OPTIMIZING PSEUDO-BOOLEAN POLYNOMIALS USING HYPERGRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

The challenge of solving NP-hard combinatorial optimization problems with deep learning has attracted considerable interest. Modern graph neural networks are capable of efficiently solving problem instances in an unsupervised manner, however the graph structure limits the scope of their application. We present a novel approach, hereafter referred to as PB-HGNN, which employs unsupervised Hypergraph Neural Networks (HGNNs) to solve the polynomial unconstrained binary optimization (PUBO) problem. By representing the high-order terms of the pseudo-Boolean polynomials as hyperedges in a hypergraph, the HGNN is enable to capture intricate variable interdependencies beyond pairwise interactions. As a result, our framework provides the possibility of solving a wide range of discrete problems that have not previously been addressed by neural networks. We evaluate PB-HGNN on random higher-order pseudo-Boolean polynomials, including the Sherrington-Kirkpatrick model, and max-3-SAT instances. Our results show that PB-HGNN outperforms baselines on these problems and is capable of solving large-scale PUBO instances.

1 INTRODUCTION

A pseudo-Boolean function is a mapping $f : \mathbb{B}^n \to \mathbb{R}$, where $\mathbb{B} = \{0, 1\}$ is a Boolean domain. All pseudo-boolean functions can be uniquely represented as multilinear pseudo-Boolean polynomials Hammer & Rudeanu (1968); Boros & Hammer (2002) of the following form:

$$f(x_1, ..., x_n) = \sum_{S \subseteq [n]} c_S \prod_{j \in S} x_j, \;\; x \in \mathbb{B}^n. \tag{1}$$

The general problem to optimize (1) is known as Polynomial Unconstrained Binary Optimization (PUBO) problem. $deg(f) = \max_{c_S \neq 0} |S|$ is the degree of pseudo-boolean polynomial. $deg(f) = 2$ yields the well-known Quadratic Unconstrained Binary Optimization (QUBO) problem. Recently, there has been a resurgence of interest in encoding various Combinatorial Optimization (CO) problems as QUBOs Lucas (2014); Glover et al. (2022). This is largely due to the development of specialized computational devices such as quantum annealing Boixo et al. (2013) and coherent Ising machines Wang et al. (2013), which exploit the interaction between QUBO solutions and the fundamental states of physical systems of the Ising Hamiltonian Vadlamani et al. (2020). In the last few years, new heuristic algorithms based on unsupervised graph neural networks (GNNs) were proposed for solving CO problems formulated as QUBO Schuetz et al. (2022a); Pugacheva et al. (2024); Ichikawa (2024). They encode a problem's binary variables as graph nodes and its nonzero quadratic terms as edges, and then train a GNN to minimize the continuous relaxation of the QUBO objective.

In this paper, we address the higher-order PUBO problem, when $deg(f) \geq 3$. In combinatorial optimization, it has been widely applied for modeling a maximum satisfiability Christian et al. (2014), uncapacitated facility location Goldengorin et al. (2003); Beresnev (1973); Hammer (1968), and the p-median AlBdaiwi et al. (2009) problems. Other important applications occur in statistical mechanics Bernasconi (1987); Liers et al. (2010) , computer vision Ishikawa (2011); Fix et al. (2015), protein folding Babbush et al. (2014), etc. Classical approaches for optimizing higher-order f rely on reducing the polynomial degree through quadratization methods Verma et al. (2021);

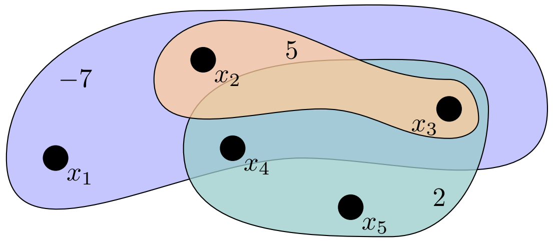


Figure 1: Hypergraph representation of $f(x_1, \ldots, x_5) = 5x_2x_3 + 2x_3x_4x_5 - 7x_1x_2x_3x_4$

Chancellor et al. (2016). By introducing additional variables, the problem can be reformulated as an integer linear program or a QUBO, which can then be solved using corresponding traditional algorithms.

While in quantum and quantum-inspired optimization it is now standard to solve PUBO *directly*—without reducing to QUBO—via higher-order Ising Hamiltonians Chermoshentsev et al. (2021); Stein et al. (2023); Kanao & Goto (2022), the neural combinatorial optimization literature still predominantly relies on explicit QUBO reductions or other objectives. In this work we introduce **PB-HGNN**, a hypergraph-based unsupervised framework that operates *natively* on higher-order pseudo-Boolean polynomials: each monomial x^S becomes a hyperedge on S with weight c_S, as shown in the fig. 1, and a lightweight residual hypergraph block produces instance-specific probabilities that are refined across epochs via an epoch-level memory. By staying in the original higher-order domain, PB-HGNN preserves multi-way interactions and simplifies the modeling pipeline.

We evaluate PB-HGNN on two representative benchmarks: random higher-order PUBO instances and MAX-3-SAT (via the standard PUBO mapping) and compare against standart algorithm GNN solvers, including QUBO-based that solve problem after the quadratization procedure. Across settings, PB-HGNN attains competitive or superior solution quality while remaining simple and scalable.

Contributions.

- Direct higher-order modeling for UL. We propose PB-HGNN, the first unsupervised neural framework that solves PUBO *without* quadratization, capturing k-way structure directly.
- Memory-driven architecture. We utlize a compact epoch-level memory that carries information across iterations, stabilizing instance-specific training with minimal overhead.
- Empirical validation. On random PUBO and MAX-3-SAT, PB-HGNN matches or surpasses quadratization-based GNN baselines.

2 COMBINATORIAL OPTIMIZATION WITH UNSUPERVISED LEARNING

Unsupervised Learning (UL). Neural approaches to CO include supervised prediction Prates et al. (2019); Gasse et al. (2019) and reinforcement learning (RL) Khalil et al. (2017); Kool et al. (2019); supervised methods require labeled near-optimal solutions and often struggle to generalize, whereas RL avoids labels but is often sample-inefficient on large state spaces. A complementary route is unsupervised, instance-specific learning: optimize a differentiable model *through* which the decision variables are produced, and update model parameters to improve the instance's objective, rather than updating the variables directly. This reparameterization often yields smoother search trajectories and fewer stalls in poor local minima Wang et al. (2022), and, by training on the instance itself, encodes useful inductive bias without requiring labeled solutions. In effect, UL turns the solver into a trainable generator of candidates that can be optimized end-to-end with standard gradient methods.

GNN-based UL Many CO problems are naturally defined on graphs, where solutions select subsets of nodes/edges; this makes graph neural networks appealing as the learnable maps that produce instance-specific decisions Cappart et al. (2021). Unsupervised GNN solvers include recurrent architectures for constrained satisfaction problems such as RUN-CSP Tönshoff J & M (2021); instance-specific training for SAT Amizadeh et al. (2018); and Erdős Goes Neural (EGN) Karalias & Loukas (2020), which learns distributions over solutions inspired by the probabilistic method, later extended via entry-wise concave relaxations Wang et al. (2022) and meta-learning variants. Min et al. (2023) train GNN with a surrogate unsupervised loss to produce an edge 'heat map', and a local search then decodes a TSP tour from it. Heydaribeni et al. (2024) first proposed using hypergraph neural networks for unsupervised combinatorial optimization. Their method, HypOp, models each constraint as a hyperedge in a constraint hypergraph and trains a HyperGNN as a learnable transformation function, with a simulated annealing step for mapping continuous relaxations back to discrete solutions. These methods differ in how they parametrize decisions, which unsupervised signals they optimize (constraint-violation penalties, differentiable surrogates), and how they control discretization (e.g., temperature schedules or annealing), but all share the goal of learning instance-specific heuristics directly from the objective without labeled solutions.

QUBO A prominent UL line reduces the target CO to QUBO, which is the problem to minimize a pseudo-Boolean polynomial 1 $f(x)$ of degree two Hammer (1968):

$$\min_{x\in\{0,1\}^n} x^T Qx, \tag{2}$$

PI-GNN Schuetz et al. (2022a) replaces binary variables by probabilities $p_\theta \in [0,1]^n$ output by a GNN and uses the relaxed QUBO as a differentiable loss, enabling general solvers whenever a QUBO formulation exists; follow-ups adapt this paradigm to specific tasks (e.g., graph coloring Schuetz et al. (2022b); Wang et al. (2023)). To direct relaxation toward discrete solutions while keeping a smooth landscape, Ichikawa (2024) proposes a principled annealing schedule by loss regularization, that first facilitates exploration and then promotes automatic rounding. Pugacheva et al. (2024) proposed the novel GNN architecture design that feeds model outputs back into the next forward pass: Apply previous node states as dynamic node features and concatenate them with static features across epochs, improving stability and convergence of the UL procedure. Our work follows this UL paradigm but extends it to *hypergraphs* to natively capture higher-order terms in the objective function.

3 Preliminaries: Hypergraph Neural Networks

A hypergraph $\mathcal{H} = (V, E)$ generalizes a graph by allowing each hyperedge $e \in E$ to connect an arbitrary subset of nodes $e \subseteq V$. This structure naturally captures higher-order relations that are not representable with simple pairwise edges and is widely used across machine learning and network science Kim et al. (2024); Estrada (2005). Let $|V| = n$ and $|E| = m$. We write the binary incidence matrix $A \in \{0,1\}^{n\times m}$ with $A_{i,e} = 1$ iff $i \in e$, and define node and hyperedge degrees by $d_v(i) = \sum_e A_{i,e}$ and $d_e(e) = \sum_i A_{i,e}$, respectively. When hyperedges carry attributes (e.g., weights, types), we denote them by a_e.

Message passing on hypergraphs. Hypergraph Neural Networks (HGNNs) extend GNN message passing by alternating *node→edge* and *edge→node* aggregations Feng et al. (2019). At layer ℓ, with node features $X^{(\ell)} \in \mathbb{R}^{n\times d_\ell}$, a general two-step update can be written as

$$\begin{aligned} &(\text{node}\to\text{edge}) \quad U_e^{(\ell)} \;=\; \mathrm{AGG}_e\big\{\, \phi_n^{(\ell)}(X_i^{(\ell)}) : i \in e \,\big\}, \\ &(\text{edge}\to\text{node}) \quad X_i^{(\ell+1)} \;=\; \sigma\Big(\psi_n^{(\ell)}\big(\bigoplus_{e\ni i} \phi_e^{(\ell)}(U_e^{(\ell)}, a_e)\big)\Big), \end{aligned}$$

where ϕ_n, ϕ_e, ψ_n are learnable maps, AGG_e and $\oplus$ are permutation-invariant set aggregators (e.g., mean/sum), and σ is an activation. This template covers a broad family of HGNNs: different normalizations, stabilizers (e.g., BatchNorm), and choices of aggregators/MLPs lead to different concrete architectures Feng et al. (2019); Kim et al. (2024).

A common instantiation (mean/sum). A widely used choice (which we adopt further) applies linear maps before each hop and uses mean on the node→edge step and sum on the edge→node step:

$$U_e^{(\ell)} \;=\; \frac{1}{|e|}\sum_{i\in e} W_1^{(\ell)} X_i^{(\ell)}, \qquad X_i^{(\ell+1)} \;=\; \sigma\Big(\mathrm{BN}\big(\sum_{e\ni i} W_2^{(\ell)} U_e^{(\ell)}\big)\Big),$$

with trainable $W_1^{(\ell)} \in \mathbb{R}^{d_\ell \times d_h}$, $W_2^{(\ell)} \in \mathbb{R}^{d_h \times d_{\ell+1}}$, and an optional BatchNorm ("BN") for stability. Hyperedge attributes can modulate messages, e.g., by multiplying each $U_e^{(\ell)}$ with a scalar weight w_e or with learned embeddings Dong et al. (2020); Chien et al. (2022). Stacking two such hops with a residual (skip) connection yields a residual hypergraph block:

$$\widehat{X}^{(\ell+1)} = \mathrm{BN}\big(\textstyle\sum_{e\ni i} W_2^{(\ell)} U_e^{(\ell)}\big), \quad X^{(\ell+1)} = \sigma\big(\widehat{X}^{(\ell+1)} + \mathrm{Skip}(X^{(\ell)})\big).$$

Alternative normalizations based on degree matrices (e.g., variants derived from D_v and D_e) are also common and can be plugged into either hop to control scale and improve conditioning Zhou et al. (2006). The generic template above is agnostic to these choices and encompasses classical HGNNs and related approximations.

4 PB-HGNN: Hypergraph Neural Networks for Pseudo-Boolean Optimization

In this section, we describe a detailed explanation of the proposed framework.

Hypergraph Construction. Given a pseudo-Boolean polynomial 1 of degree $d \geq 3$, we construct a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ represents the set of variables $\{x_1, x_2, \dots, x_n\}$ and $\mathcal{E} = \{e_S : S \subseteq \{1, 2, \dots, n\}, c_S \neq 0\}$ represents hyperedges, weighted by $w_S = c_S$, corresponding to polynomial terms $c_S \prod_{i \in S} x_i$. An illustrative example on a small pseudo-Boolean polynomial is shown in Figure 1. This representation preserves the exact structure of higher-order dependencies.

Continuous Relaxation. In the literature on unsupervised learning for QUBO (e.g., PI-GNN and related approaches), it is common to replace the binary decision variables $x_i \in \{0,1\}$ by continuous probabilities $p_i \in [0,1]$. A neural network is then trained to output these probabilities, which represent the likelihood that variable i is assigned the value 1. The resulting continuous objective is

$$\mathcal{L}(p) = \sum_{S\subseteq[n]} c_S \prod_{i\in S} p_i, \qquad p \in [0,1]^n. \tag{3}$$

While this relaxation is usually motivated from a machine learning perspective, it coincides exactly with the *multilinear extension* Calinescu et al. (2007) of the underlying pseudo-Boolean function. This connection provides both an algebraic and a probabilistic interpretation of $\mathcal{L}(p)$.

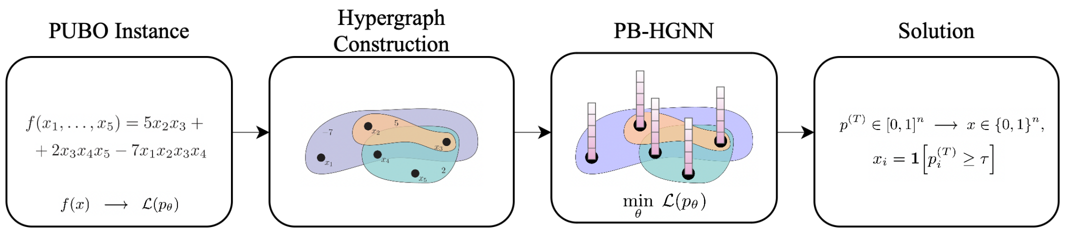
Lemma 1 (Multilinear extension of pseudo-Boolean polynomials). *Let $f : \{0,1\}^n \to \mathbb{R}$ be a pseudo-Boolean polynomial 1. Then its multilinear extension coincides with $\mathcal{L}(p)$ in equation 3.*

The multilinear extension admits a natural probabilistic view: if $X = (X_1, \dots, X_n)$ with $X_i \sim \mathrm{Bernoulli}(p_i)$ independently, then

$$\mathcal{L}(p) = \mathbb{E}[f(X)].$$

Hence the relaxed objective evaluates the expected value of the original discrete function under independent randomized rounding with marginals p. This interpretation is crucial in practice: optimizing $\mathcal{L}(p)$ corresponds to searching for probability distributions over feasible solutions that yield high-quality discrete outcomes upon sampling or rounding.

The continuous variables are parametrized by a hypergraph neural network: $p = \sigma(\mathcal{F}_\theta(\mathcal{H}, X^{(0)}))$ where σ is the sigmoid function, $\mathcal{F}_\theta$ is the HGNN with parameters θ, and $X^{(0)} \in \mathbb{R}^{n\times d_0}$ are node embeddings.



PUBO Instance
$f(x_1,\ldots,x_5) = 5x_2x_3 + 2x_3x_4x_5 - 7x_1x_2x_3x_4$
$f(x) \longrightarrow \mathcal{L}(p_\theta)$
Hypergraph Construction
PB-HGNN
$\min_\theta \mathcal{L}(p_\theta)$
Solution
$p^{(T)} \in [0,1]^n \longrightarrow x \in \{0,1\}^n,$
$x_i = \mathbf{1}\left[p_i^{(T)} \geq \tau\right]$

Figure 2: The PB-HGNN framework high-level design. (1) *PUBO Instance:* the input is a pseudo-Boolean polynomial and its probabilistic extension used for training. (2) *Hypergraph Construction:* each nonzero monomial becomes a hyperedge on its variables with weight equal to the coefficient. (3) *PB-HGNN:* a hypergraph neural network predicts probabilities by minimizing probabilistic extension. (4) *Solution:* After obtaining probabilities on the final iteration (epoch) *T*, a discrete assignment is obtained by thresholding.

Unsupervised Learning. The full PB-HGNN training framework is illustrated in Figure 3. Starting from the pseudo-Boolean polynomial, we construct the hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ as described above. Each node $v_i \in \mathcal{V}$ is initialized with an embedding $X_i^{(0)} \in \mathbb{R}^{d_0}$, drawn from a normalized random distribution or from problem-specific features if available. These embeddings are propagated through a stack of hypergraph convolutional and residual layers,

$$X^{(\ell+1)} = \phi_\ell\big(X^{(\ell)}, \mathcal{H}; \theta_\ell\big),$$

where ϕ_ℓ denotes the ℓ-th HGNN layer parameterized by θ_ℓ. After L layers, we obtain node-level hidden states $X^{(L)}$. A final projection layer followed by a sigmoid activation produces probabilities

$$p \;=\; \sigma\big(X^{(L)} W_{\text{out}} + \mathbf{1}\, b_{\text{out}}\big) \;\in [0,1]^n, \qquad X^{(L)} \in \mathbb{R}^{n \times d_L},\;\; W_{\text{out}} \in \mathbb{R}^{d_L \times 1},\;\; b_{\text{out}} \in \mathbb{R}.$$

These probabilities are interpreted as soft assignments of binary variables. The loss $\mathcal{L}(p)$ is computed directly from p and the polynomial coefficients, as detailed in the previous subsection.

Training proceeds in an unsupervised manner: given p, we evaluate the loss $\mathcal{L}(p)$, perform backpropagation through all layers of the network, and update the HGNN parameters $\theta = \{\theta_\ell, W_{\text{out}}, b_{\text{out}}\}$ using stochastic gradient descent (AdamW in our implementation). This iterative process continues until convergence or early stopping, yielding a trained PB-HGNN that maps a problem instance to a set of variable probabilities. After that, probabilities p are projected to a discrete solution by applying an indicator function,

$$x_i = \mathbf{1}\{\, p_i > \tau \,\}, \qquad \tau \in (0,1),$$

with $\tau = 0.5$ as the standard threshold. This simple rounding yields a binary assignment that can optionally be refined by local search.

4.1 PB-HGNN ARCHITECTURE

Memory-augmented hypergraph convolution. The performance of unsupervised neural solvers for combinatorial optimization depends critically on the underlying GNN architecture. In PB-HGNN we employ a memory-augmented hypergraph convolution block as the core unit. Alongside the static node embeddings $X^{(0)} \in \mathbb{R}^{n \times d_0}$, each node i carries a compact hidden memory $m_i^t \in \mathbb{R}^{d_m}$ ($d_m \ll d$), and we collect these row-wise into $M^t \in \mathbb{R}^{n \times d_m}$. At epoch t, the block takes $[\, X^{(0)} \,\|\, M^t \,]$ as input, performs hypergraph message passing with a residual (skip) connection, produces probabilities p^t, and updates the memory by projecting the current hidden states:

$$m_i^{t+1} \;=\; \tanh\big(W_m\, h_i^t\big), \qquad W_m \in \mathbb{R}^{d \times d_m}.$$

where $h_i^t \in \mathbb{R}^d$ denotes the block's output representation of node i at epoch t, and $W_m \in \mathbb{R}^{d \times d_m}$. We detach M^{t+1} from the computation graph before the next epoch, preventing backpropagation across epochs and keeping the per-epoch training cost stable. Thus the model refines its predictions over training iterations without recomputing from scratch.

Conceptually, our design is a lightweight variant of the iterative refinement strategy Pugacheva et al. (2024). In QI-GNN, refinement is performed across epochs by feeding the previous epoch's predicted probabilities back into the model as additional node features. PB-HGNN follows the same high-level idea to carry information across epochs, but does so via a compact hidden memory: instead of using raw predictions, we propagate a learned state M^t obtained from the current hidden representations and detached before the next epoch. This keeps the signal low-dimensional, reduces noise from raw outputs, and yields a simple, stable refinement mechanism embedded in the architecture.

Algorithm 1 PB-HGNN: one-epoch forward/backward with epoch-level memory

```
1:  **Input:** hypergraph $\mathcal{H} = (V, E)$; node features $X^{(0)} = \{x_i^{(0)}\}$; weights $\{w_e\}$; incidence $V(e), E(i)$; order emb. $\phi(\cdot)$; memory $M^t = \{m_i^t\}$
2:  **Output:** probabilities $P^t = \{p_i^t\}$; next memory $M^{t+1} = \{m_i^{t+1}\}$
3:  $h_i^{0,t} \leftarrow [\,x_i^{(0)} \,\|\, m_i^t\,]$;   $z_i^t \leftarrow \mathrm{ReLU}(W_{\mathrm{in}} h_i^{0,t})$        ▷ input fusion & projection
4:  $u_e^t \leftarrow \frac{1}{|e|} \sum_{i \in V(e)} W_1\, z_i^t$        ▷ node→edge: mean aggregation
5:  $\tilde{a}_i^t \leftarrow \sum_{e \in E(i)} \big[ W_2(u_e^t \odot \phi(|e|)) \big] \cdot w_e$        ▷ edge→node: sum & modulation
6:  $a_i^t \leftarrow \mathrm{ReLU}\big(\mathrm{BN}_1(\tilde{a}_i^t)\big)$        ▷ norm & activation
7:  $u_e'^{\,t} \leftarrow \frac{1}{|e|} \sum_{i \in V(e)} W_1'\, a_i^t$        ▷ second pass: node→edge
8:  $\tilde{b}_i^t \leftarrow \sum_{e \in E(i)} \big[ W_2'(u_e'^{\,t} \odot \phi(|e|)) \big] \cdot w_e$        ▷ second pass: edge→node
9:  $b_i^t \leftarrow \mathrm{BN}_2(\tilde{b}_i^t)$;   $h_i^t \leftarrow \mathrm{ReLU}\big(b_i^t + \mathrm{Skip}(z_i^t)\big)$        ▷ residual (skip)
10: $p_i^t \leftarrow \sigma\big(w_{\mathrm{out}}^\top h_i^t + b_{\mathrm{out}}\big)$        ▷ output head (sigmoid)
11: $\mathcal{L} \leftarrow \sum_S c_S \prod_{j \in S} p_j^t$        ▷ probabilistic relaxation loss
12: **update** $\Theta$ with AdamW using $\nabla_\Theta \mathcal{L}$        ▷ backward
13: $m_i^{t+1} \leftarrow \tanh(W_m h_i^t)$; **detach** all $m_i^{t+1}$        ▷ memory update for next epoch
14: **return** $P^t, M^{t+1}$
```

We use a single residual hypergraph–convolution block whose state is carried *across epochs* via a compact per-node memory. For selected epoch, we show Algorithm 1 with details of the forward/backword procedure. Alongside the static embeddings $X^{(0)} \in \mathbb{R}^{n \times d_0}$, each node i keeps a memory vector $m_i^t \in \mathbb{R}^{d_m}$ ($d_m \ll d$); we stack them row-wise as M^t for input fusion. In each epoch t, the block (i) concatenates features with memory and projects to the working width (line 3); (ii) performs a two-hop message pass with *mean* node→edge aggregation (line 4) and *sum* edge→node aggregation modulated by the hyperedge order embedding $\phi(|e|)$ and polynomial weight w_e (lines 5–6); (iii) repeats the two hops with independent parameters and applies BatchNorm and a residual (skip) connection (lines 7–9). The output head produces per-node probabilities via a sigmoid (line 10), and the loss is the multilinear extension of the original polynomial evaluated at these probabilities (line 11). We update parameters with AdamW (line 12), then update the per-node memories by a tanh projection of the current hidden states and *detach* them before the next epoch to avoid backpropagation through epochs (line 13). This realizes an iterative refinement mechanism across epochs through a lightweight, learned hidden state rather than by feeding back raw predictions.

Training details. By default, we use a single residual hypergraph–convolution block with BatchNorm and ReLU, a linear output head with a sigmoid, and a compact per-node memory of size $d_m = \max\{1, \lfloor d/8 \rfloor\}$, where d is the working hidden width. Node features $X^{(0)} \in \mathbb{R}^{n \times d_0}$ are *static*: unless problem-specific features are provided, we initialize them with ℓ_2-normalized Gaussian vectors. Hyperedges are stored in a sparse incidence format and carry (i) the polynomial weight $w_e = c_S$ and (ii) an order embedding $\phi(|e|)$; in our implementation we cap the order vocabulary at $|e| \leq 5$ for efficiency (higher orders are clamped). We optimize parameters θ with AdamW, apply gradient clipping (max-norm 0.5), and use a REDUCELRONPLATEAU scheduler; early stopping is triggered when the loss stabilizes. Mixed precision is enabled when available. The memory state is propagated *across epochs* and *detached* before the next epoch to avoid backpropagation through time. At inference, we decode by thresholding $x_i = \mathbf{1}\{p_i > \tau\}$ with default $\tau = 0.5$. For efficiency, we pre-compile loss terms by degree (vectorized for $d = 1$, batched for $d = 2$, tensorized for $d \geq 3$). While the architecture naturally supports deeper stacks and alternative memory updates, our experiments use this minimal configuration for speed and stability.

5 Numerical Experiments

5.1 Random Polynomials

We consider synthetic pseudo-Boolean objectives of fixed degree $k \geq 3$ with n binary variables. Following prior work Chermoshentsev et al. (2021), we generate three standard classes of random instances: (i) *Class I (dense ± 1):* each k-tuple coefficient $J_{i_1 \dots i_k}$ is sampled i.i.d. from $\{\pm 1\}$ with equal probability - it corresponds to the extended version of Sherrington–Kirkpatrick model. Chermoshentsev et al. (2021). (ii) *Class II (dense uniform):* $J_{i_1 \dots i_k} \sim \mathrm{Unif}[-1, 1]$ i.i.d.; (iii) *Class III (sparse):* start from Class II and independently drop each coefficient with probability 0.9 (set it to zero), producing a sparse higher-order interaction tensor.

Let $x \in \{0, 1\}^n$. A degree-k random polynomial reads

$$p_k(x) \;=\; \sum_{1 \leq i_1 < \cdots < i_k \leq n} J_{i_1 \dots i_k}\, x_{i_1} \cdots x_{i_k}, \tag{4}$$

with coefficients $J_{i_1 \dots i_k}$ drawn according to the chosen class. When needed, we also include lower-order terms to allow affine shifts:

$$p(x) \;=\; C_\emptyset + \sum_i C_i x_i + \sum_{i<j} C_{ij} x_i x_j \;+ \sum_{i_1 < \cdots < i_k} J_{i_1 \dots i_k}\, x_{i_1} \cdots x_{i_k}. \tag{5}$$

The optimization task is $\min_{x \in \{0,1\}^n} p(x)$ (or equivalently, maximization with flipped signs). This is a native PUBO instance with hyperedges $e_{\{i_1, \dots, i_k\}}$ weighted by $J_{i_1 \dots i_k}$. **?**

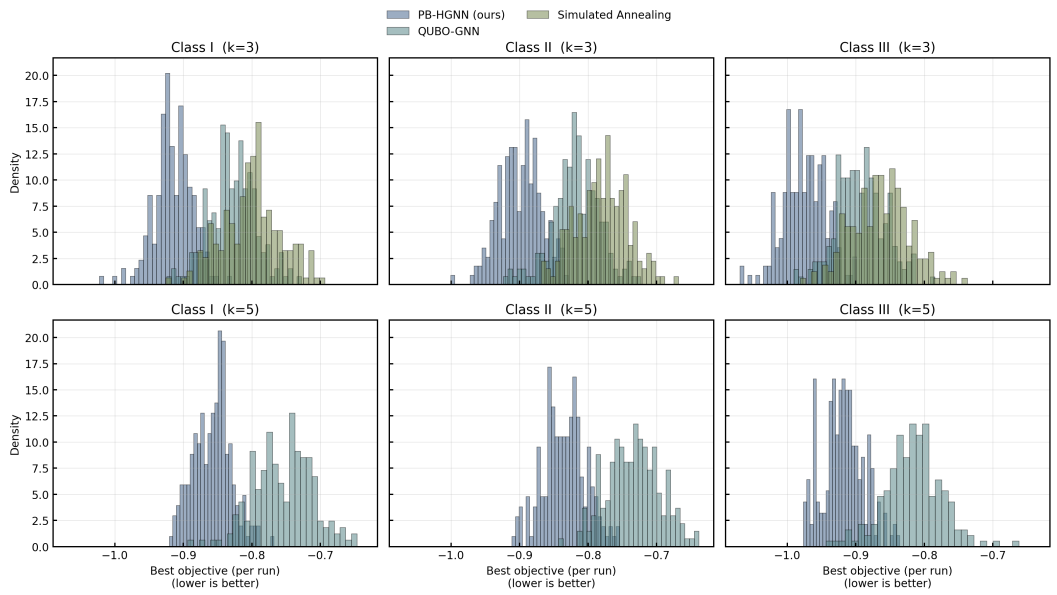


Figure 3: Outcome distributions at $k = \{3, 5\}$ (PB-HGNN vs. SA). Columns correspond to the three random PUBO classes, Each panel aggregates $R{=}100$ independent runs (different random initializations) and shows the density of the *normalized* best objective reached within the same compute budget. We omit QUBO GNNs for k = 5 since it requires very large number of variables (tens thousands) in the QUBO reformulation

Experimental Setup and Results. We study random PUBO instances at fixed size $n{=}100$ and degrees $k \in \{3, 5\}$ for the three classes, as defined above(. For each (k, class) we generate 10 independent instances and, unless noted, report aggregates across these instances. PB-HGNN is trained for 10,000 epochs with AdamW. We evaluate the discrete objective $p(x)$ each epoch, and keep the best-so-far value. As baselines, we use (i) a QUBO-GNN trained on a quadratized version of the same instances with the same epoch budget and optimizer, and (ii) Simulated Annealing (SA) on the PUBO form.

We visualize results with three figure sets. We show *Outcome distributions* 3 (2×3 grid): for each panel we aggregate the final best-of-run objectives over $R{=}100$ runs (multiple random initializations) and display density histograms for PB-HGNN, QUBO-based GNN (that solves the QUBO problem after quadratization) : we select Pugacheva et al. (2024) as it acheives the best performance among QUBO GNNs, and Simulated Annealing, revealing both spread and tail behavior. To make values comparable across instances with different scales and sparsities, we report $\widetilde{p}(x) = p(x)/\|c\|_1$, where $\|c\|_1 = \sum_S |c_S|$ is the ℓ_1 norm of the PUBO coefficients for that instance; lower is better. This scale-free normalization removes instance-dependent magnitude effects (e.g., number of nonzeros, coefficient ranges), so the histograms reflect solver quality rather than raw objective scale. Across all classes, SA is competitive and lies close to PB-HGNN under this normalization, indicating that SA is not drastically worse at $k{=}5$. Across classes and both degrees, PB-HGNN consistently reaches lower objectives faster than the QUBO baselines and exhibits tighter outcome distributions; the overhead plots highlight the rapidly growing variable/term counts for higher k, underscoring the advantage of operating natively on the PUBO hypergraph.

5.2 MAX-3-SAT

Problem description. Given Boolean variables $y_1, \dots, y_n \in \{0,1\}$ and a 3-CNF formula with m clauses $C_r = (\ell_{r1} \vee \ell_{r2} \vee \ell_{r3})$, each literal ℓ_{rk} being either a variable $y_{i_{rk}}$ or its negation $\neg y_{i_{rk}}$, the goal in MAX-3-SAT is to assign truth values to the variables so as to maximize the number of satisfied clauses. A clause is satisfied if at least one of its three literals is true, and violated if all three are false. The canonical objective is therefore

$$\max_{y\in\{0,1\}^n} \sum_{r=1}^{m} \mathbf{1}\{C_r(y) = \text{True}\} \;=\; m - \min_{y\in\{0,1\}^n} \sum_{r=1}^{m} \mathbf{1}\{C_r(y) = \text{False}\}.$$

We consider the equivalent minimization of the number of violated clauses (right-hand side), which is convenient for our PUBO mapping. MAX-3-SAT is NP-hard and widely used as a benchmark for discrete optimization; instances are often specified by their clause density $\alpha = m/n$ and may be *weighted* (each clause has a weight) or *unweighted* (all weights = 1). Performance is typically reported as the fraction of satisfied clauses or the number of violations at the returned assignment.

PUBO formulation. Let $x_i \in \{0,1\}$ encode $y_i = \text{True} \iff x_i = 1$. For a literal ℓ on x_i, define its falsity factor

$$\bar{\ell}(x) = \begin{cases} 1 - x_i, & \ell = x_i, \\ x_i, & \ell = \neg x_i. \end{cases}$$

A 3-clause $C_r = (\ell_{r1} \vee \ell_{r2} \vee \ell_{r3})$ is violated iff all three literals are false, hence

$$v_r(x) = \prod_{k=1}^{3} \overline{\ell_{rk}}(x).$$

Thus MAX-3-SAT becomes the unconstrained pseudo-Boolean optimization

$$\min_{x\in\{0,1\}^n} p(x) = \sum_{r=1}^{m} v_r(x) \;=\; C_\emptyset + \sum_i C_i x_i + \sum_{i<j} -C_{ij} x_i x_j + \sum_{i<j<k} C_{ijk} x_i x_j x_k,$$

where the coefficients $\{C_\emptyset, C_i, C_{ij}, C_{ijk}\}$ follow from expanding the products and collecting like terms. (Full coefficient formulas and derivation are provided in Appendix B.)

Experimental Setup and Results. We evaluate PB-HGNN on the MAX-3-SAT problem following the experimental design of Yau et al. citeyau2024are. We run Two settings are considered: (i) random unweighted instances with $n = 100$ variables and clause density $\alpha = m/n \in$

Table 1: Average number of unsatisfied clauses for Max-3-SAT on random instances with $N = 100$ variables and $\alpha \in \{4.00, 4.15, 4.30\}$. Standard deviation is shown after $\pm$. In parentheses: average runtime per instance (seconds).

Method	$\alpha = 4.00$	$\alpha = 4.15$	$\alpha = 4.30$
WalkSAT	0.94±0.92	1.46±1.11	1.97±1.28
Survey Propagation	3.32±0.81	3.87±0.79	3.94±0.93
ErdősGNN	5.46±1.91	6.14±2.01	6.79±2.03
OptGNN	4.46±1.68	5.15±1.76	5.84±2.18
PB-HGNN (ours)	3.11±0.98	4.01±1.10	4.29±1.45

$\{4.00, 4.15, 4.30\}$, and (ii) benchmark instances from the UUF SATLIB dataset. In both cases, the objective is to minimize the number of violated clauses.

For the random instance setting, each formula is generated on the fly with uniformly random clauses, and results are averaged over multiple test instances. We directly replicate the baselines reported in Yau et al. (2024): *WalkSAT* (classic stochastic local search solver), *Survey Propagation* Braunstein et al. (2005), *ErdősGNN* Karalias & Loukas (2020), and unsupervised OptGNN Yau et al. (2024). We then add a row for our *PB-HGNN*. Performance is measured as the average number of unsatisfied clauses (lower is better). The results show that our method outperforms both neural solvers, while being inferiour to the best classical problem-specific method.

For the second setting, we evaluate on the UUF benchmark SAT instances from SATLIB, commonly used to test incomplete Max-SAT solvers. We select 100 unsatisfiable instances from uuf125.538.100 dataset, with problems of 100 variables and 538 clauses, and report the average fraction of satisfied clauses across instances. We compare with local search sat-specific heuristics WalkSAT and Heuristic Backbone Sampling (HBS), and neural baseline *ErdősGNN* Karalias & Loukas (2020).

Table 2: Performance on UUF benchmark instances from SATLIB. We report average clause satisfaction (higher is better). PB-HGNN results will be added in the last row.

Method	average, satisfied clauses
WalkSAT	536.42
HBS	536.11
ErdősGNN	528.75
PB-HGNN (ours)	534.30

6 Conclusions

We introduced PB-HGNN, an unsupervised hypergraph–neural framework that optimizes pseudo-Boolean polynomials directly in their native higher-order form. By mapping each nonzero monomial to a hyperedge with the corresponding coefficient, and training a memory-augmented HGNN to minimize the probabilistic relaxation, our method manages to capture multi-way variable interactions. Across representative benchmark families: random higher-order PUBO, including the Sherrington–Kirkpatrick spin model, and MAX-3-SAT, PB-HGNN delivers competitive or superior solution quality to GNN baselines built on reduced quadratic formulations. heuristics, while scaling to thousands of variables. These results demonstrate that hypergraph message passing paired with instance-specific unsupervised training is a novel and effective recipe for higher-order combinatorial optimization.

REFERENCES

Bader F. AlBdaiwi, Diptesh Ghosh, and Boris Goldengorin. Data aggregation for p-median problems. *Journal of Combinatorial Optimization*, 21(3):348–363, June 2009. ISSN 1573-2886. doi: 10.1007/s10878-009-9251-8.

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *International Conference on Learning Representations*, 2018. URL `https://api.semanticscholar.org/CorpusID:53544639`.

Ryan Babbush, Alejandro Perdomo-Ortiz, Bryan O'Gorman, William Macready, and Alan Aspuru-Guzik. Construction of energy functions for lattice heteropolymer models: Efficient encodings for constraint satisfaction programming and quantum annealing, April 2014. ISSN 1934-4791. URL `http://dx.doi.org/10.1002/9781118755815.ch05`.

Vladimir L. Beresnev. On a problem of mathematical standardization theory. *Upr Sist*, (5):11:43–54 (in Russian), 1973.

J. Bernasconi. Low autocorrelation binary sequences: statistical mechanics and configuration space analysis. *Journal de Physique*, 48(4):559–567, 1987. ISSN 0302-0738. doi: 10.1051/jphys:01987004804055900. URL `http://dx.doi.org/10.1051/jphys:01987004804055900`.

Sergio Boixo, Tameem Albash, Federico M. Spedalieri, Nicholas Chancellor, and Daniel A. Lidar. Experimental signature of programmable quantum annealing. *Nature Communications*, 4(1), June 2013. ISSN 2041-1723. doi: 10.1038/ncomms3067. URL `http://dx.doi.org/10.1038/ncomms3067`.

Endre Boros and Peter L. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1–3):155–225, November 2002. ISSN 0166-218X. doi: 10.1016/s0166-218x(01)00341-9.

A. Braunstein, M. Mézard, and R. Zecchina. Survey propagation: An algorithm for satisfiability. *Random Structures amp; Algorithms*, 27(2):201–226, March 2005. ISSN 1098-2418. doi: 10.1002/rsa.20057. URL `http://dx.doi.org/10.1002/rsa.20057`.

Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *IPCO*, pp. 182–196, 2007. URL `https://doi.org/10.1007/978-3-540-72792-7_15`.

Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 4348–4355, 8 2021. doi: 10.24963/ijcai.2021/595. URL `https://doi.org/10.24963/ijcai.2021/595`.

N. Chancellor, S. Zohren, P. A. Warburton, S. C. Benjamin, and S. Roberts. A direct mapping of max k-sat and high order parity checks to a chimera graph. *Scientific Reports*, 6(1), November 2016. ISSN 2045-2322. doi: 10.1038/srep37107. URL `http://dx.doi.org/10.1038/srep37107`.

Dmitry A. Chermoshentsev, Aleksei O. Malyshev, Mert Esencan, Egor S. Tiunov, Douglas Mendoza, Alán Aspuru-Guzik, Aleksey K. Fedorov, and Alexander I. Lvovsky. Polynomial unconstrained binary optimisation inspired by optical simulation. 6 2021.

Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are allset: A multiset function framework for hypergraph neural networks. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=hpBTIv2uy_E`.

Kofler Christian, Greistorfer Peter, Wang Haibo, and Kochenberger Gary. A penalty function approach to max 3-sat problems. *Karl-Franzens-University Graz*, 2014.

Yihe Dong, Will Sawin, and Yoshua Bengio. Hnhn: Hypergraph networks with hyperedge neurons. *ArXiv*, abs/2006.12278, 2020. URL `https://api.semanticscholar.org/CorpusID:219965680`.

Juan A. Estrada, Rodriguez-Velazquez. Complex networks as hypergraphs. 2005. doi: 10.48550/ARXIV.PHYSICS/0505137. URL https://arxiv.org/abs/physics/0505137.

Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33013558. URL https://doi.org/10.1609/aaai.v33i01.33013558.

Alexander Fix, Aritanan Gruber, Endre Boros, and Ramin Zabih. A hypergraph-based reduction for higher-order binary markov random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(7):1387–1395, July 2015. ISSN 2160-9292. doi: 10.1109/tpami.2014.2382109. URL http://dx.doi.org/10.1109/TPAMI.2014.2382109.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems 32*, 2019.

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, April 2022. ISSN 1572-9338. doi: 10.1007/s10479-022-04634-2.

Boris Goldengorin, Diptesh Ghosh, and Gerard Sierksma. Branch and peg algorithms for the simple plant location problem. *Computers amp; Operations Research*, 30(7):967–981, June 2003. ISSN 0305-0548. doi: 10.1016/s0305-0548(02)00049-7.

Peter L. Hammer. Plant location—a pseudo-boolean approach. *Isr J Technol*, pp. 6:330–332, 1968.

Peter L. Hammer and Sergiu Rudeanu. *Boolean Methods in Operations Research and Related Areas*. Springer Berlin Heidelberg, 1968. ISBN 9783642858239. doi: 10.1007/978-3-642-85823-9. URL http://dx.doi.org/10.1007/978-3-642-85823-9.

Nasimeh Heydaribeni, Xinrui Zhan, Ruisi Zhang, Tina Eliassi-Rad, and Farinaz Koushanfar. Distributed constrained combinatorial optimization leveraging hypergraph neural networks. *Nature Machine Intelligence*, 6(6):664–672, May 2024. ISSN 2522-5839. doi: 10.1038/s42256-024-00833-7. URL http://dx.doi.org/10.1038/s42256-024-00833-7.

Yuma Ichikawa. Controlling continuous relaxation for combinatorial optimization, 2024.

H Ishikawa. Transformation of general binary mrf minimization to the first-order case. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(6):1234–1249, June 2011. ISSN 2160-9292. doi: 10.1109/tpami.2010.91. URL http://dx.doi.org/10.1109/TPAMI.2010.91.

Taro Kanao and Hayato Goto. Simulated bifurcation for higher-order cost functions. *Applied Physics Express*, 16(1):014501, dec 2022. doi: 10.35848/1882-0786/acaba9. URL https://dx.doi.org/10.35848/1882-0786/acaba9.

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6659–6672. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/49f85a9ed090b20c8bed85a5923c669f-Paper.pdf.

Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 6348–6358, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/d9896106ca98d3d05b8cbdf4fd8b13a1-Abstract.html.

Sunwoo Kim, Soo Yong Lee, Yue Gao, Alessia Antelmi, Mirko Polato, and Kijung Shin. A survey on hypergraph neural networks: An in-depth and step-by-step guide. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 6534–6544, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671457. URL https://doi.org/10.1145/3637528.3671457.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByxBFsRqYm.

Frauke Liers, Enzo Marinari, Ulrike Pagacz, Federico Ricci-Tersenghi, and Vera Schmitz. A non-disordered glassy model with a tunable interaction range. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(05):L05003, May 2010. ISSN 1742-5468. doi: 10.1088/1742-5468/2010/05/l05003. URL http://dx.doi.org/10.1088/1742-5468/2010/05/L05003.

Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2, 2014. ISSN 2296-424X. doi: 10.3389/fphy.2014.00005.

Yimeng Min, Yiwei Bai, and Carla P Gomes. Unsupervised learning for solving the travelling salesman problem. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=lAEc7aIW20.

Marcelo Prates, Pedro H. C. Avelar, Henrique Lemos, Luis C. Lamb, and Moshe Y. Vardi. Learning to solve np-complete problems: A graph neural network for decision tsp. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4731–4738, Jul. 2019. doi: 10.1609/aaai.v33i01.33014731. URL https://ojs.aaai.org/index.php/AAAI/article/view/4399.

Daria Pugacheva, Yuriy Zotov, Andrei Ermakov, and Igor Lyskov. Unsupervised graph neural networks with recurrent features for solving combinatorial optimization problems, 2024. URL https://openreview.net/forum?id=9qtswuW5ux.

Martin J. A. Schuetz, J. Kyle Brubaker, and Helmut G. Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, April 2022a. doi: 10.1038/s42256-022-00468-6. URL https://doi.org/10.1038/s42256-022-00468-6.

Martin J. A. Schuetz, J. Kyle Brubaker, Zhihuai Zhu, and Helmut G. Katzgraber. Graph coloring with physics-inspired graph neural networks. *Phys. Rev. Res.*, 4:043131, Nov 2022b. doi: 10.1103/PhysRevResearch.4.043131. URL https://link.aps.org/doi/10.1103/PhysRevResearch.4.043131.

Jonas Stein, Farbod Chamanian, Maximilian Zorn, Jonas Nüßlein, Sebastian Zielinski, Michael Kölle, and Claudia Linnhoff-Popien. Evidence that pubo outperforms qubo when solving continuous optimization problems with the qaoa. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, pp. 2254–2262, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701207. doi: 10.1145/3583133.3596358. URL https://doi.org/10.1145/3583133.3596358.

Wolf H Tönshoff J, Ritzert M and Grohe M. Graph neural networks for maximum constraint satisfaction. *Frontiers in Artificial Intelligence*, 3, 2021. doi: 10.3389/frai.2020.580607. URL https://doi.org/10.3389/frai.2020.580607.

Sri Krishna Vadlamani, Tianyao Patrick Xiao, and Eli Yablonovitch. Physics successfully implements lagrange multiplier optimization. *Proceedings of the National Academy of Sciences*, 117(43):26639–26650, October 2020. ISSN 1091-6490. doi: 10.1073/pnas.2015192117. URL http://dx.doi.org/10.1073/pnas.2015192117.

Amit Verma, Mark Lewis, and Gary Kochenberger. Efficient qubo transformation for higher degree pseudo boolean functions, 2021. URL https://arxiv.org/abs/2107.11695.

Haoyu Peter Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. 2022. URL https://openreview.net/forum?id=HjNn9oD_v47.

Xiangyu Wang, Xueming Yan, and Yaochu Jin. A graph neural network with negative message passing for graph coloring, 2023.

Zhe Wang, Alireza Marandi, Kai Wen, Robert L. Byer, and Yoshihisa Yamamoto. Coherent ising machine based on degenerate optical parametric oscillators. *Phys. Rev. A*, 88:063853, Dec 2013. doi: 10.1103/PhysRevA.88.063853. URL https://link.aps.org/doi/10.1103/PhysRevA.88.063853.

Morris Yau, Nikolaos Karalias, Eric Hanqing Lu, Jessica Xu, and Stefanie Jegelka. Are graph neural networks optimal approximation algorithms? In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=SxRblm9aMs.

Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In B. Schölkopf, J. Platt, and T. Hoffman (eds.), *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. URL https://proceedings.neurips.cc/paper_files/paper/2006/file/dff8e9c2ac33381546d96deea9922999-Paper.pdf.

A PROOFS

Lemma 1. (Multilinear extension of pseudo-boolean polynomial). *Let $f : \{0,1\}^n \to \mathbb{R}$ be a pseudo-Boolean polynomial*

$$f(x) = \sum_{S\subseteq[n]} c_S \prod_{i\in S} x_i, \qquad x \in \{0,1\}^n. \tag{6}$$

Then its multilinear extension coincides with $\mathcal{L}(p)$ in equation 3, i.e.,

$$F(p) = \sum_{S\subseteq[n]} c_S \prod_{i\in S} p_i, \qquad p \in [0,1]^n.$$

Proof. By definition, the multilinear extension of a set function f is

$$F(p) = \sum_{S\subseteq[n]} f(S) \prod_{i\in S} p_i \prod_{i\notin S} (1-p_i).$$

Evaluating $f(S)$ from its polynomial form gives

$$f(S) = f(\mathbf{1}_S) = \sum_{T\subseteq[n]} c_T \prod_{j\in T} \mathbf{1}_S(j) = \sum_{T\subseteq S} c_T.$$

Substituting into $F(p)$ yields

$$F(p) = \sum_{S\subseteq[n]} \Big(\sum_{T\subseteq S} c_T\Big) \prod_{i\in S} p_i \prod_{i\notin S} (1-p_i).$$

Swapping the order of summation gives

$$F(p) = \sum_{T\subseteq[n]} c_T \sum_{S\supseteq T} \prod_{i\in S} p_i \prod_{i\notin S} (1-p_i).$$

For each fixed T, factor out $\prod_{i\in T} p_i$ and write $U = S\setminus T$:

$$\sum_{S\supseteq T} \prod_{i\in S} p_i \prod_{i\notin S} (1-p_i) = \Big(\prod_{i\in T} p_i\Big) \sum_{U\subseteq[n]\setminus T} \prod_{i\in U} p_i \prod_{i\notin U} (1-p_i).$$

The inner sum is the expansion of $\prod_{i\in[n]\setminus T}(p_i + (1-p_i)) = 1$. Hence

$$F(p) = \sum_{T\subseteq[n]} c_T \prod_{i\in T} p_i,$$

which equals $\mathcal{L}(p)$ in equation 3. □

A.1 Hyperparameter Settings

Default hyperparameters optimized through extensive experiments: embedding dimension $d = 16$, hidden dimension $d_h = 8$, memory dimension $d_m = 2$, number of layers $L = 2$, learning rate $\alpha = 10^{-3}$ with ReduceLROnPlateau scheduling, dropout rate $\rho = 0.05$, weight decay $\lambda_2 = 10^{-6}$.

A.2 Efficient Loss Computation

Naively evaluating equation 3 costs $O(n^d)$. We pre-compile terms by degree:

$$\mathcal{L}(p) = \sum_{d=0}^{D} \mathcal{C}[d](p), \tag{7}$$

where: - $\mathcal{C}[1](p) = \sum_i c_i p_i$ (vectorized), - $\mathcal{C}[2](p) = \sum_{i,j} c_{ij} p_i p_j$ (batched matrix ops), - $\mathcal{C}[d](p) = \sum_{S:|S|=d} c_S \prod_{i \in S} p_i$ for $d \geq 3$ (tensorized).

The hypergraph is stored in sparse index format

$$\mathcal{E}_{\text{sparse}} = \{(i, e) : i \in e\},$$

reducing complexity from $O(nm)$ to $O(|\mathcal{E}_{\text{sparse}}|)$.

B Derivation details for MAX 3 SAT PUBO

For a literal on variable x_i, let the negation bit $\alpha \in \{0, 1\}$ indicate its sign ($\alpha = 0$ for x_i, $\alpha = 1$ for $\neg x_i$). Define $s := 1 - 2\alpha \in \{+1, -1\}$. Then the falsity factor can be written as

$$\bar{\ell}(x) = (1 - \alpha) - s\, x_i.$$

For a clause $C_r = (\ell_{r1} \vee \ell_{r2} \vee \ell_{r3})$ on indices $I_r = \{i_{r1}, i_{r2}, i_{r3}\}$ with signs $\{\alpha_{r1}, \alpha_{r2}, \alpha_{r3}\}$ and $s_{rk} = 1 - 2\alpha_{rk}$, the violation indicator expands as

$$v_r(x) = \prod_{k=1}^{3} \big((1 - \alpha_{rk}) - s_{rk} x_{i_{rk}}\big) = c_{\emptyset}^{(r)} + \sum_{p \in I_r} c_p^{(r)} x_p + \sum_{\{p,q\} \subset I_r} c_{pq}^{(r)} x_p x_q + c_{pqr}^{(r)} x_p x_q x_r,$$

with coefficients

$$\begin{aligned}
c_{\emptyset}^{(r)} &= \prod_{k=1}^{3} (1 - \alpha_{rk}), \\
c_{i_{rk}}^{(r)} &= -\, s_{rk} \prod_{\ell \neq k} (1 - \alpha_{r\ell}), \\
c_{i_{rk} i_{r\ell}}^{(r)} &= s_{rk}\, s_{r\ell} \prod_{j \neq k,\ell} (1 - \alpha_{rj}), \\
c_{i_{r1} i_{r2} i_{r3}}^{(r)} &= -\, s_{r1} s_{r2} s_{r3}.
\end{aligned}$$

Summing over clauses yields

$$p(x) = \sum_{r=1}^{m} v_r(x) = C_{\emptyset} + \sum_i C_i x_i + \sum_{i<j} C_{ij} x_i x_j + \sum_{i<j<k} C_{ijk} x_i x_j x_k,$$

where $C_{\emptyset} = \sum_r c_{\emptyset}^{(r)}$, $C_i = \sum_r c_i^{(r)}$, $C_{ij} = \sum_r c_{ij}^{(r)}$, and $C_{ijk} = \sum_r c_{ijk}^{(r)}$. Equivalently, one may enumerate all 2^3 sign patterns of a clause and record the resulting monomials; both approaches are algebraically identical.