OPTIMIZING PSEUDO-BOOLEAN POLYNOMIALS USING HYPERGRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Unsupervised graph neural networks solve combinatorial optimization instances without labels, but a graph encodes only pairwise interactions. We present PB-HGNN, an unsupervised hypergraph neural solver for polynomial unconstrained binary optimization (PUBO). It couples an objective-centric hypergraph representation — every monomial, or every compact factor such as a cut indicator or a clause, is a hyperedge on its variables — with a training procedure built for that representation: recurrent logit feedback corrected for two-stage hypergraph aggregation (leave-one-out messages and noise-driven exploration), best-incumbent tracking, and exact evaluation of the loss on the factored objective without polynomial expansion. The representation alone is not sufficient: a feed-forward hypergraph solver on the same hypergraph falls behind classical metaheuristics at scale. Under equal wall-clock budgets, PB-HGNN outperforms simulated annealing, greedy local search and prior hypergraph-neural solvers (HypOp, BIPNN) on hypergraph Max-Cut with up to 10^6 variables and on five real hypergraphs, matches or beats them on minimum hitting set, is competitive on MAX-3-SAT, dominates quadratization-based GNNs on random higher-order PUBO, and handles instances whose PUBO expansion is infeasible.

1 INTRODUCTION

A pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$ has a unique multilinear polynomial representation (Hammer & Rudeanu, 1968; Boros & Hammer, 2002)

$$f(x) = \sum_{S \subseteq [n]} c_S \prod_{j \in S} x_j, \qquad x \in \{0,1\}^n,\ c_S \in \mathbb{R}. \tag{1}$$

Minimizing (1) is the Polynomial Unconstrained Binary Optimization (PUBO) problem; $\deg f = \max_{c_S \neq 0} |S|$, and $\deg f = 2$ is QUBO. QUBO encodings of combinatorial optimization (CO) have regained traction (Lucas, 2014; Glover et al., 2022) through quantum annealers (Boixo et al., 2013) and coherent Ising machines (Wang et al., 2013), and unsupervised graph neural networks (GNNs) have been proposed for QUBO-formulated CO (Schuetz et al., 2022a; Pugacheva et al., 2024; Ichikawa, 2024): variables become nodes, quadratic terms become edges, and the GNN is trained to minimize a continuous relaxation of the objective.

We address higher-order PUBO, $\deg f \geq 3$, which arises in maximum satisfiability (Christian et al., 2014), facility location (Goldengorin et al., 2003; Beresnev, 1973; Hammer, 1968), the p-median problem (AlBdaiwi et al., 2009), statistical mechanics (Bernasconi, 1987; Liers et al., 2010), computer vision (Ishikawa, 2011; Fix et al., 2015) and protein folding (Babbush et al., 2014). The classical route reduces the degree by quadratization (Verma et al., 2021; Chancellor et al., 2016), introducing auxiliary variables and then solving a QUBO or an ILP. Quantum and quantum-inspired solvers now optimize PUBO directly through higher-order Ising Hamiltonians (Chermoshentsev et al., 2021; Stein et al., 2023; Kanao & Goto, 2022); on the learning side, hypergraph neural networks (HGNNs) are the natural counterpart — a hyperedge carries a k-way interaction — and have recently entered CO, first to encode constraints (HypOp, Heydaribeni et al. 2024) and then polynomial objectives (BIPNN, Bai et al. 2025).

We present **PB-HGNN**, which pairs this hypergraph representation with a training method built for it. Each monomial x^S, or each compact objective factor such as a cut indicator or a clause,

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

becomes a hyperedge on S with weight c_S; a dual-aggregation HGNN outputs instance-specific probabilities, which are refined across epochs by feeding the previous epoch's logits back into the network as a recurrent feature. Our central observation is that the representation is the starting point, not the solution: the same hypergraph optimized by a feed-forward HGNN falls behind classical metaheuristics at scale, whereas the training mechanisms introduced here turn it into a solver that surpasses them under equal wall-clock budgets on instances with up to 10^6 variables.

Contributions.

- *Native higher-order modeling with factored hyperedges.* PB-HGNN solves PUBO without quadratization on a hypergraph built from the objective. Unlike constraint-centric HGNN solvers (HypOp) and unlike BIPNN, which shares the monomial-hyperedge construction but requires the expanded polynomial, PB-HGNN treats compact factors (cut indicators, clauses, covering penalties) as hyperedges and evaluates the loss on them exactly, so instances whose expansion is infeasible (one hyperedge of size 187 yields 2^{187} monomials) are handled without approximation.
- *Recurrent-feedback training on hypergraphs.* We lift the recurrent-feature mechanism of QRF-GNN (Pugacheva et al., 2024) to hypergraphs and show that the transfer is not mechanical: two-stage (node→edge→node) aggregation echoes a node's own recurrent signal back to it and averages out the stochastic exploration that dropout provides on graphs. Leave-one-out messages and scaled noise on the recurrent channel correct both; we verify the effect on the training dynamics and find it decisive at scale.
- *Empirical validation under equal wall-clock budgets.* On hypergraph Max-Cut with up to 10^6 variables and on five real hypergraphs PB-HGNN surpasses simulated annealing, greedy local search and the hypergraph solvers HypOp and BIPNN; on minimum hitting set it matches or beats them; it is competitive on MAX-3-SAT, dominates quadratization-based GNNs on random PUBO, and extends to instances whose PUBO expansion no solver can materialize.

2 Related Work

Neural CO methods are supervised (Prates et al., 2019; Gasse et al., 2019), reinforcement-based (Khalil et al., 2017; Kool et al., 2019), or unsupervised and instance-specific: a differentiable model that *produces* the decision variables is trained to improve the instance's objective, which smooths the search and avoids poor local minima (Wang et al., 2022). Unsupervised GNN solvers include RUN-CSP (Tönshoff J & M, 2021), instance-specific SAT training (Amizadeh et al., 2018), Erdős Goes Neural (Karalias & Loukas, 2020) and its extensions (Wang et al., 2022; Karalias et al., 2022), and heat-map decoding for TSP (Min et al., 2023). The QUBO line replaces variables by GNN-output probabilities and uses the relaxed quadratic as the loss: PI-GNN (Schuetz et al., 2022a) and task-specific variants (Schuetz et al., 2022b; Wang et al., 2023), annealed regularization for discreteness (Ichikawa, 2024), and QRF-GNN (Pugacheva et al., 2024), which feeds the previous epoch's outputs back as a recurrent feature. Higher-order objectives in this line are handled by quadratization.

Two HGNN solvers are closest to ours. HypOp (Heydaribeni et al., 2024) introduced HGNNs for unsupervised CO, modelling each *constraint* as a hyperedge and decoding by simulated annealing. BIPNN (Bai et al., 2025) moved the hypergraph to the objective: every monomial of a polynomial binary program is a hyperedge, and a feed-forward HGNN$^+$ is trained with an annealed discreteness penalty, with reported gains over Gurobi and SCIP on nonlinear instances with up to 2×10^4 variables. PB-HGNN builds on the same objective-centric representation and differs in what is done with it: factors are hyperedges without expansion, and the network is trained with recurrent logit feedback, leave-one-out messages, exploration noise and incumbent tracking. We include both as baselines and find that these differences, not the representation, decide solution quality at scale. An extended discussion is given in Appendix B.

3 PB-HGNN

Objective hypergraph. Given (1), we build $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ with one node per variable and one hyperedge e_S per nonzero monomial, weighted by $w_S = c_S$ (Figure 2). The same construction

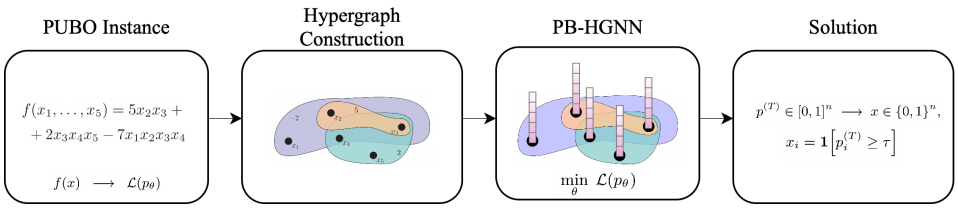

Figure 1: PB-HGNN. (1) A PUBO instance and its multilinear relaxation. (2) Hypergraph construction: each monomial or objective factor becomes a hyperedge weighted by its coefficient. (3) A hypergraph neural network is trained to minimize the relaxation. (4) At every epoch the probabilities are thresholded and scored exactly; the best discrete assignment over the run is returned.

underlies BIPNN (Bai et al., 2025). Many objectives arrive as sums of compact *factors* over small variable sets — cut indicators $1 - \prod_{i\in e} x_i - \prod_{i\in e}(1-x_i)$, clause violations $\prod_k \overline{\ell_k}(x)$, covering penalties $\prod_{i\in e}(1-x_i)$. Expanding a factor over t variables yields up to 2^t monomials that cancel catastrophically in floating point. PB-HGNN therefore treats factors as first-class: each factor is one hyperedge on its variable set with the factor weight as w_e, and the loss is evaluated on the factored expression, grouped by factor cardinality so that each group is one batched product. The two views coincide for a plain monomial list and are exact at binary points.

Relaxed loss and incumbent tracking. Following unsupervised QUBO solvers, binary variables are replaced by probabilities $p \in [0,1]^n$ and the loss is

$$\mathcal{L}(p) = \sum_{S\subseteq[n]} c_S \prod_{i\in S} p_i, \tag{2}$$

which is exactly the multilinear extension of f (Calinescu et al., 2007) (Lemma 3.1, proof in Appendix A): $\mathcal{L}(p) = \mathbb{E}[f(X)]$ for independent $X_i \sim \text{Bernoulli}(p_i)$, so minimizing $\mathcal{L}$ searches over product distributions whose samples have low expected cost.

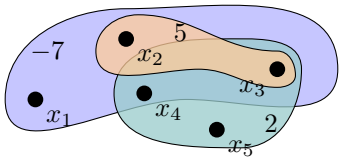

Figure 2: Objective hypergraph of $f = 5x_2x_3 + 2x_3x_4x_5 - 7x_1x_2x_3x_4$.

Lemma 3.1 (Multilinear extension)**.** *For f as in (1), the multilinear extension of f equals $\mathcal{L}(p)$ in (2).*

The probabilities are parametrized by an HGNN, $p = \sigma(\mathcal{F}_\theta(\mathcal{H}, X^{(0)}))$, trained by Adam on $\mathcal{L}$ (Figure 1). Discretization is interleaved with training: at every epoch $x^t = \mathbf{1}\{p^t > \frac{1}{2}\}$ is scored with the exact objective (one loss evaluation, since (2) is exact at binary points) and the best assignment seen is kept. This matters because the training dynamics below are deliberately noisy: the iterate visits many basins, and the best discrete solution is typically not the final one.

Hypergraph convolution. HGNNs alternate node→edge and edge→node aggregation (Feng et al., 2019; Kim et al., 2024). We use linear maps before each hop, mean aggregation on both, a degree-normalized edge→node step and a self (root) transform as in GraphSAGE (Hamilton et al., 2017):

$$U_e = \frac{1}{|e|}\sum_{i\in e} W_1 h_i, \qquad h_i' = \sigma\Big(\mathrm{BN}\big(\tfrac{1}{d_v(i)}\sum_{e\ni i} W_2 U_e + R\,h_i\big)\Big), \tag{3}$$

with $d_v(i)$ the node degree. Both details are load-bearing: node degrees of objective hypergraphs are heavy-tailed, so an unnormalized sum saturates the output sigmoid, and without Rh_i a mean-aggregated readout smooths toward a constant (Appendix C).

3.1 Architecture: Recurrent Feedback on Hypergraphs

Recurrent logit feedback. Following QRF-GNN (Pugacheva et al., 2024), each node receives, alongside its static input $X_i^{(0)}$, the detached pre-sigmoid logit it produced in the previous epoch, $h_i^{0,t} = [X_i^{(0)} \,\|\, z_i^{t-1}]$ with $z^0 = \mathbf{0}$. No gradient flows across epochs, so the per-epoch cost is

constant. The logit is unbounded, so it keeps the model's confidence after the sigmoid saturates, and it is fed back raw: a learned projection ($\tanh(W_m h^{t-1})$, used in an earlier version) saturates early and degenerates to a frozen sign latch. The purpose of the channel is to let every node see the current states of its *neighbors* and adapt to the evolving candidate solution. Two properties of hypergraph message passing break this, and we correct both.

Leave-one-out (LOO) messages. Graph convolutions aggregate over neighbors $j \neq i$. In the two-stage hypergraph pass node i belongs to every incident hyperedge, so each aggregate U_e contains i's own state — a fraction $1/|e|$ of the message, one half for $|e| = 2$. We compute the mean-branch edge→node message in leave-one-out form: with $\mu_j = W_1 h_j$ and $S_e = \sum_{j \in e} \mu_j$,

$$t_{e \to i} = \frac{S_e - \mu_i}{|e| - 1}, \tag{4}$$

and singleton hyperedges send nothing. This restores neighbor-state semantics at the cost of a per-incidence transform, improves cut quality on dense real hypergraphs by itself, and is a precondition for the noise below to act as coordinated exploration rather than per-node jitter (Appendix E).

Noisy recurrent feedback. On graphs the exploration that makes the recurrent mechanism effective is driven by dropout: perturbed logits are fed back and the trajectory performs coordinated moves between near-optimal configurations. On hypergraphs the two-stage aggregation averages these perturbations over edge members and incident edges, so the effective per-node noise shrinks with degree: on a co-tagging hypergraph we measure 4–5 variable flips per 10 epochs, versus ~ 40 for QRF-GNN on Gset graphs of comparable size. We restore the amplitude by injecting noise into the recurrent channel,

$$\tilde{z}^{\,t-1} = z^{t-1} + \sigma \,\mathrm{std}(z^{t-1})\, \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I_n), \tag{5}$$

with one scalar σ (default 0.2). Scaling by the running standard deviation makes the kick track the saturation level: once the logits polarize, the noise flips borderline variables, and the network, reading the noisy neighbor states through LOO messages, responds with coordinated multi-variable moves. The walk amplitude rises to 100–130 flips per 10 epochs, every sampled epoch visits a distinct discrete solution, and excursions of several hundred variables from the incumbent are followed by returns to better optima (Appendix E). The noise is applied only in training and adds no parameters.

Dual-aggregation block and readout. The core block runs two convolutions of the form (3) in parallel, differing in the node→edge aggregator: a mean branch with LOO messages $t_{e \to i}$ and a max branch $U_e^{\max} = \max_{i \in e} W_1' \rho(h_i)$ with a small pre-aggregation MLP ρ. On the edge→node step each message is modulated by a learned order embedding $\phi(|e|)$ and scaled by w_e,

$$a_i^{\bullet} = \frac{1}{d_v(i)} \sum_{e \ni i} w_e \, W_2^{\bullet}\big(U_e^{\bullet} \odot \phi(|e|)\big) + R^{\bullet} h_i, \qquad \bullet \in \{\mathrm{mean}, \max\}, \tag{6}$$

$$h_i' = \mathrm{LeakyReLU}\big(\mathrm{BN}_{\mathrm{mean}}(a_i^{\mathrm{mean}}) + \mathrm{BN}_{\max}(a_i^{\max})\big).$$

The mean branch measures the average pull of a hyperedge's variables (the gradient direction of the product); the max branch detects whether any variable in the hyperedge is strongly decided, which determines whether a higher-order monomial is active. We stack $L = 2$ blocks; the readout applies dropout and one mean/mean convolution of width 1, producing z^t and $p^t = \sigma(z^t)$. Static inputs concatenate a random block $\mathrm{Unif}[0,1]^{n \times d}$, two constant blocks and the standardized stationary distribution of the hypergraph random walk (the hypergraph analogue of PageRank, computed in $O(\mathrm{nnz}(A))$ per power iteration); with the recurrent logit, $d_0 = 5d + 1$. Algorithm 1 lists one epoch.

Training as stochastic exploration. PB-HGNN is trained with dropout $\rho = 0.5$ before the readout, a constant learning rate and a long horizon (10^3–5×10^4 epochs, set by the evaluation protocol). With the noisy feedback, optimization is not a descent into one basin: each epoch kicks the configuration while the recurrent logits pull it back toward the incumbent region, producing a random walk over near-optimal discrete solutions. In traced runs the logits polarize within a few hundred epochs ($> 99\%$ of probabilities outside $[0.05, 0.95]$); afterwards consecutive sampled epochs differ

Algorithm 1 PB-HGNN: one training epoch with noisy recurrent logit feedback

Input: hypergraph $\mathcal{H} = (V, E)$, static features $X^{(0)}$, weights $\{w_e\}$, order embedding $\phi(\cdot)$, previous logits z^{t-1} (detached), noise scale σ, incumbent $(x^{\text{best}}, f^{\text{best}})$
Output: logits z^t, updated incumbent
$\tilde{z}^{\,t-1} \leftarrow z^{t-1} + \sigma\,\mathrm{std}(z^{t-1})\,\varepsilon, \quad \varepsilon \sim \mathcal{N}(0, I_n)$ ▷ feedback noise (training only)
$h_i \leftarrow [\, X_i^{(0)} \,\|\, \tilde{z}_i^{\,t-1} \,]$ ▷ recurrent input fusion
for $\ell = 1, \ldots, L$ **do** ▷ dual-aggregation blocks
  $t_{e\to i} \leftarrow \frac{1}{|e|-1} \sum_{j \in V(e),\, j \neq i} W_1^{\ell} h_j$; $U_e^{\max} \leftarrow \max_{i \in V(e)} W_1'^{\ell} \rho^{\ell}(h_i)$ ▷ node→edge, LOO mean
  $a_i^{\text{mean}} \leftarrow \frac{1}{d_v(i)} \sum_{e \in E(i)} w_e \big[W_2^{\text{mean}\,\ell} (t_{e\to i} \odot \phi(|e|)) \big] + R^{\text{mean}\,\ell} h_i$ ▷ edge→node
  $a_i^{\max} \leftarrow \frac{1}{d_v(i)} \sum_{e \in E(i)} w_e \big[W_2^{\max\,\ell} (U_e^{\max} \odot \phi(|e|)) \big] + R^{\max\,\ell} h_i$
  $h_i \leftarrow \mathrm{LeakyReLU}\big(\mathrm{BN}_{\text{mean}}^{\ell}(a_i^{\text{mean}}) + \mathrm{BN}_{\max}^{\ell}(a_i^{\max}) \big)$
end for
$z^t \leftarrow \mathrm{HyperConv}_{\text{out}}(\mathrm{Dropout}_{\rho}(h)); \quad p^t \leftarrow \sigma(z^t)$ ▷ convolutional readout
$\mathcal{L} \leftarrow \mathcal{L}(p^t)$ ▷ multilinear / factored relaxation loss
update Θ with Adam using clipped $\nabla_\Theta \mathcal{L}$ ▷ backward
$x^t \leftarrow \mathbf{1}\{p^t > \frac{1}{2}\}$; **if** $f(x^t) < f^{\text{best}}$ **then** $(x^{\text{best}}, f^{\text{best}}) \leftarrow (x^t, f(x^t))$ ▷ on-device incumbent
detach z^t for the next epoch
return $z^t, (x^{\text{best}}, f^{\text{best}})$

by $\sim 10^2$ variables while the visited assignments stay 1-flip locally optimal or near-optimal. Interleaved thresholding harvests every basin visited. Removing the feedback, the noise, the high dropout or the long horizon collapses the exploration and degrades solution quality; LOO is what makes the noise act as coordinated exploration on dense instances (Section 4.5). Defaults: $L = 2$, hidden width 50, $d = 10$ ($d_0 = 51$), order vocabulary $|e| \leq 5$, Adam with constant learning rate and gradient clipping, no early stopping (Appendix D).

4 Experiments

Equal wall-clock protocol. Unless stated otherwise, every method receives the same budget $B = 60\,\text{s}$ per instance on one GPU. Iteration counts are calibrated on-line: a short warm-up measures the per-sweep (SA) or per-epoch (neural) cost and the main run is sized to fill the budget; one-off `torch.compile` warm-up is excluded for all learned methods. Solvers: *SA-long* (one calibrated multi-chain anneal with vectorized $O(|e|)$ flip deltas), *SA-restarts* (best of repeated 1,000-sweep anneals), *Greedy+LS* (randomized greedy with local search, restarted until the budget is exhausted), our reimplementation of *HypOp* (Heydaribeni et al., 2024) with its original training/SA-fine-tuning split, our reimplementation of *BIPNN* (Bai et al., 2025) in its published configuration (Appendix D.5) decoded with a local-search pass (*BIPNN+LS*), and PB-HGNN, reported directly (*PB*) or with HypOp's training/SA-fine-tuning split (*+SAft*). PB-HGNN uses one fixed configuration throughout (Section 3.1; $\sigma = 0.2, 0.4$ on the synthetic family; hidden width 32 on the three largest real instances to fit GPU memory, Appendix D). Every reported value is recomputed from the returned assignment by the exact evaluator.

4.1 Hypergraph Max-Cut at scale

For $\mathcal{H} = (V, E, w)$, hypergraph Max-Cut maximizes the weight of hyperedges whose vertices straddle the partition,

$$\max_{x \in \{0,1\}^n} \sum_{e \in E} w_e \Big[1 - \textstyle\prod_{i \in e} x_i - \prod_{i \in e} (1 - x_i) \Big], \tag{7}$$

a PUBO of degree $\max_e |e|$; each cut indicator is one factor hyperedge.

Synthetic hypergraphs. We generate ER hypergraphs with $k = 3$, $m = 5n$ unit-weight hyperedges and n from 5×10^4 to 10^6 (5M hyperedges). Table 1 reports the mean cut ratio. PB-HGNN is best at every size, by $+0.40$ points over the best SA variant at $n = 5 \times 10^4$; the margin persists

Table 1: Hypergraph Max-Cut on ER hypergraphs ($k = 3$, $m = 5n$), 60 s per method. Mean cut ratio $\pm$ std over seeds (5 seeds for $n \in \{2 \times 10^5, 5 \times 10^5\}$, 2 otherwise).

n	Greedy+LS	SA-long	SA-restarts	HypOp	BIPNN+LS	PB-HGNN
5×10^4	0.7920±0.0004	0.9296±0.0001	0.9258±0.0002	0.9289±0.0001	0.8824±0.0030	**0.9336**±0.0004
2×10^5	0.7851±0.0011	0.9249±0.0002	0.9254±0.0001	0.9234±0.0002	0.7342±0.0061	**0.9330**±0.0004
5×10^5	0.7826±0.0003	0.9235±0.0001	0.9252±0.0000	0.9151±0.0001	0.7663±0.0036	**0.9305**±0.0000
10^6	0.7821±0.0002	0.9218±0.0001	0.9251±0.0000	0.8955±0.0020	0.7632±0.0013	**0.9289**±0.0004

Table 2: Max-Cut on real hypergraphs (Benson et al., 2018) (unique simplices of size 2–4, unit weights), 60 s per method. Mean cut ratio; 2 seeds (6 on tags-math-sx, where the SA/PB margin is smaller than the 2-seed spread).

Dataset	n	m	Greedy+LS	SA-long	SA-rest.	HypOp	BIPNN+LS	PB	PB+SAft
tags-math-sx	1,616	140,015	0.8242	0.8267	0.8260	0.8220	0.8218	0.8274	**0.8276**
threads-ask-ubuntu	88,921	114,482	0.7559	0.9193	0.9068	0.9330	0.9369	**0.9481**	0.9479
tags-stack-overflow	49,177	3,884,295	0.8640	0.8881	0.8906	0.8827	0.8516	**0.9064**	0.9060
coauth-MAG-Geology	754,196	663,195	0.7124	0.9724	0.9705	0.9720	0.3381	**0.9748**	0.9747
coauth-DBLP	1,433,148	1,780,066	0.6868	0.9575	0.9569	0.9545	0.4637	**0.9595**	0.9591

at $n = 10^6$, where HypOp falls behind by over 3 points. The scaling is complementary: the per-flip cost of SA grows with the incidence structure and its sweep count collapses at large n, whereas PB-HGNN updates all variables per epoch at a cost linear in the instance and retains thousands of epochs at $n = 10^6$; HypOp spends most of its budget on SA fine-tuning and inherits SA's scaling. The noisy feedback is worth $+0.05$ to $+0.4$ points over the noiseless variant. BIPNN shows that the shared representation is not sufficient: its wide feed-forward pipeline retains under 1,000 epochs in the budget and its final-epoch decoding tracks no incumbent, leaving it 5 points behind PB-HGNN at $n = 5 \times 10^4$ and below Greedy+LS at every larger size.

Real hypergraphs. We repeat the protocol on five hypergraphs from the Benson collection (Benson et al., 2018): co-authorship (coauth-DBLP, coauth-MAG-Geology), Q&A co-participation (threads-ask-ubuntu) and co-tagging (tags-math-sx, tags-stack-overflow), keeping the unique simplices of size 2–4 as unit-weight hyperedges, so every hyperedge is a monomial factor of degree at most 4. A PB-HGNN variant is best on all five instances (Table 2). On the four large instances the margin is wide: $+2.9$ points over the best SA variant on threads-ask-ubuntu, $+1.6$ on tags-stack-overflow, up to $+2.4$ over HypOp — heavy-tailed, irregular instances where flip-based search stalls within the budget while the relaxation keeps improving. The small dense instance (tags-math-sx, $n = 1{,}616$) is the most favourable for SA, which nearly exhausts the local-search landscape; there the feedback noise closes the gap: without it PB-HGNN trails SA-long by 0.3 points, with it PB-HGNN leads by 0.07 ($115{,}851 \pm 64$ vs. $115{,}749 \pm 42$ cut hyperedges, 6 seeds). BIPNN is competitive only on threads-ask-ubuntu (0.9369, above SA and HypOp, 1.1 points behind PB-HGNN); on the two co-authorship hypergraphs its annealed final iterate collapses (0.34, 0.46) although the best solution passed during training reaches ≈ 0.67 — the method keeps no incumbent.

Weighted and non-expandable instances. Two further Max-Cut studies are reported in Appendix F. On weighted random hypergraphs ($n = 1{,}000$, $k = 5$, three topologies, three weight distributions, m up to 5×10^4, i.e. ~ 1.07M monomials), PB-HGNN attains the highest cut ratio in every configuration and is 4–8$\times$ faster than SA and 4–6$\times$ faster than HypOp. On the NDC drug-substance hypergraph (Heydaribeni et al., 2024) ($n = 3{,}767$, $m = 29{,}810$, edge sizes 2–187) the largest hyperedge alone expands to 2^{187} monomials, infeasible for any solver that materializes the polynomial (QUBO reformulations, quantum annealing, BIPNN's expanded configuration); PB-HGNN evaluates the factored cut loss directly and obtains the largest cut, $\sim 14\times$ faster than HypOp and an order of magnitude faster than the SA baselines from the literature (Table 4; BIPNN runs on the original hypergraph with a matched 120 s budget and the same local-search polish).

Table 3: Weighted hypergraph Max-Cut on synthetic random hypergraphs ($n = 1{,}000$, $k = 5$). Each cell shows cut ratio $r(x)$ / runtime per instance, averaged over 20 instances.

Topology	Weights	m	Greedy (r / t)	SA (r / t)	HypOp (r / t)	PB-HGNN (r / t)
ER	Unif$[0,1]$	3,000	1.0000 / 15s	1.0000 / 57s	1.0000 / 95s	**1.0000** / 19s
ER	Unif$[0,1]$	10,000	0.9928 / 67s	0.9974 / 180s	0.9981 / 152s	**0.9988** / 29s
ER	Unif$[0,1]$	20,000	0.9826 / 68s	0.9856 / 211s	0.9858 / 187s	**0.9863** / 41s
ER	Unif$[0,1]$	50,000	0.9681 / 142s	0.9716 / 427s	0.9728 / 318s	**0.9754** / 51s
ER	$\lvert\mathcal{N}(0,1)\rvert$	20,000	0.9802 / 70s	0.9837 / 217s	0.9844 / 195s	**0.9851** / 42s
ER	power-law	20,000	0.9779 / 71s	0.9826 / 223s	0.9830 / 198s	**0.9842** / 43s
PP	Unif$[0,1]$	20,000	0.9882 / 67s	0.9921 / 208s	0.9930 / 192s	**0.9947** / 40s
PP	$\lvert\mathcal{N}(0,1)\rvert$	20,000	0.9854 / 69s	0.9907 / 214s	0.9914 / 197s	**0.9931** / 42s
PP	power-law	20,000	0.9831 / 70s	0.9892 / 221s	0.9901 / 200s	**0.9918** / 43s
Reg	Unif$[0,1]$	20,000	0.9791 / 66s	0.9823 / 206s	0.9831 / 190s	**0.9850** / 40s
Reg	$\lvert\mathcal{N}(0,1)\rvert$	20,000	0.9763 / 68s	0.9798 / 213s	0.9810 / 196s	**0.9826** / 41s
Reg	power-law	20,000	0.9742 / 69s	0.9784 / 219s	0.9796 / 199s	**0.9811** / 42s

Table 4: NDC drug-substance hypergraph Max-Cut ($n = 3{,}767$, $m = 29{,}810$, max edge size 187). Higher cut is better.

Method	Cut size ↑	Runtime
Greedy (50 restarts)	27,833	21s
Direct GD (500 epochs)	28,070	3.8s
SA (our implementation)	28,671	∼1,200s
SA (Heydaribeni et al., 2024)	28,472	3,616s
HypOp (Heydaribeni et al., 2024)	28,849	1,679s
BIPNN+LS (Bai et al., 2025)	28,452	120s
PB-HGNN (ours)	28,922	**123**s

4.2 Minimum hitting set on real hypergraphs

The minimum hitting set (hypergraph vertex cover) asks for the smallest $C \subseteq V$ intersecting every hyperedge. With $x_i = \mathbf{1}\{i \in C\}$ we minimize the penalized PUBO

$$\sum_{i \in V} x_i + \lambda \sum_{e \in E} \prod_{i \in e} (1 - x_i), \qquad \lambda = 2, \tag{8}$$

whose optimum is a minimum cover since removing a vertex is never worth uncovering an edge; each penalty is one factor hyperedge of degree ≤ 4 on the hypergraphs of Section 4.1. We add *GreedyCover*, the problem-specific set-cover heuristic (repeatedly add the vertex covering most uncovered edges), restarted with randomized tie-breaking until the budget is exhausted. Every method's output passes the same deterministic repair (local search, greedy repair to feasibility, local-search trim), so all reported covers are feasible.

Table 5 separates two comparisons. Against general-purpose solvers PB-HGNN is the strongest at scale: on tags-stack-overflow, coauth-MAG-Geology and coauth-DBLP it improves on SA-long by 0.4%, 3.2% and 3.7% and on HypOp by 5.1%, 2.4% and 3.5%; on the two remaining instances it matches SA within one cover element (threads-ask-ubuntu, 14,381 vs. 14,380) or 0.2% (tags-math-sx). Against the problem-specific GreedyCover it obtains a smaller cover on three of five datasets and stays within 0.5–0.7% on the other two. BIPNN, optimizing the same penalized PUBO, trails every other solver by 13–84%: without incumbent tracking or fine-tuning its annealed final iterate over-selects vertices, and the shared repair cannot trim the excess. A generic polynomial optimizer on the raw objective (8) thus matches, and at scale beats, both metaheuristics and the dedicated covering heuristic.

Table 5: Minimum hitting set on real hypergraphs, 60 s per method. Mean cover size over 2 seeds (lower is better). GreedyCover is problem-specific; all other baselines are general-purpose PUBO solvers.

Dataset	n	Greedy+LS	SA-long	SA-rest.	GreedyCover	HypOp	BIPNN	PB	PB+SAft
tags-math-sx	1,616	803.5	**691.0**	700.5	703.0	692.5	783.5	700.5	692.5
threads-ask-ubuntu	88,921	42,883	**14,380**	14,771	14,389.5	14,388.5	16,767	14,435	14,381
tags-stack-overflow	49,177	35,387	10,668	10,696	**10,553.5**	11,199	15,429	10,624.5	10,630.5
coauth-MAG-Geology	754,196	342,881.5	189,710.5	193,533.5	184,203	188,279.5	290,540.5	183,773.5	**183,695**
coauth-DBLP	1,433,148	670,346	313,021.5	314,519.5	**299,973**	312,427.5	555,544	302,365	301,523

Table 6: MAX-3-SAT. Left: random formulas, $n = 100$, mean $\pm$ std unsatisfied clauses over 20 instances (lower is better). Right: SATLIB `uuf125.538` ($n = 125$, $m = 538$, unsatisfiable), mean satisfied clauses (higher is better). Runtime per instance in parentheses. Literature baselines from Yau et al. (2024); PB-HGNN run by us on the same instances with matched wall-clock. Bold: best general-purpose (non-SAT-specific) solver.

Method	$\alpha = 4.00$	$\alpha = 4.15$	$\alpha = 4.30$	`uuf125.538`
WalkSAT	0.94±0.92 (0.01 s)	1.46±1.11 (0.01 s)	1.97±1.28 (0.01 s)	536.42
Survey Propagation	3.32±0.81 (0.001 s)	3.87±0.79 (0.001 s)	3.94±0.93 (0.001 s)	—
HBS	—	—	—	536.11
ErdősGNN	5.46±1.91 (0.01 s)	6.14±2.01 (0.01 s)	6.79±2.03 (0.01 s)	528.75
OptGNN	4.46±1.68 (0.01 s)	5.15±1.76 (0.01 s)	5.84±2.18 (0.01 s)	—
PB-HGNN (ours)	**1.40**±0.97 (27 s)	**1.40**±0.97 (28 s)	**2.10**±1.22 (28 s)	**535.35** (28 s)

4.3 MAX-3-SAT

For clauses $C_r = (\ell_{r1} \vee \ell_{r2} \vee \ell_{r3})$ we minimize the number of violated clauses $\sum_r \prod_{k=1}^{3} \overline{\ell_{rk}}(x)$, where $\bar{\ell}(x) = 1 - x_i$ for $\ell = x_i$ and x_i for $\ell = \neg x_i$; each clause is one factor hyperedge, and the expanded degree-3 PUBO is given in Appendix H. Following Yau et al. (2024) we use random formulas with $n = 100$ and clause density $\alpha \in \{4.00, 4.15, 4.30\}$ (20 instances each; PB-HGNN trained for 10^4 epochs) and the SATLIB family `uuf125.538` ($n = 125$, $m = 538$, unsatisfiable). Baseline numbers are from Yau et al. (2024). PB-HGNN cuts violations by 64–73% relative to the strongest neural baseline from the literature (OptGNN) at every density and beats Survey Propagation throughout (Table 6). Against WalkSAT, a SAT-specific local search on the symbolic formula, PB-HGNN is ahead at $\alpha = 4.15$ (1.40 vs. 1.46) and within a small constant factor elsewhere. On `uuf125.538` PB-HGNN satisfies 535.35 clauses on average (28 s per instance), 0.2% below WalkSAT (536.42) and HBS (536.11) and 6.6 clauses above ErdősGNN (528.75).

4.4 RANDOM POLYNOMIALS

Following Chermoshentsev et al. (2021) we use three classes of random degree-k objectives: dense ± 1 coefficients (the higher-order Sherrington-Kirkpatrick model), dense $\mathrm{Unif}[-1, 1]$, and a sparse variant (10% of the coefficients nonzero at $k = 3$, 1% at $k = 5$); $n = 100$ throughout except the dense $k = 5$ classes, which run at $n = 50$ (2.1M monomials). We use 10 instances and 3 seeds per class and 30 s per run. Figure 3 shows, per class, the distribution over runs of the relative gap to the best objective any method found on the instance. Against SA-long, Greedy+LS, HypOp, direct gradient descent on (2) and a QUBO-GNN on the quadratized objective (QRF-GNN, Pugacheva et al. 2024), PB-HGNN is within 0.1–0.4% of the best solution found at $k = 3$, ahead of HypOp on two of three classes, and far ahead of the quadratized GNN, whose relative gap is 84–96% because the auxiliary variables destroy the landscape; at $k = 5$ these small instances are exhausted by flip-based search, which dominates all learned solvers (Appendix G). The regime in which the learned solver pays off is the large, irregular one of Sections 4.1 and 4.2.

4.5 ABLATION OF THE RECURRENT MECHANISM

Table 7 isolates the components of Section 3.1 on two real hypergraphs of different shape (tags-math-sx: small and dense, mean degree ~ 260; threads-ask-ubuntu: large and sparse), at a fixed horizon

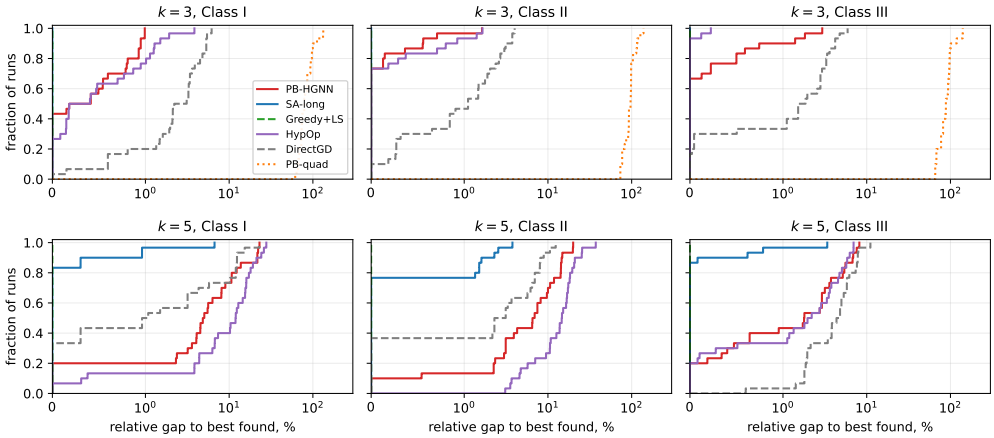

Figure 3: Random PUBO: empirical distribution of the per-run relative gap to the best objective found on the instance (lower-left is better; mass at zero is ties with the best known solution). $k = 3$ at $n = 100$ (top), $k = 5$ (bottom; dense classes at $n = 50$). PB-quad is omitted at $k = 5$.

of 8,000 epochs, 3 seeds. Each ingredient contributes and the gains compound: raw logit feedback over no feedback ($+0.34\%$ on tags-math-sx), LOO over plain aggregation ($+0.19\%$), noise over noiseless LOO ($+0.26\%$). The optimum of σ is flat and task-dependent (0.2 dense, 0.4–0.6 sparse), and every $\sigma \in [0.1, 0.6]$ improves on the noiseless baseline. LOO by itself helps on the dense instance and is neutral on the sparse one, where noise alone already reaches the best cut; on the dense instance LOO is required for the noise to help (115,504 without vs. 115,928 with): without it a node's own noisy echo dominates the message and the kick acts as per-node jitter. Alternatives we evaluated — hyperedge-state feedback, hyperedge dropout, stronger readout dropout, an EMA memory channel, feeding back the incumbent — are neutral or harmful (Appendix E, which also reports the traced dynamics: 7 flips per 10 epochs without feedback, 3–5 with plain logit feedback, 100–130 coordinated flips with noise).

5 Conclusions

PB-HGNN optimizes pseudo-Boolean polynomials in their native higher-order form by pairing an objective-centric hypergraph — each monomial or compact factor is a hyperedge with its coefficient — with a training method built for it: recurrent logit feedback with leave-one-out messages and scaled noise, best-incumbent tracking, and exact evaluation of the factored loss. Under equal wall-clock budgets it surpasses simulated annealing, greedy local search and the hypergraph solvers HypOp and BIPNN on hypergraph Max-Cut up to 10^6 variables and on five real hypergraphs, matches or beats them on minimum hitting set, is competitive on Max-3-SAT, dominates quadratization-based GNNs on random PUBO, and handles instances whose expansion is infeasible (max edge size 187). The comparison with BIPNN isolates the source of the gains: the same hypergraph optimized by a feed-forward network falls far behind, so the training mechanism, not the representation alone, is what makes native higher-order optimization effective at scale.

Limitations. On small dense instances ($n \leq 100$) flip-based local search exhausts the landscape within seconds and remains preferable; the learned solver pays off on large, irregular instances. Leave-one-out messages materialize per-incidence buffers, which forced a smaller hidden width on the three largest real hypergraphs. The noise scale σ has a flat but task-dependent optimum, and we used one value per benchmark family rather than per instance. Finally, PB-HGNN is an instance-specific optimizer: nothing is transferred between instances, and its per-epoch cost grows with the number of incidences, so the equal-budget comparison, not asymptotic optimality, is the relevant yardstick.

AI USE STATEMENT

In this work, generative AI tools were used to assist with code development and manuscript editing. All AI-assisted code was verified and tested for correctness by the authors, and all experimental results were produced and validated by the authors' own implementations. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The model architecture and training loop are specified in Section 3 and Algorithm 1, with the complete hyperparameter configuration in Appendix D and the by-degree loss compilation in Appendix D.4. The equal wall-clock benchmarking protocol, including baseline calibration, is described in Section 4. All datasets are public: SATLIB, the NDC hypergraph, and the Benson hypergraph collection (Benson et al., 2018); preprocessing (unique simplices of size 2–4) is stated where the datasets are introduced. Source code will be released as supplementary material.

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.

Sen Bai, Chunqi Yang, Xin Bai, Xin Zhang, and Zhengang Jiang. BIPNN: Learning to solve binary integer programming via hypergraph neural networks. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*, 2025.

Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. Simplicial closure and higher-order link prediction. *Proceedings of the National Academy of Sciences*, 115 (48):E11221–E11230, 2018.

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.

Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *Integer Programming and Combinatorial Optimization (IPCO 2007)*, volume 4513 of *Lecture Notes in Computer Science*, pp. 182–196. Springer, 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. *arXiv preprint arXiv:2106.13167*, 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.

Ernesto Estrada and Juan A. Rodríguez-Velázquez. Complex networks as hypergraphs. *arXiv preprint physics/0505137*, 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 & Operations Research*, 30(7):967–981, June 2003. ISSN 0305-0548. doi: 10.1016/s0305-0548(02)00049-7.

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

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.

Nikolaos Karalias, Joshua Robinson, Andreas Loukas, and Stefanie Jegelka. Neural set function extensions: Learning with discrete functions in high dimensions. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/6294a235c0b80f0a2b224375c546c750-Abstract-Conference.html.

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.

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. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 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 3.1. (Multilinear extension of pseudo-boolean polynomial). *Let* $f : \{0,1\}^n \to \mathbb{R}$ *be a pseudo-Boolean polynomial*

$$f(x) = \sum_{S \subseteq [n]} c_S \prod_{i \in S} x_i, \qquad x \in \{0,1\}^n. \tag{9}$$

Then its multilinear extension coincides with $\mathcal{L}(p)$ *in* (2), 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 (2). □

B Extended Related Work

Unsupervised learning for CO. Neural CO approaches include supervised prediction (Prates et al., 2019; Gasse et al., 2019), which requires labeled near-optimal solutions and often struggles to generalize, and reinforcement learning (Khalil et al., 2017; Kool et al., 2019), which is label-free but 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, updating its parameters to improve the instance's objective rather than the variables directly. This reparameterization yields smoother search trajectories and fewer stalls in poor local minima (Wang et al., 2022), and turns the solver into a trainable generator trained end-to-end without labels.

GNN-based unsupervised solvers. Many CO problems are defined on graphs, where solutions select subsets of nodes or edges, which makes graph neural networks natural learnable maps from instances to decisions (Cappart et al., 2021). Unsupervised GNN solvers include recurrent architectures for constraint satisfaction such as RUN-CSP (Tönshoff J & M, 2021), instance-specific training for SAT (Amizadeh et al., 2018), and Erdős Goes Neural (Karalias & Loukas, 2020), which learns distributions over solutions inspired by the probabilistic method, later extended via entry-wise concave relaxations (Wang et al., 2022), meta-learning variants, and high-dimensional set-function extensions (Karalias et al., 2022). Min et al. (2023) train a GNN with a surrogate unsupervised loss to produce an edge heat map from which local search decodes a TSP tour. Across this line, methods differ in parametrization, loss design (constraint-violation penalties, differentiable surrogates) and discretization strategy (temperature schedules, annealing), but share the goal of learning instance-specific heuristics from the objective alone.

QUBO-based unsupervised solvers. A prominent line reduces the target CO problem to QUBO, $\min_{x\in\{0,1\}^n} x^\top Q x$, the degree-two case of (1) (Hammer, 1968). PI-GNN (Schuetz et al., 2022a) replaces binary variables with GNN-output probabilities and uses the relaxed QUBO as a differentiable loss; follow-ups specialize the paradigm to graph coloring (Schuetz et al., 2022b; Wang et al., 2023). To bias the relaxation toward discrete solutions while keeping a smooth landscape, Ichikawa (2024) introduce an annealed regularizer, and QRF-GNN (Pugacheva et al., 2024) feeds the previous epoch's outputs back into the network as a recurrent feature, improving stability and convergence. PB-HGNN follows this paradigm but operates on hypergraphs, capturing higher-order terms natively instead of quadratizing them.

Hypergraph-neural solvers. HypOp (Heydaribeni et al., 2024) introduced hypergraph neural networks for unsupervised CO, modelling each constraint as a hyperedge and decoding the relaxation through simulated annealing; its hypergraph is constraint-centric, and message passing does not see the objective. BIPNN (Bai et al., 2025) maps every monomial of a polynomial binary program to a hyperedge, converts constraints and nonlinear terms into polynomial penalties, and trains a feed-forward HGNN$^+$ encoder with random input features and an annealed discreteness penalty $\gamma(t)\sum_i(1-(2p_i-1)^2)$; its loss materializes the dense incidence matrix of the expanded polynomial. PB-HGNN differs in three respects: factored objectives are hyperedges without expansion, with the loss evaluated exactly on the factors; the network carries a recurrent logit channel corrected for hypergraph aggregation (leave-one-out messages and scaled noise); and the best discrete assignment is tracked over the run rather than read from the final iterate. Appendix D.5 describes our BIPNN reimplementation.

C Hypergraph Neural Network Background

A hypergraph $\mathcal{H} = (V, E)$ generalizes a graph by allowing each hyperedge $e \in E$ to connect an arbitrary subset of nodes $e \subseteq V$; it captures higher-order relations that pairwise edges cannot (Kim et al., 2024; Estrada & Rodríguez-Velázquez, 2005). With $|V| = n$ and $|E| = m$, the incidence matrix is $A \in \{0,1\}^{n\times m}$ with $A_{i,e} = 1$ iff $i \in e$; node and hyperedge degrees are $d_v(i) = \sum_e A_{i,e}$ and $d_e(e) = \sum_i A_{i,e}$. Hyperedge attributes (weights, types) are denoted a_e.

Message passing. HGNNs extend GNN message passing by alternating node→edge and edge→node aggregation (Feng et al., 2019). At layer ℓ, with node features $X^{(\ell)} \in \mathbb{R}^{n\times d_\ell}$,

$$U_e^{(\ell)} = \mathrm{AGG}_e\big\{\phi_n^{(\ell)}(X_i^{(\ell)}) : i \in e\big\}, \qquad X_i^{(\ell+1)} = \sigma\Big(\psi_n^{(\ell)}\big(\bigoplus\nolimits_{e\ni i}\phi_e^{(\ell)}(U_e^{(\ell)}, a_e)\big)\Big),$$

where ϕ_n, ϕ_e, ψ_n are learnable maps, AGG_e and $\oplus$ are permutation-invariant aggregators (mean, max, sum) and σ is an activation. Different normalizations, stabilizers (BatchNorm) and aggregator choices give the concrete architectures in the literature (Feng et al., 2019; Kim et al., 2024).

The normalized mean/mean instantiation. PB-HGNN uses linear maps before each hop and mean aggregation on both steps, (3). Two details are load-bearing on realistic instances. First, the edge→node step must be normalized by the node degree $d_v(i)$: node degrees of objective hypergraphs are heavy-tailed, and an unnormalized sum produces activations that differ by orders of magnitude between nodes, saturating the output sigmoid and killing the gradient. Second, each hop keeps a direct self transform $R^{(\ell)}X_i^{(\ell)}$ as in GraphSAGE (Hamilton et al., 2017); without it a node learns about itself only through its hyperedges, and a mean-aggregated readout smooths toward a constant. Hyperedge attributes can modulate messages by a scalar weight w_e or by learned embeddings (Dong et al., 2020; Chien et al., 2022); degree-matrix normalizations derived from D_v and D_e are also common (Zhou et al., 2006).

D Training Details

D.1 Hyperparameter Settings

The default configuration, used unchanged across the large-scale experiments (Sections 4.1 and 4.2): static feature width $d = 10$, giving $d_0 = 5d + 1 = 51$ input channels (random block $n \times d$, two constant blocks $n \times 2d$, standardized hypergraph-PageRank block $n \times 2d$, one recurrent logit channel); hidden width 50; $L = 2$ dual-aggregation blocks; order-embedding vocabulary capped at $|e| \le 5$ (higher orders clamped); dropout $\rho = 0.5$ applied once before the convolutional readout; recurrent feedback of detached pre-sigmoid logits initialized at $z^0 = \mathbf{0}$, with leave-one-out edge→node messages on the mean branch and Gaussian feedback noise $\sigma = 0.2$ (0.4 on the synthetic Max-Cut family); decoding threshold $\tau = 0.5$. Leave-one-out messages materialize per-incidence buffers, so on the three largest real instances (tags-stack-overflow, coauth-MAG-Geology, coauth-DBLP) the hidden width is reduced to 32 to fit GPU memory.

D.2 Optimization and Training Loop

We optimize the parameters θ with Adam at a constant learning rate $\alpha = 1.4 \times 10^{-2}$ and apply gradient clipping with max-norm 2.0; no learning-rate schedule and no early stopping are used. The training horizon is set by the evaluation protocol: a fixed epoch count on the small benchmarks, or an on-line calibration that measures the per-epoch cost and sizes the run to a wall-clock budget on the large ones (typically 10^3–5×10^4 epochs). At every epoch the probabilities are thresholded at $\tau = 0.5$ and the discrete assignment is scored with the exact objective; the incumbent (best assignment so far) is tracked on-device to avoid host synchronization, and is the solution returned at the end of training. The recurrent state (previous hidden representations and logits) lives in preallocated buffers updated in place, which keeps tensor addresses stable and allows the whole training step to be captured with CUDA graphs (`torch.compile`, `reduce-overhead` mode); capture is disabled beyond $\sim 10^6$ nodes where it is unreliable, and those instances run eagerly. For efficiency, we pre-compile loss terms by degree (vectorized for $d = 1$, batched for $d = 2$, tensorized for $d \geq 3$); see Appendix D.4.

D.3 Compute Resources

All experiments were run on workstations equipped with a single consumer-grade GPU (≤ 32 GB VRAM) and a multi-core CPU. No data-center accelerators such as H100/A100 or distributed clusters were used. Per-instance wall-clock times are reported alongside each results table; the equal-budget studies use 60 s per method per instance. The total compute for the full study, including preliminary runs not reported in the paper, is on the order of a few hundred GPU-hours.

D.4 Efficient Loss Computation

Naively evaluating (2) costs $O(n^d)$. We pre-compile terms by degree:

$$\mathcal{L}(p) = \sum_{d=0}^{D} \mathcal{C}[d](p), \tag{10}$$

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 $O(|\mathcal{E}_{\text{sparse}}|)$.

For factored objectives (cut indicators, clause violations, covering penalties), the loss is evaluated directly on the factors, grouped by factor cardinality so that each group is one batched product over an index tensor. This avoids the $2^{|e|}$ monomial expansion entirely: a factor over t variables costs $O(t)$ per evaluation, and the expression remains exact at binary points, so the same routine scores discrete assignments.

D.5 The BIPNN Baseline

BIPNN (Bai et al., 2025) is an unsupervised HGNN-based PUBO solver that shares our monomial-hypergraph representation. We compare against a faithful sparse reimplementation of the released code: two HGNN$^+$ convolutions (Linear $\rightarrow$ BatchNorm $\rightarrow$ two-stage degree-normalized propagation $D_v^{-1} H W D_e^{-1} H^\top$, ReLU and dropout between layers) followed by a linear readout and sigmoid; random uniform input features; Adam with learning rate 10^{-4}, weight decay 5×10^{-8}, gradient clipping at 5.5; and the continuous-annealing penalty $\gamma(t) \sum_i \left(1 - (2p_i - 1)^2\right)$ with γ ramped linearly from -1, crossing zero at 62.5% of the epoch horizon (the schedule of their released max-cut experiment). The released loss materializes the dense $n \times$ terms incidence matrix, infeasible beyond toy sizes; our reimplementation evaluates the identical multilinear objective with the same sparse factored loss used by every other method, which changes nothing about the method. Following their published configuration, message passing runs on the hypergraph of *expanded* PUBO monomials with signed coefficients as hyperedge weights (their better-performing variant), falling back

to the original factor hypergraph where the expansion exceeds memory (instances with more than 10^6 factors); widths are 1024–1024–512 as published, reduced to 128–128–64 / 64–64–32 when the incidence buffers exceed GPU memory ($> 2\times10^6$ / $> 8\times10^6$ incidences). Decoding thresholds the final annealed output at 0.5, as in the released evaluation code. On their Cora hypergraph max-cut testbed the reimplementation reaches 1,385–1,386 of 1,413 hyperedges cut at their published 5,000-epoch setting, within 0.7% of the 1,394 reported in their paper. Under the equal-budget protocol we calibrate the epoch count from a short warm-up run exactly as for the other learned methods, and apply a local-search polish (reported as BIPNN+LS) or the shared repair pipeline (hitting set).

E Ablation of the Recurrent Mechanism

This appendix isolates the contribution of each component of the recurrent mechanism of Section 3.1 and documents the training dynamics that motivate it. All runs in Table 8 share one protocol: the default architecture (Appendix D.1), 8,000 training epochs, 3 seeds; we report the mean cut on two real hypergraphs of very different shape — tags-math-sx (small, dense, mean degree ~ 260) and threads-ask-ubuntu (large, sparse).

Table 7: Ablation of the recurrent mechanism on real hypergraph Max-Cut (8,000 epochs, mean cut over 3 seeds, higher is better). Full table in Table 8.

Variant	tags-math-sx	threads-ask-ubuntu
no feedback	115,021	108,263
previous logits (plain QRF-GNN transfer)	115,417	108,420
previous logits + LOO	115,631	108,401
previous logits + noise $\sigma = 0.4$, no LOO	115,504	**108,673**
previous logits + LOO + noise $\sigma = 0.2$	**115,928**	108,559
previous logits + LOO + noise $\sigma = 0.4$	115,732	108,645

Table 8: Ablation of the recurrent mechanism on real hypergraph Max-Cut (8,000 epochs, mean cut over 3 seeds, higher is better). Top block: memory representations. Middle block: alternative mechanisms we evaluated and rejected, all on top of logits + LOO. Bottom block: feedback-noise scale σ.

Variant	tags-math-sx	threads-ask-ubuntu
no feedback (`none`)	115,021	108,263
previous probabilities	115,251	108,142
learned memory $\tanh(W_m h)$	115,387	108,306
previous logits (QRF-GNN transfer)	115,417	108,420
previous logits + LOO	115,631	108,401
+ hyperedge-state feedback	115,550	108,395
+ hyperedge dropout 0.15	115,586	108,246
+ readout dropout $0.5 \to 0.65$	115,458	108,379
EMA (two-timescale) memory	115,340	108,290
best-incumbent feedback channel	115,350	108,259
logits + LOO + noise, $\sigma = 0.1$	115,766	108,413
logits + LOO + noise, $\sigma = 0.2$	**115,928**	108,559
logits + LOO + noise, $\sigma = 0.4$	115,732	108,645
logits + LOO + noise, $\sigma = 0.6$	115,329	108,642
logits + LOO + noise, $\sigma = 0.8$	115,238	108,569
logits + noise $\sigma = 0.4$, no LOO	115,504	**108,673**

Three observations. (i) Each ingredient of the final mechanism contributes: raw logit feedback over no feedback (+0.34% on tags-math-sx), LOO over plain aggregation (+0.19%), and feedback noise over noiseless LOO (+0.26%); the gains compound. (ii) The optimum of σ is flat and task-dependent (0.2 on the dense instance, 0.4–0.6 on the sparse one); every $\sigma \in [0.1, 0.6]$ improves on the noiseless baseline. On the dense instance LOO is required for the noise to help (115,504 without vs. 115,928 with): without LOO a node's own noisy echo dominates the message, and

the kick acts as per-node jitter rather than as neighbor-state exploration. (iii) The alternatives we evaluated — feeding back per-hyperedge satisfaction states, structural noise via hyperedge dropout, stronger readout dropout, an exponential-moving-average memory channel, and feeding back the best incumbent — are all neutral or harmful; the simple scaled Gaussian kick on the logit channel is the only variant that consistently improves both instances.

Training dynamics. Table 9 summarizes traced dynamics on tags-math-sx (assignments sampled every 10 epochs). The metrics quantify the exploration regime: flips per sample (Hamming distance between consecutive sampled assignments), mean Hamming distance to the best-so-far solution, and the fraction of sampled assignments that are distinct. Without feedback the iterate jitters near one configuration (7 flips per sample, all within ~ 40 variables of the incumbent) and plateaus early. Logit feedback slows the walk (3–5 flips) but couples the moves through message passing, which already improves quality. Feedback noise changes the regime qualitatively: the walk makes 100–130 coordinated flips per sample, every sample is a new discrete solution, excursions reach 700 variables from the incumbent — and the best cut found improves accordingly, in $3\times$ fewer epochs. This is the same exploration signature reported for QRF-GNN on graphs, which plain transfer of the mechanism to hypergraphs fails to reproduce.

Table 9: Training dynamics on tags-math-sx (sampled every 10 epochs; ranges over 2 seeds). The noiseless traces run 100k epochs, the noisy trace 30k.

Variant	best cut	flips/sample	Hamming to best	distinct samples
no feedback	114,911–115,029	7.4	27–43	99%
previous logits	115,481–115,632	3.5–4.2	200–216	62–70%
previous logits + LOO	115,711–115,757	4.5–5.2	68–198	84–87%
+ noise σ=0.2	**115,859–115,888**	102–134	248–711	100%

Task dependence. The same mechanism applied to the minimum hitting set (Section 4.2) tells a complementary story: there, plain logit feedback *hurts* (on threads-ask-ubuntu, cover 14,557 vs. 14,342 for no feedback at 8,000 epochs), because the penalized covering objective rewards convergent refinement rather than exploration. Feedback noise neutralizes the harm (14,353–14,390) and on the dense instance turns feedback slightly beneficial (697.7 vs. 700.3), making the noisy-feedback configuration a safe default across both task families.

F Additional Max-Cut Results

F.1 Weighted hypergraph Max-Cut on synthetic hypergraphs

We generate hypergraphs with $n = 1{,}000$ and order $k = 5$, varying topology (Erdős-Rényi (ER), planted-partition (PP) with two equal blocks, and k-regular), edge weights ($\mathrm{Unif}[0, 1]$, half-normal $|\mathcal{N}(0, 1)|$, power-law $w \propto u^{-2}$) and density $m \in \{3{,}000, \ldots, 50{,}000\}$ (up to $\sim$1.07M monomials). For each configuration we draw 20 instances and compare against Greedy (100 restarts), SA with vectorized $O(|e|)$ flip deltas, and HypOp (Heydaribeni et al., 2024). We report the cut ratio $r(x) = \sum_{e \text{ cut}} w_e / \sum_e w_e$ and runtime per instance (Table 3). PB-HGNN attains the highest cut ratio in every configuration; the gap over Greedy, SA and HypOp grows from a few thousandths in the saturated sparse regime ($m = 3{,}000$) to 0.3–0.4 points at $m = 50{,}000$, where multi-way couplings become more entangled and flip-based search increasingly stalls. PB-HGNN is also the fastest non-trivial solver: 4–8$\times$ faster than SA at the largest density and 4–6$\times$ faster than HypOp.

F.2 NDC drug-substance hypergraph

In the NDC hypergraph (Heydaribeni et al., 2024) each node is a substance and each hyperedge a drug formulation; Max-Cut partitions substances so as to maximize formulations spanning both groups. The instance has $n = 3{,}767$, $m = 29{,}810$ and edge sizes from 2 to 187 (median 3, mean 4.84). The largest hyperedge alone expands to $2^{187} \approx 10^{56}$ monomials, so no method that materializes the polynomial (QUBO reformulations, quantum annealing, BIPNN's expanded-hypergraph configuration) applies; PB-HGNN evaluates the factored cut loss directly and SA uses its $O(|e|)$ flip

deltas. Table 4 compares against SA and HypOp numbers from Heydaribeni et al. (2024), our own Greedy (50 restarts) and SA, direct gradient descent on the multilinear extension (500 epochs), and our BIPNN reimplementation run on the original hypergraph with a matched 120 s budget and the same local-search polish (its raw final decoding reaches only $\approx 21{,}400$ cut hyperedges). PB-HGNN obtains the largest cut, $\sim 14\times$ faster than HypOp and an order of magnitude faster than the SA baselines from the literature.

G Random Polynomials

Following Chermoshentsev et al. (2021), we consider three classes of random degree-k objectives. *Class I (dense ± 1)*: $J_{i_1 \dots i_k} \sim \mathrm{Unif}\{\pm 1\}$, the higher-order Sherrington-Kirkpatrick model. *Class II (dense uniform)*: $J_{i_1 \dots i_k} \sim \mathrm{Unif}[-1, 1]$. *Class III (sparse)*: Class II with each coefficient kept with probability 0.1 ($k = 3$) or 0.01 ($k = 5$). Including lower-order terms, the objective is

$$p(x) = C_\emptyset + \sum_i C_i x_i + \sum_{i<j} C_{ij} x_i x_j + \sum_{i_1 < \cdots < i_k} J_{i_1 \dots i_k}\, x_{i_1} \cdots x_{i_k}, \tag{11}$$

minimized over $x \in \{0, 1\}^n$; each monomial is a hyperedge with weight $J_{i_1 \dots i_k}$. We use $n = 100$ at $k = 3$ (161,700 degree-3 monomials in the dense classes) and for the sparse class at $k = 5$ (752,875 monomials); the dense $k = 5$ classes run at $n = 50$ (2,118,760 monomials), since dense degree-5 at $n = 100$ (75M monomials) is infeasible for every method. Each panel has 10 instances and 3 seeds per method, 30 s per run. Methods: SA-long, Greedy+LS, HypOp, direct gradient descent on the multilinear extension (DirectGD), PB-HGNN, and PB-quad — the same network trained on the Rosenberg-quadratized objective (QUBO-GNN; omitted at $k = 5$, where the quadratization exceeds memory).

Figure 3 shows, per panel, the empirical distribution of the relative gap $(v - v^\star)/|v^\star|$ between a run's objective v and the best objective $v^\star$ any method found on that instance; Table 10 lists the means. At $k = 3$ PB-HGNN is within 0.1–0.4% of the best known solutions, ahead of HypOp on Classes I and II, and 200–700$\times$ closer to the optimum than the quadratized network, whose auxiliary variables leave it 84–96% away. SA-long and Greedy+LS reach the best known solution in 80–100% of runs: at $n \le 100$ the landscape is small enough for flip-based search to be exhaustive, and at $k = 5$ this advantage widens to several percent over every learned solver. The random-polynomial family therefore isolates the benefit of native higher-order modeling over quadratization; the advantage over metaheuristics appears in the large, irregular regime of Sections 4.1 and 4.2.

Table 10: Random PUBO: mean relative gap to the best objective found on the instance (%, lower is better), 30 runs per cell.

k	Class	PB-HGNN	SA-long	Greedy+LS	HypOp	DirectGD	PB-quad
3	I	0.38	**0.00**	**0.00**	0.60	2.79	84.1
3	II	0.14	**0.00**	**0.00**	0.21	1.44	95.8
3	III	0.30	**0.00**	**0.00**	0.01	1.93	89.6
5	I (n=50)	7.24	0.31	**0.00**	11.51	4.49	—
5	II (n=50)	7.51	0.49	**0.00**	14.31	3.69	—
5	III	2.55	0.16	**0.00**	2.55	4.49	—

H 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

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

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.

I Higher-Order Sensor Placement

Problem description. Sensor placement is a classical combinatorial decision task in which one selects a subset of candidate locations to deploy sensors so as to maximize coverage of a set of targets. While the standard formulations are quadratic at most (utility plus pairwise overlap penalties), several practically important phenomena (triangulation, occlusion resolution, and fault-tolerant coverage) are inherently *higher-order*: their value emerges only when a specific group of sensors is jointly active. We therefore consider a higher-order variant of sensor placement that admits a natural pseudo-Boolean polynomial formulation:

$$f(x) \;=\; \sum_i a_i\,x_i \;-\; \sum_{i<j} b_{ij}\,x_i x_j \;+\; \sum_{S\in\mathcal{H}} \gamma_S \prod_{i\in S} x_i \;-\; \lambda\left(\sum_i x_i - B\right)^2, \tag{12}$$

where $x_i\in\{0,1\}$ indicates whether a sensor is deployed at candidate location i. The linear coefficients a_i encode individual sensor utility. Pairwise coefficients $b_{ij}\geq 0$ penalize redundant overlapping placements. The higher-order terms $\gamma_S>0$ reward groups of sensors whose joint deployment yields synergistic value (triangulation, fault tolerance, occlusion resolution). The last term is a soft budget penalty encouraging the solution to select approximately B sensors. The set $\mathcal{H}$ of higher-order interactions is determined by the spatial geometry of the instance.

For a medium-sized instance with $n=150$ candidate locations, $T=60$ targets, and budget $B=35$, the induced PUBO contains $14{,}090$ nonzero terms, including $2{,}173$ cubic and 592 quartic monomials. Such a formulation is well outside the comfort zone of QUBO solvers: a full quadratization of all higher-order terms would substantially blow up the variable count, which is why we benchmark only against PUBO-native baselines here.

Experimental setup and results. We compare PB-HGNN against three baselines that operate directly on the PUBO objective: (i) a random assignment, (ii) a greedy heuristic with multiple restarts, and (iii) Simulated Annealing on the discrete polynomial. For each method we report (a) the total objective $f(x)$, (b) the contribution of the higher-order synergy terms $\sum_{S\in\mathcal{H}}\gamma_S\prod_{i\in S}x_i$ (*Synergy*), which directly measures how well a method exploits the higher-order structure that motivates this benchmark, and (c) the budget violation $|\sum_i x_i - B|$.

PB-HGNN attains the strongest overall objective (413.51) while remaining within the budget. The improvement is most pronounced on the synergy component (456.31 vs. 433.00 for SA and 423.12 for Greedy), confirming that operating natively on the higher-order hypergraph allows the model to exploit cubic and quartic interactions that the baselines either do not see explicitly (Greedy) or can only access through expensive flip-evaluation (SA).

Table 11: Higher-order sensor placement on a medium instance ($n = 150$, $T = 60$, $B = 35$, with $14{,}090$ PUBO terms including $2{,}173$ cubic and 592 quartic monomials). Higher *Objective* and *Synergy* are better, while lower *Budget violation* is better.

Method	Objective ↑	Synergy ↑	Budget violation ↓
Random	134.92	97.05	0.00
Greedy	397.19	423.12	0.00
SA	399.83	433.00	0.00
PB-HGNN (ours)	**413.51**	**456.31**	0.00