BIPNN: LEARNING TO SOLVE BINARY INTEGER PROGRAMMING VIA HYPERGRAPH NEURAL NETWORKS

Sen Bai
Changchun University of Science and Technology, China
baisen@cust.edu.cn

Chunqi Yang
Changchun University of Science and Technology, China
yangchunqi@mails.cust.edu.cn

Xin Bai
Huawei Technologies Co. Ltd China
baixinbs@163.com

Xin Zhang
Changchun University of Science and Technology, China
zhangxin@cust.edu.cn

Zhengang Jiang
Changchun University of Science and Technology, China
jiangzhengang@cust.edu.cn

May 28, 2025

ABSTRACT

Binary (0-1) integer programming (BIP) is pivotal in scientific domains requiring discrete decision-making. As the advance of AI computing, recent works explore neural network-based solvers for integer linear programming (ILP) problems. Yet, they lack scalability for tackling nonlinear challenges. To handle nonlinearities, state-of-the-art Branch-and-Cut solvers employ linear relaxations, leading to exponential growth in auxiliary variables and severe computation limitations. To overcome these limitations, we propose **BIPNN** (**B**inary **I**nteger **P**rogramming **N**eural **N**etwork), an **unsupervised** learning framework to solve nonlinear BIP problems via hypergraph neural networks (HyperGNN). Specifically, **(I)** BIPNN reformulates BIPs-constrained, discrete, and nonlinear ($\sin$, $\log$, $\exp$) optimization problems-into unconstrained, differentiable, and polynomial loss functions. The reformulation stems from the observation of a precise one-to-one mapping between polynomial BIP objectives and hypergraph structures, enabling the unsupervised training of HyperGNN to optimize BIP problems in an end-to-end manner. On this basis, **(II)** we propose a GPU-accelerated and continuous-annealing-enhanced training pipeline for BIPNN. The pipeline enables BIPNN to optimize large-scale nonlinear terms in BIPs fully in parallel via straightforward gradient descent, thus significantly reducing the training cost while ensuring the generation of discrete, high-quality solutions. Extensive experiments on synthetic and real-world datasets highlight the superiority of our approach.

1 Introduction

For decades, binary integer programming (BIP)—a powerful mathematical tool characterized by discrete binary decision variables (0 or 1)—is of critical importance in numerous domains, such as operational optimization [1, 2, 3], quantum computing [4, 5, 6], computational biology [7, 8], materials science and computational chemistry [9, 10]. However, BIP is known to be NP-complete [11], making large-scale BIP instances computationally intractable.

Along with AI computing shines in scientific discovery, the potential of neural network-based IP solvers has emerged in recent years. To address integer linear programming (ILP) problems, MIP-GNN [12] leverages graph neural networks (GNN) to improve the performance. Another GNN&GBDT-guided framework [13] for large-scale ILP problems can save up 99% of running time in achieving the same solution quality as SCIP [14], a leading IP solver. However, these neural network-based ILP solvers lack scalability for nonlinear BIPs.

To handle nonlinearities, state-of-the-art Branch-and-Cut solvers (e.g., SCIP [15]) rely on linear relaxation, which introduces a number of auxiliary variables. Once linearized, these problems are solved using linear programming

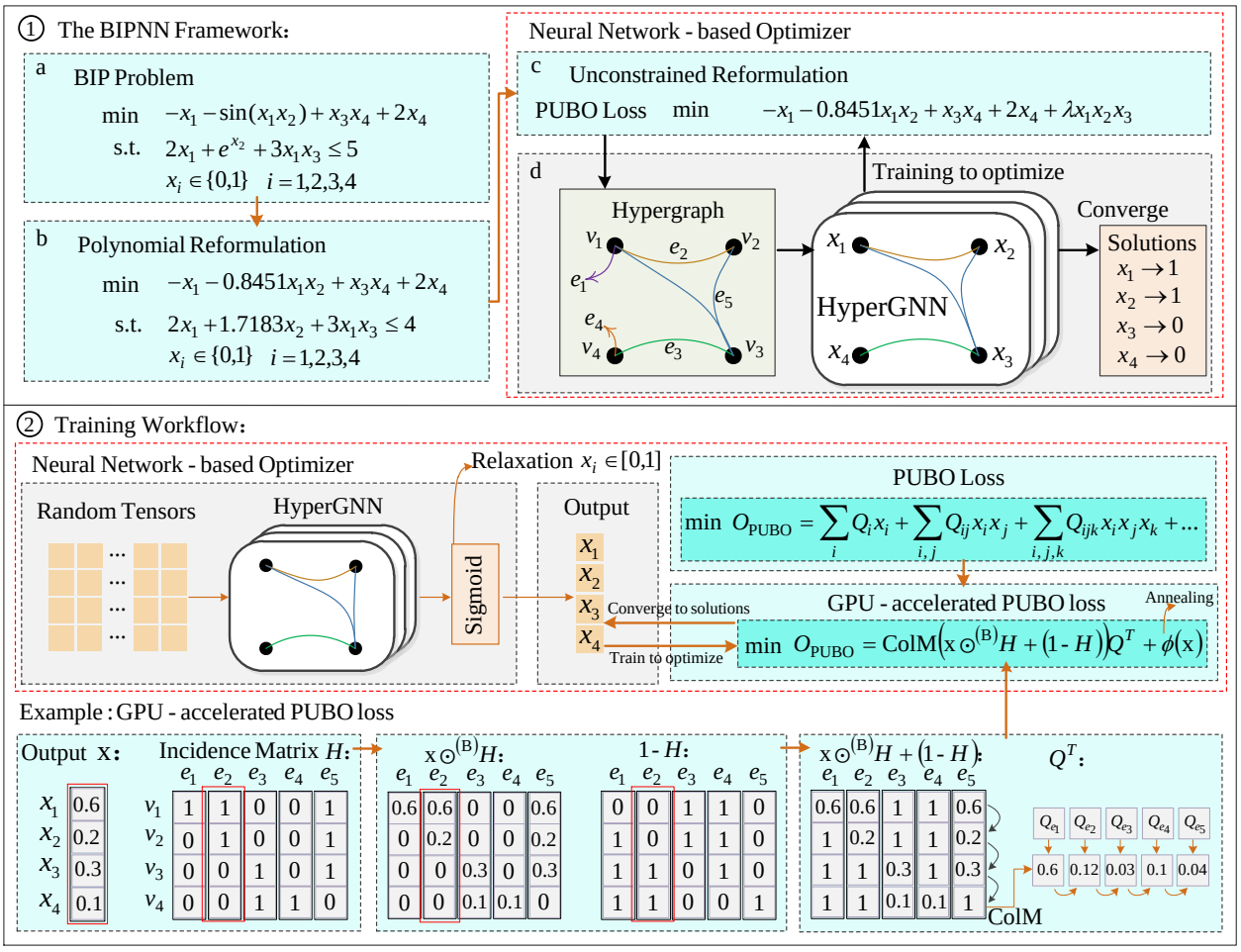


Figure 1: The BIPNN framework.

(LP) solvers (e.g., the Simplex method[1]). Consequently, large-scale nonlinear BIPs often suffer from prohibitive computational costs. As BIP solvers continue to evolve, linearization remains indispensable for making nonlinearities more tractable for BIP solvers.

These limitations motivate us to develop a streamlined and general-purpose BIP solver to advance the state of the art. To profoundly adapt to real-world applications, our work grapples with challenges arising from neural networks' unique characteristics beyond linearization-based methods, as summarized below:

Challenge 1. Meticulously modeling nonlinear terms in BIP objectives and constraints;

Challenge 2. Utilizing GPU's parallel computing capability.

To this end, in this work we propose **BIPNN** (**B**inary **I**nteger **P**rogramming **N**eural **N**etwork), an unsupervised BIP solver that bridges the gap between nonlinear BIP and deep neural networks. Our overarching idea stems from the observation of one-to-one mapping correspondence between polynomial BIP objectives and hypergraph structures (upper right of Fig. 1). As depicted in Fig. 1, our framework consists of three phases:

1) In the first phase, we employ broadly applicable penalty term method to convert constrained BIP problems into polynomial unconstrained binary optimization (PUBO[2]) formalism. To handle exponential and trigonometric terms, we propose a novel transformation to represent them in the form of polynomials. These refined polynomial objectives are adaptable to neural network-based solvers when applied as loss functions.

[1]To be precise, the Simplex method is designed to solve linear programming (LP) problems in polynomial time, meaning they belong to the class P [16].

[2]The mathematical formulation PUBO is well-known in quantum computing, for modeling complex optimization problems in a way quantum computers may solve efficiently.

2) In the second phase, we leverage hypergraph neural networks (HyperGNN) to address **Challenge 1**, capturing high-order correlations between binary decision variables, or in other words the polynomial terms in the refined PUBO objective. By applying a relaxation strategy to the PUBO objective to generate a differentiable loss function with which we train the HyperGNN in an unsupervised manner.

3) Nevertheless, when we train these HyperGNNs to minimize the PUBO objectives, we encounter severe obstacles of low computational efficiency in these polynomial losses with numerous variables. In the third phase, leveraging GPUs, we further propose an algorithm to address **Challenge 2** via matrix operations on the incidence matrices of hypergraphs.

In summary, we contribute:

1) BIPNN, an unsupervised HyperGNN-based solver that allows learning approximate BIP solutions in an end-to-end differentiable way with strong empirical performance.

2) An empirical study of the performance of BIPNN on synthetic and real-world data, demonstrating that unsupervised neural network solvers outperform classic BIP solvers such as SCIP and Tabu in tackling large-scale nonlinear BIP problems.

3) Large-scale nonlinear optimization has long been challenging due to its inherent complexity and scalability issues. We advance this field by employing several nonlinearity modeling methods for BIP, including the polynomial reformulation and unconstrained reformulation. These methods provide instructive guidance for unsupervised neural network-based solvers.

2 Notations and Definitions

In the following, we will formulate the BIP problem and articulate the definition of hypergraphs.

Definition 1 (Formulation of BIP). *Non-linear BIP is an optimization problem where the decision variables* $\mathbf{x} = (x_1, x_2, ..., x_m)$ *are restricted to binary values (*0 *or* 1*), and the objective function* O_{BIP} *or constraints (or both) are nonlinear. Below is the general formulation.*

$$\begin{aligned} \min \quad & O_{\text{BIP}} = f(\mathbf{x}) \\ \text{s. t.} \quad & g_k(\mathbf{x}) \leq 0 \quad \text{for all} \quad k = 1, 2, \ldots, K \\ & q_l(\mathbf{x}) = 0 \quad \text{for all} \quad l = 1, 2, \ldots, L \\ & x_i \in \{0, 1\} \quad \text{for all} \quad i = 1, 2, \ldots, n \end{aligned} \tag{1}$$

where $f(\mathbf{x})$*,* $g_k(\mathbf{x})$ *and* $q_l(\mathbf{x})$ *are nonlinear functions of the decision variables* $\mathbf{x}$*.* □

Definition 2 (Hypergraph). *A hypergraph is defined by* $G = (V, E)$*, where* $V = \{v_1, v_2, ..., v_{|V|}\}$ *stands for a set of vertices and* $E = \{e_1, e_2, ..., e_{|E|}\}$ *denotes a set of hyperedges. Each hyperedge* $e_j \in E$ *is a subset of* V*. A hypergraph* G *can be represented by the incidence matrix (Fig. 1 at the bottom)* $H \in \{0, 1\}^{|V| \times |E|}$*, where* $H_{ij} = 1$ *if* $v_i \in e_j$*, or otherwise* $H_{ij} = 0$*.* □

3 BIPNN: HyperGNN-based Optimizer for PUBO-formulated BIP

For easier comprehension of our approach, in this section we first elaborate how to solve an unconstrained, PUBO-formulated BIP problem as depicted in Eq. 2. Then, in Sec. 4, we will show how to transform a general BIP problem with constraints and nonlinear terms into PUBO formalism.

3.1 Modeling PUBO-formulated BIPs via Hypergraphs

BIPNN employs a HyperGNN-based optimizer (upper right of Fig. 1) to solve PUBO-formulated BIP problems. Inspired by the binary characteristic of variables, we can reformulate general BIPs as PUBO problems through the polynomial reformulation in Sec.4.1 and unconstrained reformulation in Sec.4.2. A PUBO problem is to optimize the cost function:

$$O_{\text{PUBO}} = \sum_i Q_i x_i + \sum_{i,j} Q_{ij} x_i x_j + \sum_{i,j,k} Q_{ijk} x_i x_j x_k + \cdots \tag{2}$$

where $x_i \in \{0, 1\}$ are binary descision variables and the set of all decision variables is denoted by $\mathbf{x} = (x_1, x_2, \cdots, x_m)$. As shown in Fig. 2, for ease of representation, a PUBO objective O_{PUBO} with n terms can be decomposed into two components: the PUBO matrix $Q = [Q_1, Q_2, ..., Q_n]$, and n linear or polynomial terms such as x_i, $x_i x_j$, or $x_i x_j x_k$.

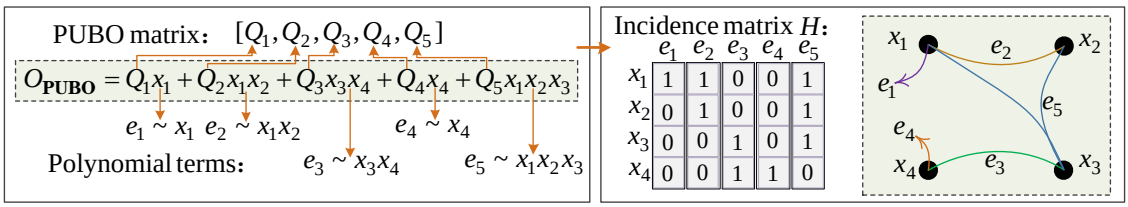


Figure 2: Modeling PUBO-formulated BIPs via hypergraphs.

In this way, we discover multi-variable interactions in O_{PUBO} can be modeled as a hypergraph $G = (V, E)$, where $|E| = n$, and each hyperedge $e \in E$ encodes a single descision variable x_i or a polynomial term such as x_ix_j or $x_ix_jx_k$.

3.2 Neural Network-based Optimizer

The training workflow of the neural network-based optimizer is illustrated at the bottom of Fig. 1.

HyperGNN Architecture. Initially, for a PUBO-transformed hypergraph $G = (V, E)$, HyperGNNs take the incidence matrix H of G and a randomly initialized $X^{(0)} \in \mathbb{R}^{m\times d}$ as inputs. Subsequently, BIPNN applies the sigmoid function to produce the output vector $\mathbf{x} = (x_1, x_2, \cdots, x_m)$, where $x_i \in [0, 1]$ are the relaxation of decision variables $x_i \in \{0, 1\}$. The HyperGNN model operates as follows:

$$\mathbf{x} = \text{sigmoid}(\text{HyperGNN}(H, X^{(0)})) \tag{3}$$

where HyperGNN is a multi-layer hypergraph convolutional network, such as HGNN+ [17], HyperGCN [18], or UniGCN [19].

Training to Optimize. As an unsupervised learning model, BIPNN relaxes the PUBO objective O_{PUBO} into a differentiable loss function and trains to optimize it. Specifically, O_{PUBO} can be expressed by the output $\mathbf{x}$ and the incidence matrix H as depicted in Fig. 1. We aim to find the optimal solution $\mathbf{x}^s = \text{argmin}O_{\text{PUBO}}(\mathbf{x}, H)$. As training progresses, $x_i \in \mathbf{x}$ will gradually converge to binary solutions.

GPU-accelerated Training. For a large-scale BIP problem, numerous polynomial terms in O_{PUBO} lead to a high computational cost. To address this, an intuitive idea is to leverage GPU-supported matrix operations to accelerate training. However, PUBO problems lack a straightforward matrix formulation. To this end, we propose GPU-accelerated PUBO objective as follows.

$$O_{\text{PUBO}} = \text{ColM}(\mathbf{x} \odot^{(\text{B})} H + (1 - H))Q^T \tag{4}$$

where $\mathbf{x}$ is the output of HyperGNN, H is the incidence matrix, and $Q = [Q_1, Q_2, ..., Q_n]$ is the PUBO matrix. More concretely, $\mathbf{x} \odot^{(\text{B})} H$ denotes the element-wise Hadamard product with broadcasting between m-dimensional vector $\mathbf{x}$ and matrix $H \in \mathbb{R}^{m\times n}$. We add $1 - H$ on $\mathbf{x} \odot^{(\text{B})} H$ to fill zero-valued elements with 1. Based on this operation, we use the column-wise multiplication denoted by ColM on the first dimension of the matrix obtained by $\mathbf{x} \odot^{(\text{B})} H + (1 - H)$. Through the ColM operation we obtain an n-dimensional vector, of which each element represents a polynomial term in O_{PUBO}. The final loss function is computed by scaling each polynomial term with its respective coefficient Q_i. The detailed explanation is illustrated in Fig. 1.

Time Complexity Analysis. For $\mathbf{x} \in \mathbb{R}^m$, $Q \in \mathbb{R}^{1\times n}$, and $H \in \mathbb{R}^{m\times n}$, the time complexity of Eq. 4 is $O(m \times n)$. For GPU-accelerated training, element-wise operations such as Hadamard product are fully parallelizable. Column-wise product over m leads to time complexity $O(\log m)$. Thus, the theoretical best GPU time complexity is $O(\log m)$. Utilizing T cores, the realistic GPU time complexity is $O(\frac{m\times n}{T})$.

Annealing Strategy. To achieve unsupervised learning, BIPNN relaxes PUBO problems into continuous space. The differentiable relaxation of discrete decision variables sometimes leads to continuous solutions $x_i \in [0, 1]$. To address this, we employ the continuous relaxation annealing (CRA) [20] method. Specifically, BIPNN uses the following loss function: $O_{\text{PUBO}} = \text{ColM}(\mathbf{x} \odot^{(\text{B})} H + (1 - H))Q^T + \phi(\mathbf{x})$, where $\phi(\mathbf{x}) = \gamma \sum_{i=1}^{n}(1 - (2x_i - 1)^\alpha)$ is the penalty term, γ controls the penalty strength and α is an even integer. We initialize $\gamma < 0$ and gradually increase it to a positive value as training progresses. The annealing strategy enhances the performance of BIPNN in three aspects, **(i)** In the high-temperature phase ($\gamma < 0$), it smooths the HyperGNN, preventing it from getting trapped in local optima; **(ii)** In the low-temperature phase ($\gamma > 0$), it enforces the discreteness of solutions; **(iii)** It effectively accelerates the training process.

4 BIPNN: Polynomial & Unconstrained Reformulation of BIP

In this section, we explain how to reformulate nonlinear BIPs as unconstrained and polynomial optimization problems, which are compatible with our neural network-based optimizer.

4.1 Polynomial Reformulation of BIP

Our approach is inspired by the observation that for any binary variable, a nonlinear term such as e^x can be exactly fitted by a polynomial equivalent $h(x) = ax + b$, such that $h(x) = e^x$ for $x \in \{0, 1\}$. That is, $h(x) = (e-1)x + 1$, where $h(0) = 1$ and $h(1) = e$. To handle univariate nonlinearities, including trigonometric, logarithmic, and exponential terms (e.g., $\sin x$, $\log x$, and e^x), we have the following transformation: $h(x) = (h(1) - h(0))x + h(0)$. For multivariate terms such as $e^{x_i x_j}$ and $sin(x_i x_j)$, where $x_i x_j \in \{0, 1\}$, we can perform the transformation as follows: $h(\prod_{i \in S} x_i) = (h(1) - h(0)) \prod_{i \in S} x_i + h(0)$.

BIPNN employs a more general method to handle more intricate multivariate nonlinear terms (such as $\sin(x_i + x_j)$). For a set of binary decision variables $x_1, x_2, ..., x_n$, a non-linear function $h(x_1, x_2, ..., x_n)$ can be transformed into the polynomial forms as follows.

$$h(x_1, x_2, ..., x_m) = \sum_{S \subseteq \{1,2,...,m\}} c_S \prod_{i \in S} x_i \tag{5}$$

By setting up a system of equations based on all possible combinations of $x_1, x_2, ..., x_m$, we can determine the coefficients c_S to precisely fit $h(x_1, x_2, ..., x_m)$ by leveraging simple inclusion-exclusion principle (refer to Appendix A) as below.

$$c_S = \sum_{T \subseteq S} (-1)^{|S|-|T|} f(T) \tag{6}$$

where $f(T)$ represents the function value when the variables in the subset T are 1 and the others are 0. For each subset S, it needs to calculate $2^{|S|}$ values of $f(T)$. □

As an example, we have $\sin(x_1 + x_2) = 0.8415x_1 + 0.8415x_2 - 0.7737x_1x_2$. A toy example of $\sin(x_1 + x_2 + x_3)$ is illustrated in Appendix A. To be noticed, polynomial reformulation of all nonlinear terms in a BIP objective is not necessary. If the transformation becomes overly complex, we may opt to retain the original nonlinear term and directly incorporate it as part of the loss function of HyperGNN.

4.2 Unconstrained Reformulation of BIP

We propose a novel penalty method to transform the constrained BIP problem into an unconstrained form. In penalty methods [21, 22], unconstrained reformulation is achieved by adding "penalty terms" to the objective function that penalize violations of constraints. A well-constructed penalty term must be designed such that it equals 0 if and only if the constraint is satisfied, and takes a positive value otherwise. Specifically, given a BIP problem in Eq. 1, for inequality constraints $g_k(\mathbf{x}) \leq 0$, we have penalty terms $P_k(\mathbf{x}) = \lambda_k \cdot (\max(0, g_k(\mathbf{x})))^2$, for equality constraints $q_l(\mathbf{x}) = 0$, we have penalty terms $Q_l(\mathbf{x}) = \mu_l \cdot (q_l(\mathbf{x}))^2$, where λ_k, μ_l are sufficiently large penalty coefficients. By combining all terms into a single objective function, we have an unconstrained BIP objective:

$$\min \quad O_{\text{BIP}} = f(\mathbf{x}) + \sum_{k=1}^{K} \lambda_k \cdot (\max(0, g_k(\mathbf{x})))^2 + \sum_{l=1}^{L} \mu_l \cdot (q_l(\mathbf{x}))^2 \tag{7}$$

As part of the loss function of BIPNN, O_{BIP} must be differentiable to enable gradient-based optimization. However, $\max(0, g_k(\mathbf{x}))$ is not a continuously differentiable function, thus finding an appropriate penalty term is crucial. We propose two methods to address this issue:

1) **ReLU-based Penalty**. We can use $\text{ReLU}(g_k(\mathbf{x}))^2 = (\max(0, g_k(\mathbf{x})))^2$ to handle constraints. This is a general method for a large number of variables x_i in a constraint $g_k(\mathbf{x})$.

2) **Polynomial Penalty**. In the following, we present an algorithm to construct polynomial penalty terms with 2^Δ time complexity for $g_k(\mathbf{x})$, where Δ is the number of variables in constraint $g_k(\mathbf{x})$.

For binary variables, do there exist polynomial penalty terms that correspond to BIP constraints? To answer this question, we have the following discussion. For $x_1 + 2x_2 - 2 \leq 0$, we observe that the violating subset $\{x_1 = 1, x_2 = 1\}$ corresponds to polynomial penalty term $\lambda(x_1x_2)$. For another constraint $x_1 + 3x_2 - 2 \leq 0$, the violating subsets $\{x_1 = 0, x_2 = 1\}$ and $\{x_1 = 1, x_2 = 1\}$ correspond to the polynomial penalty term $\lambda(x_2 + x_1x_2)$ or λx_2. Through an in-depth analysis, we propose a novel method to transform nonlinear BIP constraints into polynomial penalty terms. To

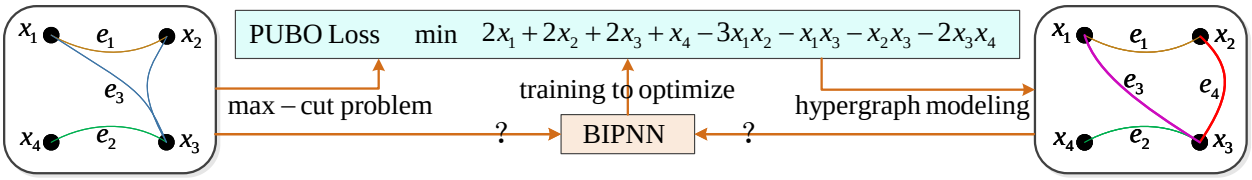


Figure 3: To solve the hypergraph max-cut problem, BIPNN generates a new hypergraph structure. However, both of these hypergraphs can be utilized for training the HyperGNN model.

handle an inequality constraint $g(\mathbf{x}) \leq 0$ for the BIP problem in Eq. 1, our method consists of three steps (to see a toy example, refer to Appendix B):

(i) Initially, we express the constraint $g(\mathbf{x}) \leq 0$ as a boolean indicator function: $\psi(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \text{ (violation)} \\ 0 & \text{otherwise (feasible)} \end{cases}$, then define minimal violation subsets $\mathcal{V}$ as the smallest variable combinations causing constraint violations:

$$\mathcal{V} = \left\{ S \subseteq \{1, ..., n\} \,\middle|\, \psi(\mathbf{x}) = 1 \text{ when } x_i = 1 \; \forall i \in S \text{ and } x_j = 0 \; \forall j \notin S \right\} \tag{8}$$

each $S \in \mathcal{V}$ cannot be reduced further without eliminating the violation.

(ii) Generate a penalty term for each minimal violation subset $S \in \mathcal{V}$:

$$P(\mathbf{x}) = \lambda \sum_{S \in \mathcal{V}} \prod_{i \in S} x_i \tag{9}$$

where λ is the penalty coefficient.

(iii) Combine each term into the BIP objective function:

$$\min \quad O_{\text{BIP}} = f(\mathbf{x}) + P(\mathbf{x}) \tag{10}$$

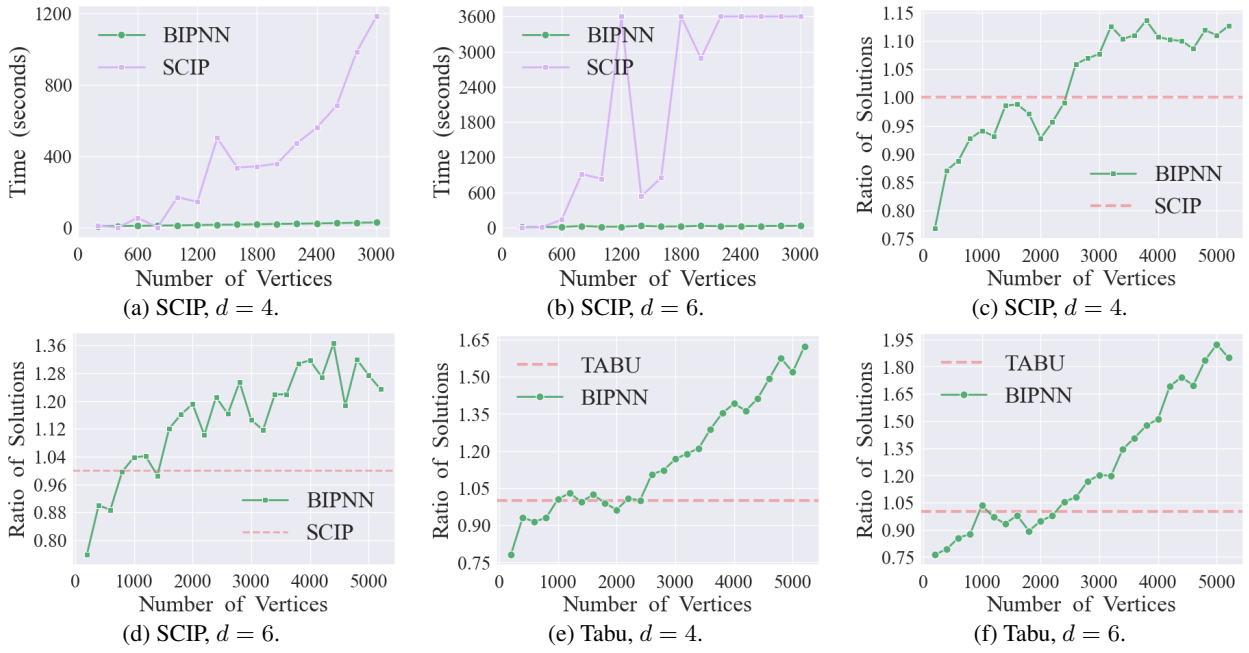
In the worst case, when an enumeration method is used in step **(i)**, it requires calculating 2^{Δ} subsets, where Δ is the number of variables in constraint $g(\mathbf{x})$. Nevertheless, in most real-world problems (e.g. max-cut, and maximal independent set or MIS) involving graphs, the variables associated with each constraint often exhibit locality. □

The polynomial penalty method facilitates to incorporate penalty terms to PUBO objectives and use GPU-accelerated training pipeline to solve BIPs. As far as we know, only a few number of constraint/penalty pairs [22] associated have been identified in existing literature. Our work significantly expands the potential application domains of the penalty method.

5 Discussion

Feasible Solutions. Firstly, a PUBO problem always has feasible solutions. The feasible set is the entire space of binary variable combinations, since there are no constraints to exclude any combination. Every possible binary assignment $x_i \in \{0, 1\}$ is inherently feasible. Secondly, the feasibility of a nonlinear BIP problem depends on the constraint compatibility—whether there exists at least one binary variable assignment $\mathbf{x} \in \{0, 1\}^m$ that satisfies all nonlinear constraints simultaneously. In BIPNN, we determine the existence of feasible solutions through **(i)** Training-phase feasibility check: if all penalty terms (e.g., constraint violations) converge to zero during training, feasible solutions exist; otherwise, the problem is infeasible. **(ii)** Post-training verification: we sample candidate solutions from the trained model and explicitly verify whether they satisfy all constraints.

The Effectiveness of BIPNN's Hypergraph Generation Mechanism. As depicted in Fig. 3, when BIPNN is applied to solve combinatorial optimization (CO) problems on hypergraphs, it generates an alternative hypergraph structure. However, both of the hypergraphs can be used as the input of BIPNN. A critical question arises: which type of hypergraph structure achieves better performance when applied to HyperGNN? The main difference between these two hypergraphs is that the hypergraph generated by BIPNN breaks down the original hypergraph's high-order hyperedges into numerous low-order ones. We argue that BIPNN training with the original hypergraph structure is more computationally efficiency, while BIPNN-generated hypergraph structure leads to more optimal solutions. In Sec. 6.3, we will empirically compare the solution quality of both methods.



(a) SCIP, $d = 4$. (b) SCIP, $d = 6$. (c) SCIP, $d = 4$.

(d) SCIP, $d = 6$. (e) Tabu, $d = 4$. (f) Tabu, $d = 6$.

Figure 4: Comparison of BIPNN and existing BIP solvers. d is the degree of polynomial terms in BIP objective functions. (a)(b) show the solving time required for BIPNN and SCIP to obtain the same solution. (c)(d) show the ratio of the solutions of BIPNN to SCIP; (e)(f) illustrate the ratio of the solutions of BIPNN to Tabu; Runtime is restricted to 1 hour.

6 Experimental Results

In this section, we describe our empirical experiments on BIPNN and baseline optimization tools.

Benchmarks. To evaluate BIPNN on BIP problems with diverse scales, the datasets are generated using DHG library[3]. To evaluate the quality of solutions and computational efficiency of BIPNN, datasets of varying scales are generated in three steps: Initially, DHG library is applied to generate hypergraph structures (where $|E| = 2|V|$). Subsequently, a random coefficient is assigned to each hyperedge (representing a polynomial term) to generate PUBO objective functions. Thereafter, several constraints (penalty terms) were randomly incorporated into the PUBO objectives. To demonstrate the effectiveness of BIPNN on real-world settings, we also conduct experiments on the hypergraph max-cut problem (refer to Appendix C), a well-known BIP problem benchmark. Moreover, we conduct experiments on publicly-available hypergraph datasets (refer to Appendix D).

Baseline Methods. In our experiments, the baseline methods include optimization techniques and tools such as SCIP [14], Tabu search [23].

Implementation Details. Experiments are conducted on an Intel Core i9-12900K CPU with 24 cores, and an NVIDIA GeForce RTX 3090 GPU with 24 G of memory. We adopt two-layer HGNN+ [17] as the HyperGNN model for the experiments.

6.1 Comparison with Linearization-based BIP Solvers

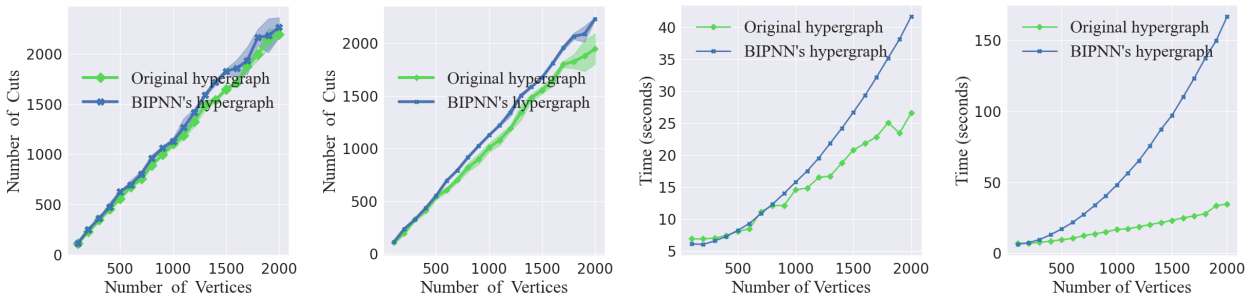
SCIP. SCIP is an exact solver based on the branch-and-cut algorithm. Theoretically, given sufficient time and computational resources, SCIP guarantees an exact solution. However, for large-scale problems, due to time constraints, SCIP may terminate prematurely and return the approximate solution. To conduct the experiment, we generate a specific BIP instance for each size of variables. Specifically, for a BIPNN-generated hypergraph, the number of vertices (variables) $|V|$ ranges from 200 to 3000. The degrees of vertices are set to 4 (Fig. 4a) and 6 (Fig. 4b) respectively.

Fig. 4a and Fig. 4b show the comparison of the solving time for BIPNN and SCIP. We evaluate the solving time taken by BIPNN to obtain the best approximate solution and the time required by SCIP to find the same solution. Experimental

[3]https://deephypergraph.readthedocs.io/en/latest/index.html

Table 1: The solutions of graph/hypergraph max-cut problems (1-hour time limit).

Method	BAT	EAT	UAT	DBLP	CiteSeer	Primary	High	Cora
SCIP	655	3,849	7,899	2,869	3,960	7,603	4,599	1,215
Tabu	652	3,972	8,402	2,710	3,717	8,500	5,160	1,360
BIPNN	651	**3,978**	**8,407**	2,801	3,852	**8,509**	**5,216**	**1,384**



(a) $d = 4$. (b) $d = 6$. (c) $d = 4$. (d) $d = 6$.

Figure 5: Comparison of the quality of solutions and time efficiency of BIPNN when it applys its generated hypergraph structure or the original hypergraph structure to solve hypergraph max-cut problems. d is the degree of polynomial terms in BIP objective functions. (a)(b) show the numbers of cuts; (c)(d) show the solving time.

results demonstrate that the solving time of BIPNN grows linearly and slowly with increasing problem size, while SCIP's solving time exhibits exponential growth. This trend becomes more pronounced when the degree of polynomial terms is 6.

Moreover, we impose a 1-hour time limit and evaluate the solution quality of BIPNN and SCIP across varying scales of BIP instances. Fig. 4c and Fig. 4d show the comparative ratio of solutions obtained by BIPNN and SCIP. Specifically, the comparative ratio is defined as $\frac{O^s_{\text{BIPNN}}}{O^s_{\text{SCIP}}}$, where O^s_{BIPNN} and O^s_{SCIP} are the solutions obtained by BIPNN and SCIP. Experimental results demonstrate that BIPNN starts outperforming SCIP when the number of variables exceeds $2,500$ when $d = 4$. As the problem size increases, BIPNN's solutions increasingly outperform SCIP's solutions. For $d = 6$, BIPNN outperforms SCIP when the number of vertices exceeds $1,000$.

Tabu Search. Tabu search is a heuristic method that typically provides approximate solutions. We also impose a 1-hour time limit and evaluate the difference in solution quality for Tabu when the degrees of polynomial terms are set to 4 and 6. The number of vertices (variables) $|V|$ in the hypergraph generated by BIPNN ranges from 200 to $5,000$. Experimental results are depicted in Fig. 4e ($d = 4$) and Fig. 4f ($d = 6$). As shown in the figures, BIPNN achieves the performance comparable to Tabu when the number of variables exceeds $1,000$. When the number of variables exceeds $2,500$, BIPNN significantly outperforms Tabu as the variable count increases further.

6.2 Comparison on Real-world Datasets

We compare our method against baseline methods on real-world graph and hypergraph datasets, including BAT, EAT, UAT, DBLP, CiteSeer, Primary, High, and Cora (refer to Appendix D). Graph datasets include BAT, EAT, UAT, DBLP, and CiteSeer. Hypergraph datasets include Primary, High, and Cora. Graph and hypergraph max-cut problems are selected as the BIP problem benchmarks. We impose 1 hour time limit and evaluate the number of cuts obtained by BIPNN, SCIP, and Tabu. As depicted in Tab. 1, SCIP achieved the best performance on three graph datasets, while BIPNN achieved the best performance on two graph datasets and all three hypergraph datasets. In summary, compared to the graph max-cut problem, due to higher degree of polynomial terms in the objective function of the hypergraph max-cut problem, BIPNN tends to achieve better performance on hypergraph datasets.

6.3 Comparative Analysis on Hypergraph Generation Mechanism

In Sec. 5 and Fig. 3, we propose to evaluate the effectiveness of BIPNN's hypergraph generation mechanism by comparing the effects of its generated hypergraph structures against the original hypergraph structures in a hypergraph CO problem. In this section, we select hypergraph max-cut as benchmark and conduct experiments to evaluate the performance of BIPNN under both of the hypergraph structures. Experimental results are depicted in

Fig. 5. The number of variables ranges from 100 to 2000. The degrees of polynomial terms d are set to $d = 4$ and $d = 6$ respectively. We perform 10 tests each time and record the average value of the cut numbers. As illustrated in Fig. 5a and Fig. 5b, the hypergraph structure generated by BIPNN can identify more cuts in comparison. However, as depicted in Fig. 5c and Fig. 5d, when the parameter d is larger, the number of hyperedges (polynomial terms in PUBO objectives) in the hypergraph structure generated by BIPNN increases sharply, leading to significantly higher computational costs. The results align with the theoretical analysis we presented in Sec. 5.

6.4 Ablation Study

GPU Acceleration. The superior time efficiency of BIPNN is primarily attributed to the GPU-accelerated algorithm employed in computing large-scale PUBO loss functions. Fig. 6 shows a comparison of the training times for BIPNN with or without the GPU-accelerated algorithm. We evaluate the training time of BIPNN on the hypergraph max-cut problem. The number of variables ranges from 200 to 1000. The degree of polynomial terms is set to 4. We train BIPNN for a fixed number of 1000 epochs. As Fig. 6 illustrates, when GPU acceleration is applied to compute the PUBO loss function, the training time does not exhibit significant growth with an increasing number of variables. In contrast, without GPU acceleration, the training time increases rapidly as the number of variables rises.

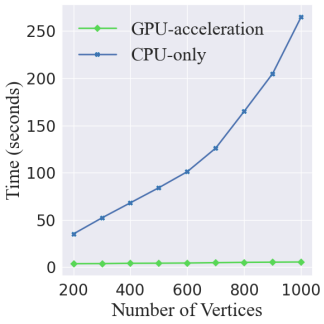


Figure 6: Comparison of the training time for BIPNN with or without GPU accelerated algorithm for PUBO losses.

Annealing Strategy. We validate the effectiveness of the annealing strategy of BIPNN on the hypergraph max-cut problem. The experiments are conducted on Cora with $1,330$ vertices. The metrics include the number of cuts and discreteness of variables. The penalty strength γ is set to -2.5 initially and its value is gradually increased during training. The value of γ reaches 0 after 500 epochs and continued to increase thereafter. As illustrated in Fig. 7, the annealing strategy ensures BIPNN to get better solutions while guaranteeing all variables to converge to discrete values. It demonstrates that negative γ values enable BIPNN to escape local optima, thereby discovering better solutions. Moreover, when γ is set to positive values, it facilitates the convergence of variables toward discrete values.

7 Conclusion

This work proposes BIPNN, a novel neural network solver for nonlinear BIP problems. It reformulates nonlinear BIPs into PUBO cost functions, which correspond to hypergraph structures. On this basis, these PUBO cost functions are used as loss functions for HyperGNNs, enabling the model to solve BIPs in an unsupervised training manner. Compared with existing BIP solvers (e.g., SCIP) that rely on linearization, BIPNN reduces the training cost by optimizing nonlinear BIPs via straightforward gradient descent. Empirical results demonstrate that BIPNN achieves state-of-the-art performance in learning approximate solutions for large-scale BIP problems.

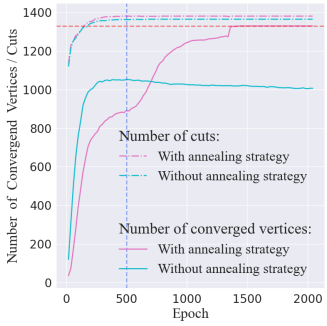


Figure 7: Quality and discreteness of solutions with or without the annealing strategy.

References

[1] Yan Qiao, Yanjun Lu, Jie Li, Siwei Zhang, Naiqi Wu, and Bin Liu. An efficient binary integer programming model for residency time-constrained cluster tools with chamber cleaning requirements. *IEEE Transactions on Automation Science and Engineering*, 19(3):1757–1771, 2021.

[2] Theodore P Papalexopoulos, Christian Tjandraatmadja, Ross Anderson, Juan Pablo Vielma, and David Belanger. Constrained discrete black-box optimization using mixed-integer programming. In *International Conference on Machine Learning*, pages 17295–17322. PMLR, 2022.

[3] Libin Wang, Han Hu, Qisen Shang, Haowei Zeng, and Qing Zhu. Structuredmesh: 3-d structured optimization of façade components on photogrammetric mesh models using binary integer programming. *IEEE Transactions on Geoscience and Remote Sensing*, 62:1–12, 2024.

[4] Giacomo Nannicini, Lev S Bishop, Oktay Günlük, and Petar Jurcevic. Optimal qubit assignment and routing via integer programming. *ACM Transactions on Quantum Computing*, 4(1):1–31, 2022.

[5] Akshay Ajagekar, Kumail Al Hamoud, and Fengqi You. Hybrid classical-quantum optimization techniques for solving mixed-integer programming problems in production scheduling. *IEEE Transactions on Quantum Engineering*, 3:1–16, 2022.

[6] Lei Fan and Zhu Han. Hybrid quantum-classical computing for future network optimization. *IEEE Network*, 36(5):72–76, 2022.

[7] Mercè Llabrés, Gabriel Riera, Francesc Rosselló, and Gabriel Valiente. Alignment of biological networks by integer linear programming: virus-host protein-protein interaction networks. *BMC bioinformatics*, 21(Suppl 6):434, 2020.

[8] Jianshen Zhu, Naveed Ahmed Azam, Fan Zhang, Aleksandar Shurbevski, Kazuya Haraguchi, Liang Zhao, Hiroshi Nagamochi, and Tatsuya Akutsu. A novel method for inferring chemical compounds with prescribed topological substructures based on integer programming. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19(6):3233–3245, 2021.

[9] Vladimir V Gusev, Duncan Adamson, Argyrios Deligkas, Dmytro Antypov, Christopher M Collins, Piotr Krysta, Igor Potapov, George R Darling, Matthew S Dyer, Paul Spirakis, et al. Optimality guarantees for crystal structure prediction. *Nature*, 619(7968):68–72, 2023.

[10] Georgia Stinchfield, Joshua C Morgan, Sakshi Naik, Lorenz T Biegler, John C Eslick, Clas Jacobson, David C Miller, John D Siirola, Miguel Zamarripa, Chen Zhang, et al. A mixed integer linear programming approach for the design of chemical process families. *Computers & Chemical Engineering*, 183:108620, 2024.

[11] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 2010.

[12] Elias B Khalil, Christopher Morris, and Andrea Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10219–10227, 2022.

[13] 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*, pages 39864–39878. PMLR, 2023.

[14] 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.

[15] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1:1–41, 2009.

[16] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[17] 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.

[18] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. *Advances in neural information processing systems*, 32, 2019.

[19] Jing Huang and Jie Yang. Unignn: a unified framework for graph and hypergraph neural networks. In *the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

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

[21] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[22] 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.

[23] Fred Glover and Manuel Laguna. *Tabu search*. Springer, 1998.

A A toy example of the polynomial reformulation of BIP (Sec. 4).

For $\sin(x_1 + x_2 + x_3)$, where $x_1, x_2, x_3 \in \{0, 1\}$, we can construct a polynomial to precisely fit the function, such that it matches $\sin(x_1 + x_2 + x_3)$ for all combinations of $x_1, x_2, x_3 \in \{0, 1\}$. For multiple binary variables, the polynomial can be generalized as:

$$P(x_1, x_2, x_3) = a_1x_1 + a_2x_2 + a_3x_3 + b_{12}x_1x_2 + b_{13}x_1x_3 + b_{23}x_2x_3 + cx_1x_2x_3 + d \tag{11}$$

Based on all possible combinations of x_1, x_2, x_3, we can set up the following equations:

1) When $x_1 = 0, x_2 = 0, x_3 = 0$: $P(0, 0, 0) = d = \sin(0) = 0$. Thus, $d = 0$.

2) When $x_1 = 0, x_2 = 0, x_3 = 1$: $P(0, 0, 1) = a_3 = \sin(1) \approx 0.8415$. Thus, $a_3 = 0.8415$.

3) When $x_1 = 0, x_2 = 1, x_3 = 0$: $P(0, 1, 0) = a_2 = \sin(1) \approx 0.8415$. Thus, $a_2 = 0.8415$.

4) When $x_1 = 1, x_2 = 0, x_3 = 0$: $P(1, 0, 0) = a_1 = \sin(1) \approx 0.8415$. Thus, $a_1 = 0.8415$.

5) When $x_1 = 0, x_2 = 1, x_3 = 1$: $P(0, 1, 1) = a_2 + a_3 + b_{23} = \sin(2) \approx 0.9093$.

Substituting $a_2 = 0.8415$ and $a_3 = 0.8415$: $b_{23} = -0.7737$.

6) When $x_1 = 1, x_2 = 0, x_3 = 1$: $P(1, 0, 1) = a_1 + a_3 + b_{13} = \sin(2) \approx 0.9093$

Substituting $a_1 = 0.8415$ and $a_3 = 0.8415$: $b_{13} = -0.7737$.

7) When $x_1 = 1, x_2 = 1, x_3 = 0$: $P(1, 1, 0) = a_1 + a_2 + b_{12} = \sin(2) \approx 0.9093$

Substituting $a_1 = 0.8415$ and $a_2 = 0.8415$: $b_{12} = -0.7737$

8) When $x_1 = 1, x_2 = 1, x_3 = 1$: $P(1, 1, 1) = a_1 + a_2 + a_3 + b_{12} + b_{13} + b_{23} + c = \sin(3) \approx 0.1411$.

Substituting known values: $c = -0.0623$.

Based on the above calculations, the polynomial is:

$$P(x_1, x_2, x_3) = 0.8415(x_1 + x_2 + x_3) - 0.7737(x_1x_2 + x_1x_3 + x_2x_3) - 0.0623x_1x_2x_3 \tag{12}$$

B A toy example of the unconstrained reformulation of BIP (Sec. 4).

For a nonlinear constraint with exponential term $g(\mathbf{x})$: $2x_1 + e^{x_2} + 3x_1x_3 \leq 5$, where $x_1, x_2, x_3 \in \{0, 1\}$, we can find the minimal violation subsets $\mathcal{V}$ based on all possible combinations of x_1, x_2, x_3.

1) When $x_1 = 0, x_2 = 0, x_3 = 0$: $g(\mathbf{x}) = 1 \leq 5$, feasible.

2) When $x_1 = 0, x_2 = 0, x_3 = 1$: $g(\mathbf{x}) = 1 \leq 5$, feasible.

3) When $x_1 = 0, x_2 = 1, x_3 = 0$: $g(\mathbf{x}) = e \leq 5$, feasible.

4) When $x_1 = 1, x_2 = 0, x_3 = 0$: $g(\mathbf{x}) = 3 \leq 5$, feasible.

5) When $x_1 = 0, x_2 = 1, x_3 = 1$: $g(\mathbf{x}) = e \leq 5$, feasible.

6) When $x_1 = 1, x_2 = 0, x_3 = 1$: $g(\mathbf{x}) = 6 \geq 5$, violation.

7) When $x_1 = 1, x_2 = 1, x_3 = 0$: $g(\mathbf{x}) = e + 2 \leq 5$, feasible.

8) When $x_1 = 1, x_2 = 1, x_3 = 1$: $g(\mathbf{x}) = 5 + e \geq 5$, violation (not minimal).

Identified minimal violation subsets: $\{x_1, x_3\}$. Thus,

$$P(\mathbf{x}) = \lambda(x_1x_3) \tag{13}$$

Final BIP objective:

$$O_{\text{BIP}} = f(\mathbf{x}) + \lambda(x_1x_3) \tag{14}$$

C The hypergraph max-cut problem.

The max-cut problem of a hypergraph $G = (V, E)$ involves partitioning the vertex set into two disjoint subsets such that the number of hyperedges crossing the partitioned blocks is maximized.

PUBO Form. The hypergraph max-cut problem on G can be formulated by optimizing a PUBO objective as follows:

$$\min \quad O_{\max-\text{cut}} = \sum_{e \in E} (1 - \prod_{i \in e} x_i - \prod_{i \in e} (1 - x_i)) \tag{15}$$

where $x_i \in \{0, 1\}$ are binary decision variables.

For a simple example illustrated in Fig. 3, the original hypergraph consists of three hyperedges: $\{x_1, x_2\}$, $\{x_3, x_4\}$, and $\{x_1, x_2, x_3\}$. Thus, the max-cut objective of G is to minimize $2x_1 + 2x_2 + 2x_3 + x_4 - 3x_1x_2 - x_1x_3 - x_2x_3 - 2x_3x_4$. BIPNN typically generates a new hypergraph structure with five hyperedges, $\{x_1, x_2\}$, $\{x_3, x_4\}$, $\{x_1, x_3\}$, and $\{x_2, x_3\}$, to solve this PUBO objective. we found that both hypergraphs can be utilized for HyperGNN training in BIPNN framework.

D Datasets.

Table 2: Summary statistics of five real-world graphs: the number of vertices $|V|$, the number of edges $|E|$. Three hypergraphs: the number of vertices $|V|$, the number of hyperedges $|E|$, the size of the hypergraph $\sum_{e \in E} |e|$.

Graphs	$\lvert V\rvert$	$\lvert E\rvert$	**Hypergraphs**	$\lvert V\rvert$	$\lvert E\rvert$	$\sum_{e \in E} \lvert e\rvert$
BAT	131	1,003	Primary	242	12,704	30,729
EAT	399	5,993	High	327	7,818	18,192
UAT	1,190	13,599	Cora	1,330	1,503	4,599
DBLP	2,591	3,528				
CiteSeer	3,279	4,552				