$: 1 if and only if all nodes in hyperedge e belong to partition k; 0 otherwise.
- $y_e \in \{0,1\}$: 1 if hyperedge e is cut (i.e., its nodes span across multiple partitions); 0 otherwise.
- $s_{\min}, s_{\max} \in \mathbb{Z}_{\ge 0}$: Auxiliary variables representing the minimum and maximum partition sizes.

D.2 HYPERGRAPH PROPER COLORING

$$
\begin{aligned}
\min_{x,y} \quad & \sum_{k=1}^{K_{\max}} y_k \\
\text{s.t.} \quad & \sum_{k=1}^{K_{\max}} x_{v,k} = 1, \quad \forall v \in V \quad \text{(Each node has exactly one color.)} \\
& \sum_{v \in e} x_{v,k} \leq |e| - 1, \quad \forall e \in E,\, \forall k \in \{1, \ldots, K_{\max}\} \\
& \quad \text{(No hyperedge is allowed to have all its nodes of the same color.)} \\
& x_{v,k} \leq y_k, \quad \forall v \in V,\, \forall k \in \{1, \ldots, K_{\max}\} \\
& x_{v,k}, y_k \in \{0, 1\}
\end{aligned}
\tag{11}
$$

where

- $x_{v,k} \in \{0, 1\}$: 1 if node v is assigned color k; 0 otherwise.
- $y_k \in \{0, 1\}$: 1 if color k is used by at least one node; 0 otherwise.

D.3 HYPERGRAPH MAX-k-CUT

$$
\begin{aligned}
\max_{x,z,y} \quad & \sum_{e \in E} y_e \\
\text{s.t.} \quad & \sum_{i=1}^{k} x_{v,i} = 1, \quad \forall v \in V \\
& z_{e,i} \leq x_{v,i}, \quad \forall e \in E,\, \forall i \in \{1, \ldots, k\},\, \forall v \in e \\
& \sum_{v \in e} x_{v,i} \geq |e| \cdot z_{e,i} - (|e| - 1), \quad \forall e \in E,\, \forall i \in \{1, \ldots, k\} \\
& y_e = 1 - \sum_{i=1}^{k} z_{e,i}, \quad \forall e \in E \\
& x_{v,i}, z_{e,i}, y_e \in \{0, 1\}
\end{aligned}
\tag{12}
$$

where

- $x_{v,i} \in \{0, 1\}$: 1 if node v is assigned to partition $i \in \{1, \ldots, k\}$; 0 otherwise.
- $z_{e,i} \in \{0, 1\}$: 1 if and only if all nodes in hyperedge e belong to partition i; 0 otherwise.
- $y_e \in \{0, 1\}$: 1 if hyperedge e is cut (i.e., its nodes span across at least two distinct partitions); 0 otherwise.

E THE MIS PROBLEM

The MIS problem for an ordinary graph $G = (V, E)$ can be formulated by counting the number of marked (colored by 1) vertices belonging to the independent set and adding a penalty when two vertices are connected by an edge.

Integer Programming Form. Given a graph $G = (V, E)$, the MIS problem can be formulated as the follows:

$$
\begin{aligned}
\min \quad & -\sum_{i \in V} x_i && (13) \\
\text{s.t.} \quad & x_i x_j = 0 \quad \text{for all} \quad (i, j) \in E && (14) \\
& x_i \in \{0, 1\} && (15)
\end{aligned}
$$

QUBO Form. The MIS problem can be formulated as the QUBO form by using penalty method:

$$\min \quad -\sum_{i \in V} x_i + P \sum_{(i,j) \in E} x_i x_j \tag{16}$$

where P is the penalty factor and $x_i \in \{0, 1\}$. $P > 0$ enforces the constraint that x_i and x_j cannot both be equal to 1 simultaneously. As illustrated in Fig. 2, the MIS problem is phrased in QUBO form by constructing the matrix:

$$Q = \begin{cases} P & \text{if } (i,j) \in E \\ -1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

The MIS problem is then to minimize the cost function $\mathbf{x}^\top Q \mathbf{x}$, where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$.