ENHANCING GNNS PERFORMANCE ON COMBINATORIAL OPTIMIZATION BY RECURRENT FEATURE UPDATE

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNNs) can solve combinatorial optimization (CO) problems in an *instance-wise unsupervised* way: a fresh network is trained on one instance with a continuous relaxation of the QUBO objective as the loss, with no labels, no pre-training and no training distribution. This regime assumes nothing about the instance and yields a stand-alone heuristic, but it is also the hardest one for a network, and existing methods of this kind (PI-GNN, CRA) are beaten by simple classical heuristics. We show that one change to the training loop closes most of this gap. At every iteration, QRF-GNN feeds the network's own previous prediction back as a detached dynamic node feature, so each node adapts its decision to the current states of its neighbours; the input evolves together with the weights, and training becomes a guided search over binary configurations. We complement the mechanism with a two-branch GraphSAGE block and compact static features. On Max-Cut, Graph Coloring and Maximum Independent Set, QRF-GNN outperforms prior instance-wise unsupervised GNNs by a wide margin, matches or exceeds pre-trained neural solvers evaluated on their own training distributions, and approaches the best dedicated heuristics, on graphs with up to a million nodes. An analysis of the training dynamics shows that the recurrent feature turns a run that freezes early in one configuration into a walk over near-binary configurations that ends in a discrete local optimum.

1 INTRODUCTION

Combinatorial Optimization (CO) selects a combination of elements of a ground set that satisfies the problem's constraints and minimises a cost. Many CO problems are NP-hard, so exact algorithms apply only to small instances and heuristics that give high-quality solutions in reasonable time are essential (Boussaïd et al., 2013).

Many CO problems naturally arise on graph-structured data, where solutions correspond to selected subsets of nodes or edges. Implicit regularities and patterns often arise in graph structure and features, making the use of machine learning and especially graph neural networks (GNNs) very promising (Cappart et al., 2023). PI-GNN (Schuetz et al., 2022a;b; Wang et al., 2023) applies GNNs to CO problems with a Quadratic Unconstrained Binary Optimization (QUBO) formulation in an unsupervised manner, by minimising a differentiable continuous relaxation of the QUBO objective. No training data is used: a fresh GNN is trained end-to-end on each problem instance and acts as a stand-alone heuristic. We call this regime *instance-wise unsupervised* learning.

Learned CO solvers differ in what they know before solving an instance. Supervised, reinforcement-learning and amortised unsupervised solvers pre-train one model on a distribution of instances and are evaluated on new instances from the same distribution; their reported time is inference only, and their quality drops under distribution shift (Joshi et al., 2022; Nath & Kuhnle, 2025; Wu et al., 2026). Instance-wise solvers start from random weights on every instance, with no data and no prior, and their reported time is the whole solution process. This regime is the hardest for a network and the most general in practice, and until now it has also been the weakest: PI-GNN loses to simple greedy heuristics (Boettcher, 2023; Angelini & Ricci-Tersenghi, 2023), since the network settles in a poor configuration when trained per instance (Wang & Li, 2023), and CRA (Ichikawa, 2024) narrowed

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.

this gap with an annealed relaxation but kept the same training loop, in which the prediction is computed from fixed inputs.

We propose a framework for instance-wise unsupervised QUBO optimisation in which the GNN's previous prediction is fed back as a dynamic node feature at every iteration. We complement this *recurrent-feature mechanism* with a tailored architecture and a compact set of static node features, and call the resulting model **QRF-GNN** (QUBO-based GNN with Recurrent Feature). Our goal is to show that this single mechanism is enough to make an instance-wise unsupervised GNN competitive not only within its family, but with pre-trained neural solvers and dedicated heuristics. Our contributions are as follows:

- We introduce the *recurrent-feature mechanism* for instance-wise unsupervised QUBO training. It changes the training loop rather than the layers, costs one input column, and improves every GNN backbone we tested; on top of it we propose a two-branch hidden block with parallel mean and max GraphSAGE aggregation.
- We analyse how the mechanism changes training. Without it the run freezes early in one configuration; with it the network keeps moving between near-binary configurations, the relaxed loss stays within 0.01% of the discrete objective, and the returned solution is a discrete local optimum (Section 6, Appendix C).
- We benchmark QRF-GNN on Max-Cut, Graph Coloring and Maximum Independent Set against instance-wise unsupervised GNNs, pre-trained neural solvers and classical heuristics, stating the regime, protocol and source of every baseline. QRF-GNN is far ahead of prior instance-wise methods, matches or exceeds pre-trained solvers evaluated in-distribution, and approaches the best dedicated heuristics, with a runtime advantage on large graphs.

2 Related work

Graph neural networks are increasingly used for CO (Cappart et al., 2023). *Supervised* (Prates et al., 2019; Gasse et al., 2019; Li et al., 2018; Sun & Yang, 2023; Li et al., 2023) and *reinforcement-learning* (Khalil et al., 2017; Kool et al., 2019; Qiu et al., 2022; Darvariu et al., 2024; Ahn et al., 2020) approaches require labelled data or careful reward design and often struggle to generalise on large graphs.

Unsupervised learning bypasses ground-truth solutions and splits into two regimes by what the model knows about the test distribution at inference time. *Amortised* methods (Tönshoff J & M, 2021; Amizadeh et al., 2018; Karalias & Loukas, 2020; Sun et al., 2023b; Wang et al., 2022; Zhang et al., 2023; Sanokowski et al., 2024) pre-train one model on graphs from a target distribution and reuse it on new instances from the same distribution; like supervised and RL solvers, they thus rely on transferred weights and are constrained out of distribution. *Instance-wise* methods, in contrast, perform no pre-training and use no auxiliary data: a fresh GNN is trained end-to-end on each input graph with the QUBO objective itself as the loss. This is the line we extend, started by PI-GNN (Schuetz et al., 2022a), refined by CRA (Ichikawa, 2024) with a relaxation-annealing schedule, and ported to graph coloring as PI-GNN, GNN-1N and GDN (Schuetz et al., 2022b; Wang et al., 2023; Li et al., 2022); it provides the most direct learning-based baselines for QRF-GNN.

A parallel line couples GNNs with discrete search (X^2GNN (Acikalin et al., 2025)) or runs gradient-based MCMC on the QUBO relaxation (iSCO (Sun et al., 2023a), ReSCO (Li & Zhang, 2025), RLSA (Feng & Yang, 2025), and the MIS-specific pCQO-MIS (Alkhouri et al., 2025)). These methods need no pre-training but are orthogonal to our recurrent-feature update inside a single end-to-end gradient-trained GNN. Recent audits (Nath & Kuhnle, 2025; Wu et al., 2026) report that several methods labelled "unsupervised" in fact rely on distribution-specific pre-training and that classical heuristics often remain competitive once this is accounted for, motivating our evaluation against both classical and amortised learning baselines.

The recurrent feature also differs from other iterative schemes. Recurrent and implicit GNNs iterate a hidden state inside the forward pass with fixed weights; iterative refinement and learned local search apply a learned operator or a learned move policy to a candidate solution; test-time optimisation adapts a pre-trained model to a test input. In QRF-GNN there is no pre-trained model, no operator

on the solution and no unrolled state: the previous prediction enters only as a detached input feature, and the only update is a gradient step on the relaxed objective.

3 PRELIMINARIES

3.1 QUBO PROBLEM

One notable CO problem is the Quadratic Unconstrained Binary Optimization (QUBO), which seeks to minimize a pseudo-Boolean polynomial $\mathcal{F}(x)$ of degree two (Hammer & Rudeanu, 1969; Boros & Hammer, 1991):

$$\min_{x\in\{0,1\}^n} \mathcal{F}(x) = x^T Q\, x = \sum_{i=1}^{n}\sum_{j=1}^{n} Q_{ij}\, x_i x_j, \tag{1}$$

where $Q \in \mathbb{R}^{n\times n}$ is a symmetric matrix of pairwise coefficients and $x = (x_1, \ldots, x_n)^T$ is the vector of binary decision variables, $x_i \in \{0, 1\}$. Any linear contribution of the form $\sum_i c_i x_i$ can be absorbed into the diagonal of Q via the identity $x_i^2 = x_i$, so the canonical form $x^T Q\, x$ already captures both pairwise and linear coefficients.

Many CO problems admit a QUBO formulation through constraint penalisation (Glover et al., 2022; Lucas, 2014).

3.2 GRAPH NEURAL NETWORKS AND MESSAGE PASSING

Let us consider an undirected graph $G = (V, E)$ with a vertex set $V = \{1, \ldots, n\}$ and an edge set $E = \{(i, j) : i, j \in V\}$. Let $h_i^l \in \mathbb{R}^{m_l}$ denote a feature vector for node i at the l-th convolution layer, and let $e = (i, j)$ be an edge between nodes i and j.

In this paper, we consider GNNs based on the message-passing protocol, which consists of two main parts: message accumulation and message aggregation. Message accumulation computes a message m_e^{l+1} for an edge e via a function ϕ, which determines how information is collected. Message aggregation combines the collected messages from the neighbors of node i via a function ρ and then updates the feature vector h_i^l by applying a function f with trainable weights W^l:

$$\begin{aligned} m_e^{l+1} &= \phi\left(h_i^l, h_j^l\right), \\ h_i^{l+1} &= f\left(h_i^l,\, \rho\left(\{m_e^{l+1} : (i, j) \in E\}\right)\right). \end{aligned} \tag{2}$$

Input features h_i^0 can be one-hot labels, random or shared dummy vectors (Cui et al., 2022), PageRank (Brin & Page, 1998), degrees, or a trainable embedding (Schuetz et al., 2022a).

3.3 UNSUPERVISED, INSTANCE-WISE TRAINING VIA QUBO RELAXATION

To solve a CO problem, (Schuetz et al., 2022a) use a continuous relaxation of the QUBO formulation as a loss function for GNNs, introducing their physics-inspired GNN (PI-GNN). Let $\Phi_\theta : G \mapsto [0, 1]^n$ denote a GNN parameterised by θ, where θ collects the trainable parameters of ϕ, f and ρ. Applying a sigmoid (binary-decision problems) or softmax (multi-class problems) activation to the final embedding h^{N_l} yields the *relaxed* probability vector

$$p(\theta) \;\equiv\; \Phi_\theta(G) \;=\; \big(p_1(\theta), \ldots, p_n(\theta)\big) \;\in\; [0, 1]^n,$$

which we treat as a continuous surrogate for the binary decision vector $x \in \{0, 1\}^n$. Substituting it into the QUBO objective $\mathcal{F}(x) = x^\top Q\, x$ gives the differentiable loss

$$\mathcal{L}(\theta;\, G) \;=\; p(\theta)^\top\, Q\, p(\theta), \tag{3}$$

which is minimised end-to-end by gradient descent without ground-truth labels. The training is *instance-wise*: the GNN is reinitialised and optimised from scratch for each input graph G, in contrast to amortised approaches that train a single model on a distribution of graphs.

To recover a solution of the original discrete problem, each $p_i(\theta)$ is converted into the binary variable x_i. The simplest approach, which we use, is to apply an indicator function $\mathbb{I}_{p_i > p^*}$ with a threshold p^*. Alternatively, sampling from a Bernoulli distribution or greedy decoding may be employed (Wang et al., 2022).

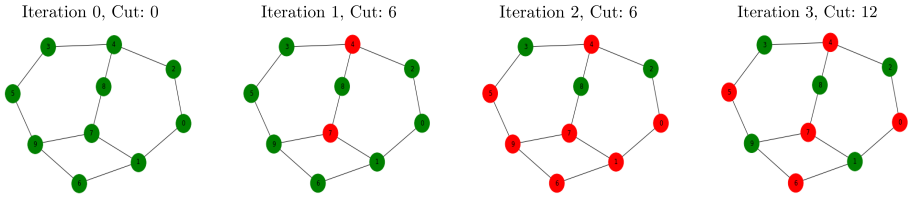

Figure 1: The example of algorithm work on a toy graph of 10 vertices and 12 edges on the Max-Cut problem. The Max-Cut involves partitioning of graph's nodes into two sets such that the number of edges between sets is maximized. At each iteration, the rounded (discretized) solution is shown: red color refers to $x_i = 0$, green to $x_i = 1$. Using intermediate states, the algorithm aims to reclassify each node to the opposite class of the majority of its nearest neighbors at each iteration, thereby optimizing the cut.

4 Method and Training Design

This section introduces our recurrent-feature framework for unsupervised GNN-based combinatorial optimization. We also provide a detailed description of the developed GNN architecture and a selected set of static features.

4.1 Recurrent-Feature Framework

Motivation. Existing GNNs often rely solely on the static node features, which can lead to significant limitations. When applying GNNs to solve a specific problem instance, this standard approach can cause the model to become biased towards these features and face node ambiguity issues, resulting in unsupervised GNNs frequently getting trapped in local extrema during training (Wang & Li, 2023).

Another intuition comes from the structure of the QUBO objective (3), which sums interactions over all pairs of active variables. As a result, the optimal decision for a node i, i.e., whether $x_i = 0$ or $x_i = 1$, depends strongly on the current states of its neighbours. This suggests that solution quality can be improved by allowing each node to adapt based on the evolving state of the graph.

Based on these two observations, we propose a recurrent-feature GNN framework that augments static features with dynamic feedback: each node receives its own predicted state from the previous iteration as an additional feature. This enables the model to reason over the current solution state via message passing and refine its predictions accordingly. The process defines an iterative optimisation loop, where the GNN progressively minimises (3) by adjusting node assignments in response to neighbour updates.

This process is illustrated in Figure 1, which shows the step-by-step work of the algorithm on a toy instance for the maximum cut optimisation problem. The algorithm iteratively updates each node's classification, assigning it to the opposite group of the majority of its neighbouring nodes, effectively improving the quality of the cut with each step. On this toy instance QRF-GNN reaches the optimal cut in only a handful of iterations, whereas an equivalently configured non-recurrent baseline (PI-GNN) requires roughly $50\times$ more iterations to converge to the same solution. Details of the computational experiment on this graph can be found in Appendix B.

Recurrent-feature design. We now formalise the intuition above as an algorithm. Consider an undirected graph $G = (V, E)$ with per-node static features $a_i \in \mathbb{R}^{m_a}$ (their construction is detailed in Section 4.2). Let $h_i^{t,N_l} \in \mathbb{R}^C$ denote the pre-activation output of the GNN's last layer at iteration t for node i, where $C{=}1$ for binary problems (Max-Cut, MIS) and $C{=}k$ for k-class graph coloring. The corresponding probability vector is $p_i^t = \sigma(h_i^{t,N_l})$, with σ a sigmoid (binary) or column-wise softmax over the k logits (multi-class). At iteration $t{=}0$ we initialise $h_i^{-1,N_l} \equiv \mathbf{0}$. At every subsequent iteration $t{+}1$ the previous-iteration output h_i^{t,N_l} is concatenated with the static features and used as the input to the GNN's first layer:

$$h_i^{t+1,0} \;=\; \big[\, a_i,\; \mathrm{sg}(h_i^{t,N_l}) \,\big] \;\in\; \mathbb{R}^{m_a+C}, \tag{4}$$

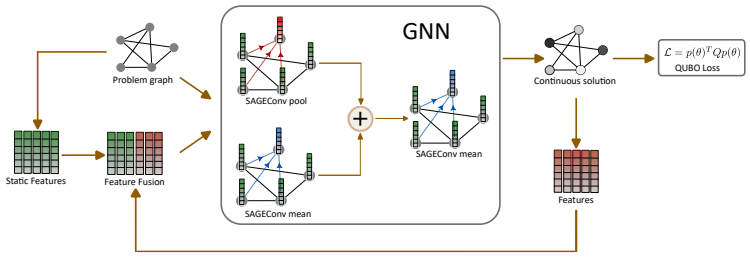

Figure 2: The QRF-GNN architecture. Firstly, the problem graph is associated with the initial QUBO problem, and the node static features are extracted. Then the dynamic features from the previous iteration is concatenated with the static features. Finally, these fused input vectors along with the graph data pass through the graph neural network to update probabilities p_i in Eq. 3.

where $\mathrm{sg}(\cdot)$ is the stop-gradient operator. The stop-gradient is the design choice that makes the framework practical: each iteration is a *single* forward–backward pass on a fresh computation graph that treats the previous-iteration output as a constant input, and a single gradient step is then applied to the GNN weights. The optimisation trajectory across iterations is therefore *not* unrolled into one large differentiable graph (no backpropagation-through-time), which keeps memory and compute per iteration $O(|V| + |E|)$ rather than $O(N_t \cdot (|V| + |E|))$ and avoids the well-known instability of long unrolls.

Algorithm 1 summarises one full iteration: the input vector $h_i^{t+1,0}$ from (4) (line 5) is propagated through the N_l GNN layers (lines 7–12), the QUBO loss (3) is evaluated on the current probabilities (line 13), and a single Adam step updates the weights $W^{t,l}$ to obtain $W^{t+1,l}$ (line 14). The loop is repeated for at most N_t iterations and stopped early once the training loss has plateaued (the precise criterion and value of N_t are stated in Section 5). After termination the final probabilities $p_i = \sigma(h_i^{N_t,N_l})$ are rounded to a binary solution by thresholding (Section 3.3).

Because the recurrent feature h_i^{t,N_l} has dimension C (one per node for binary problems, k per node for k-class coloring), it is small relative to the static feature vector and adds essentially no overhead to the input of the first GNN layer; yet it is precisely the per-node summary of the current candidate solution that the network needs in order to refine its decision based on the running state of every neighbour rather than just on graph structure.

Algorithm 1 Incorporation of the predicted solution into the iterative optimization process

1: **Input:** Graph $G(V, E)$, features $\{a_i, \forall i \in V\}$
2: **Output:** Class probabilities $\{p_i, \forall i \in V\}$
3: **for** $t \in \{0, \ldots, N_t - 1\}$ **do**
4: **for** $i \in V$ **do**
5: $h_i^{t+1,0} \leftarrow \left[a_i, h_i^{t,N_l}\right]$
6: **end for**
7: **for** $l \in \{0, \ldots, N_l - 1\}$ **do**
8: **for** $i \in V$ **do**
9: $h_{N(i)}^{t+1,l+1} \leftarrow \rho\left(\left\{h_j^{t+1,l}, \forall j \in N(i)\right\}\right)$
10: $h_i^{t+1,l+1} \leftarrow f\left(W^{t,l}\left[h_i^{t+1,l}\ h_{N(i)}^{t+1,l+1}\right]\right)$
11: **end for**
12: **end for**
13: $\mathcal{L} \leftarrow \mathcal{L}\left[\sigma\left(\left\{h_i^{t+1,N_l}, \forall i \in V\right\}\right)\right]$
14: $W^{t+1,l} \leftarrow W^{t,l} - \gamma \nabla_{\theta^l} \mathcal{L}, \forall l$
15: **end for**
16: $\{p_i \leftarrow \sigma(h_i^{N_t,N_l}), \forall i \in V\}$

4.2 Architecture and Training Pipeline

Choice of convolution. The recurrent-feature framework admits any standard graph convolution. We tried GCN (Kipf & Welling, 2017), GATv2 (Brody et al., 2021) and GraphSAGE (Hamilton et al., 2017); the recurrent update improved all three (Section 6, Appendix E), but GraphSAGE gave the best absolute quality and we adopt it for the main experiments. The full architecture is sketched in Figure 2.

Parallel-aggregation hidden block. The hidden layer applies two GraphSAGE convolutions to the same input *in parallel* with different neighbour aggregations, MEAN (capturing the relative distribution of neighbour states) and MAX (highlighting the dominant one); each branch has its own batch norm, the two are summed, and a LeakyReLU is applied on top:

$$h_i^{t+1,1} \;=\; \mathrm{LeakyReLU}\big(\;\mathrm{BN}_{\mathrm{mean}}\big(\mathrm{SAGE}^{\mathrm{mean}}(h^{t+1,0})\big)_i \;+\; \mathrm{BN}_{\mathrm{max}}\big(\mathrm{SAGE}^{\mathrm{max}}(h^{t+1,0})\big)_i\;\big)\,. \quad (5)$$

The design is conceptually similar to the Inception module (Szegedy et al., 2015): the GNN sees two complementary views of each neighbourhood at the same depth without stacking additional message-passing layers, which would induce over-smoothing (Rusch et al., 2023).

Output layer and total depth. A dropout step (probability 0.5) is applied to $h^{t+1,1}$, then a final mean-aggregation GraphSAGE convolution maps to per-node logits $h_i^{t+1,N_l} \in \mathbb{R}^C$, and the output activation σ produces the probabilities $p_i^{t+1} = \sigma(h_i^{t+1,N_l}) \in [0,1]^C$ that enter the QUBO loss (3). The network has $N_l = 2$ trainable graph layers (three GraphSAGE convolutions in total: two in the hidden block, one in the output); this deliberately shallow design suffices because information about the running solution is propagated between iterations by the recurrent loop, not by depth.

Static features. The static feature vector $a_i \in \mathbb{R}^{m_a}$ is built once per instance and concatenates a per-graph random vector of dimension d (sampled from $U(0,1)$), an all-ones bias channel of dimension $2d$, and the scalar PageRank (Brin & Page, 1998) of node i broadcast across $2d$ channels, for a total of $m_a = 5d$ columns ($d = 10$, $m_a = 50$ in all main experiments). Following (Cui et al., 2022) this combination distinguishes nodes that would otherwise be indistinguishable to a permutation-equivariant GNN. We do *not* use a trainable per-node embedding (as in PI-GNN (Schuetz et al., 2022a)): in our ablation (Appendix E) it brings no benefit while increasing memory and compute.

Computational complexity. One QRF-GNN iteration is a forward/backward pass through the three GraphSAGE convolutions of width m, one evaluation of the relaxation loss $p^\top Q p$, and one Adam update. The convolutions cost $O(|E|\,m + |V|\,m^2)$ time and $O((|V|+|E|)\,m)$ memory; for graph-defined CO problems the off-diagonal support of Q coincides with E, so the loss is also $O(|E|)$. With fixed hidden width the cost per iteration, and hence the total time under a fixed iteration cap (Section 5), scales near-linearly in $|V| + |E|$, in agreement with Tables 1a and 6 and the scaling experiment in Figure 5. This is a statement about the cost of a fixed budget, not about the time needed to reach a given solution quality.

5 NUMERICAL EXPERIMENTS

We evaluate QRF-GNN on three canonical CO problems – Max-Cut, Maximum Independent Set (MIS), and Graph Coloring – on synthetic and real-world graphs; QUBO objectives are in Appendix A, and baselines introduced in an earlier subsection are referenced by name without repeating the citation. Each table tags methods by training paradigm in a TYPE column: OR (operations-research solver), H (instance-wise heuristic, classical or gradient-based discrete sampling), SL/RL/UL (supervised/reinforcement/unsupervised learning, all amortised over the test distribution by pre-training), and $\mathrm{UL}_{\mathrm{IW}}$ (instance-wise unsupervised, no pre-training, optimised from scratch per graph). QRF-GNN is $\mathrm{UL}_{\mathrm{IW}}$; it requires no per-instance hyperparameter tuning, varying only the hidden width m and iteration cap N_t across experiments. Optimiser settings, early-stopping criterion and the per-table multi-seed reporting protocol are in Appendix B.

5.1 MAXIMUM CUT

Task. Max-Cut partitions the vertices of an unweighted graph $G = (V, E)$ into two subsets to maximise the number of cross-subset edges (QUBO objective in Appendix A, Eq. (6)).

Experimental Setup and Numerical Results. We evaluate on Barabási–Albert (BA) graphs with 4 attaching edges (Barabási & Albert, 1999) (Table 1a), the Gset benchmark (Ye, 2003) (Table 2), and random regular graphs (Appendix D.1). Baselines cover four families: classical solvers (Gurobi, SDP), heuristics (EO (Boettcher, 2003), Tabu Search (Nath & Kuhnle, 2025), BLS (Benlic & Hao, 2013), TSHEA (Wu et al., 2015)), amortised learning solvers (Erdos (Karalias & Loukas, 2020), Gflow (Zhang et al., 2023), ANYCSP (Tönshoff et al., 2023), DiffUCO (Sanokowski et al., 2024), X^2GNN (Acikalin et al., 2025)), and instance-wise unsupervised methods (PI-GNN (Schuetz et al.,

2022a), RUN-CSP (Tönshoff J & M, 2021), CRA (Ichikawa, 2024)). Baseline numbers are taken from the source papers, except PI-GNN and CRA on the BA sets, which their papers do not report and which we ran ourselves in the same setup as QRF-GNN (Appendix D.1).

Table 1: Main results on **(a)** Max-Cut on Barabási–Albert graphs (500 instances per size set) and **(b)** MIS on the canonical DIMES (Qiu et al., 2022) Erdős–Rényi instances (128 graphs of 700–800 nodes; 16 of 9000–11 000). *Size* is the mean objective over the test set; *Time* is total wall-clock. For pre-trained methods (SL/RL/UL) it is inference only and excludes pre-training; for UL_{IW} methods it is the whole solution process from random initialisation. Sources and protocols of all rows are given in Appendix D.1 and D.2.

(a) Max-Cut on BA graphs.

Method	Type	BA-[200-300]		BA-[800-1200]	
		Size	Time	Size	Time
GUROBI	OR	732.47	13:04	2915.29	1:05:29
SDP	OR	700.36	35:47	2786.00	10:00:00
EO	H	729.51	0:41	2921.31	1:39
Tabu	H	733.79	25:00	2926.60	1:35:50
Erdos	UL	693.45	0:46	2870.34	2:49
Gflow	UL	704.30	2:57	2864.61	21:20
ANYCSP	UL	735.12	20:50	2988.60	1:01:25
DiffUCO	UL	733.50	23:05	2981.10	55:00
X^2GNN	UL	735.26	34:10	2989.30	1:00:50
PI-GNN	UL_{IW}	690.32	3:28:00	2868.2	12:36:00
CRA	UL_{IW}	716.23	3:45:00	2916.3	14:08:00
QRF-GNN	UL_{IW}	732.2	8:12	2965.85	15:10

(b) MIS on canonical DIMES ER graphs.

Method	Type	ER-[700-800]		ER-[9000-11000]	
		Size	Time	Size	Time
KaMIS	H	44.87	52:13	381.31	7:36:00
iSCO	H	44.77	1:23	377.50	9:23
ReSCO	H	44.18	17:00	378.25	3:54:00
RLSA	H	44.10	0:20	375.31	1:40
Gflow	UL	41.14	2:55	349.42	1:49:43
DiffUCO	UL	43.63	1:31	–	–
RLNN	UL	43.34	1:22	363.34	11:46
DIMES	RL	42.06	12:01	332.80	12:31
X^2GNN	UL	44.91	2:01:59	346.56	28:51
PI-GNN	UL_{IW}	38.23	44:50	308.2	54:30
CRA	UL_{IW}	41.64	47:30	360.71	1:03:00
QRF-GNN	UL_{IW}	42.8	30:30	381.1	1:28:09

On BA graphs (Table 1a), X^2GNN, ANYCSP and DiffUCO lead on the large set, with QRF-GNN the strongest UL_{IW} method, ahead of every classical solver and instance-wise baseline; on the small set QRF-GNN matches Gurobi while X^2GNN and ANYCSP top the table. The three amortised leaders are pre-trained on the very BA distribution they are tested on, and their time excludes this pre-training, so their rows and ours answer different questions (Section 1).

On Gset (Table 2) baselines use heterogeneous best-of-N protocols (CRA reports best of 5 seeds (Ichikawa, 2024); details in Appendix D.1). For QRF-GNN we report mean $\pm$ standard deviation across 20 seeds and their best (seeds are independent and parallelisable on a single GPU). On the largest graph G70 QRF-GNN finds the best cut an order of magnitude faster than BLS and TSHEA. The full 36-instance comparison is in Appendix D.1 (Table 6).

Table 2: Cut edges on a representative subset of Gset instances (Ye, 2003). For QRF-GNN we report the mean $\pm$ standard deviation and the best across 20 seeds.

Graph	$\lvert V\rvert$	$\lvert E\rvert$	Heuristics (H)			UL_{IW}			**QRF-GNN (Ours)**	
			BLS	**TSHEA**	**EO**	**PI-GNN**	**RUN-CSP**	**CRA**	Avg $\pm$ Std	Best
G14	800	4694	**3064**	**3064**	3058	3026	2943	3045	3045 ± 8	3058
G15	800	4661	**3050**	**3050**	3046	2990	2928	3026	3036 ± 7	3049
G22	2000	19990	**13359**	**13359**	13323	13181	13028	13332	13330 ± 7	13340
G49	3000	6000	**6000**	**6000**	**6000**	5918	**6000**	**6000**	**6000** ± 0	**6000**
G50	3000	6000	**5880**	**5880**	5878	5820	**5880**	**5880**	5879 ± 1	**5880**
G55	5000	12468	10294	**10299**	10212	10138	10116	10201	10267 ± 9	10282
G70	10000	9999	9541	9548	9433	9421	9319	9465	9542 ± 12	**9559**

5.2 Maximum Independent Set

Task. MIS seeks a maximum-cardinality subset $S \subset V$ of pair-wise nonadjacent vertices in $G = (V, E)$ (QUBO objective with adaptive independence penalty in Appendix A, Eq. (7)).

Experimental Setup and Numerical Results. We evaluate on the canonical fixed DIMES (Qiu et al., 2022) ER test sets (128 ER-[700-800] and 16 ER-[9000-11 000] graphs); results are in Table 1b. Beyond the baselines reused from Max-Cut (Gflow, DiffUCO, X^2GNN, PI-GNN, CRA), we add the classical solver KaMIS (Lamm et al., 2017), the instance-wise gradient-based samplers iSCO (Sun et al., 2023a), ReSCO (Li & Zhang, 2025) and RLSA (Feng & Yang, 2025) (all single-chain), the amortised UL solver RLNN (Feng & Yang, 2025), and the amortised RL solver DIMES. For ER-[9000-11 000] no in-distribution X^2GNN checkpoint is released, so we run the authors' RB-trained checkpoint OOD (Appendix D.2).

On ER-[700-800] X^2GNN and KaMIS lead, with iSCO and ReSCO close behind; QRF-GNN is ahead of every amortised learning baseline and of CRA, and within 0.5% of RLSA. On ER-[9000-11 000] QRF-GNN essentially ties KaMIS and beats every learning-based and sampling baseline, the only $\mathrm{UL}_{\mathrm{IW}}$ method to match the classical solver on the large benchmark. RB-model (Xu & Li, 2006) and structured (Acikalin et al., 2025) MIS benchmarks, plus an extended ER comparison with older SL/RL baselines, are in Appendix D.2.

5.3 Graph Coloring

Task and results. We consider two standard variants on $G = (V, E)$: for a fixed budget of k colours, minimise the number of monochromatic edges; and find the smallest k that admits a valid colouring. Both share one QUBO objective (Appendix A, Eq. (8)). We evaluate on the COLOR dataset (Trick, 2002) (25–561 nodes) and on the citation graphs Cora, Citeseer and Pubmed (Li et al., 2022) (up to $\sim 20\,000$ nodes), against PI-GNN in its colouring variant (Schuetz et al., 2022b), RUN-CSP, the colouring-specific GNNs GNN-1N (Wang et al., 2023) and GDN (Li et al., 2022), and the heuristic HybridEA (Galinier & Hao, 1999; Lewis, 2021). Tables 11 and 12 (Appendix D.3) give the results: at $k = \chi$ QRF-GNN has the fewest monochromatic edges on every instance, below HybridEA on the two hardest queen graphs, and on the smallest-k variant it matches HybridEA on all but one instance and is the strongest GNN throughout.

6 Impact of the Recurrent Feature on GNN Performance

To see *how* the recurrent feature changes training, we run QRF-GNN on Gset `G14` (n=800, m=4694) twice with the same seed, hyperparameters and warm-start, with the recurrent feature enabled (**with rec**, the default) and disabled (**without rec**). Figure 4 shows the two trajectories from four angles. With the recurrent feature the model keeps reassigning a non-trivial fraction of variables (b) and visits a much larger part of $\{0, 1\}^n$ (c) instead of freezing in one configuration; the visited bitstrings keep near-best cut quality at Hamming distances of several hundred from the current best (d), and the final cut at the same budget is strictly higher (a). In short, the recurrent feature turns training from a local descent that stops early into a persistent walk through high-quality solutions. The same diagnostic on six more Gset instances (Appendix C.2, Figure 7) shows the gain in two regimes, broad exploration or early lock-on to a high-quality basin, neither of which the non-recurrent variant reproduces.

Appendix C.4 quantifies these trajectories: both variants become near-binary within tens of iterations; with the recurrent feature the relaxed loss stays within 0.01% of the discrete cut, the cut still drops in about half of the steps and the best iterate is found late, yet the returned solution is a 1-flip local optimum up to 0.2% of the cut, against 1–2% left without the feature. Appendix C.3 compares the three instance-wise methods by wall-clock in one implementation: QRF-GNN passes the final cut of both baselines within two seconds.

How local optima are left. A discrete local search can leave a 1-flip local optimum only by accepting a worse solution. The recurrent state gives the network a different route. Its logits carry not only the current assignment of each node but also how strongly the node holds it, and this margin behaves as an accumulator: the confidence of a group of nodes can erode over many iterations while the rounded solution does not change, until the whole group flips together. Figure 3 shows this on `G14`. Nodes that are about to flip lose most of their margin in the preceding iterations, whereas the margins of all other nodes stay at their typical level; in the non-recurrent variant no such precursor exists, and its flips are indistinguishable from noise. The graded feedback is essential: when the rounded solution rather than the logits is fed back, the accumulator disappears, the solution stays

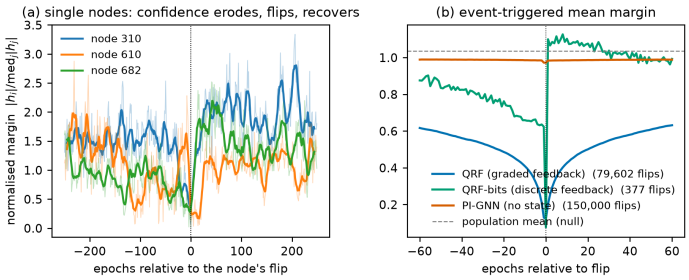

Figure 3: The logit margin is an accumulator (QRF-GNN on Gset `G14`). **(a)** Normalised margin $|h_i|/\mathrm{med}_j|h_j|$ of three nodes in a ± 250-iteration window around their flip (dotted line): the margin erodes, the node flips, the margin recovers. **(b)** Mean normalised margin around all flips ($\approx$ 80k events) for QRF-GNN, for QRF-GNN with rounded feedback, and for the non-recurrent variant; dashed: population mean.

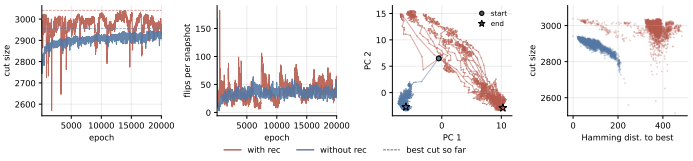

Figure 4: QRF-GNN training dynamics on Gset `G14`, comparing runs **with** and **without** the recurrent feature. **(a)** Per-epoch cut size; dashed lines mark each variant's final best cut. **(b)** Number of variables that flip between consecutive 10-epoch snapshots. **(c)** Visited bitstrings projected onto two shared leading PCA directions. **(d)** Each visited bitstring as a dot, x = Hamming distance to the variant's own best, y = cut size.

frozen for most of the run, and the final cut falls below even the non-recurrent baseline. The resulting moves are coordinated rather than random: in about a fifth of the iterations a group of connected nodes flips together with a gain that no single flip from the previous state could achieve. The same picture holds on `G22`; the underlying statistics are given in Appendix C.4.

The gain is not specific to SAGE. Figure 8 (Appendix E) compares QRF-GNN (SAGE) with GCN (Kipf & Welling, 2017) and GATv2 (Brody et al., 2021) variants of the same width on a subset of Gset instances (20 seeds each), with and without the recurrent feature. The feature raises the maximum cut for all three backbones, and SAGE dominates GCN and GATv2 in both settings, which motivates it as the default.

7 Conclusion

We proposed the recurrent-feature mechanism for instance-wise unsupervised GNN solvers of QUBO problems: the network's previous prediction is fed back as a dynamic node feature, which turns per-instance training into a guided search over binary configurations. It improves every GNN convolution we tested; with a two-branch GraphSAGE block and compact static features it gives QRF-GNN, which is far ahead of prior instance-wise unsupervised GNNs on Max-Cut, Graph Coloring and Maximum Independent Set, matches or exceeds pre-trained neural solvers evaluated in-distribution, and approaches the best dedicated heuristics. **Limitations.** QRF-GNN is a heuristic without convergence or optimality guarantees; its cost analysis concerns a fixed iteration budget, not the time to reach a target quality. Its scope is problems with compact, well-scaled QUBO formulations, and results depend on the number of restarts. Training from scratch forgoes pre-training; we see this as the price of assuming nothing about the instance, and a pre-trained initialisation is a natural extension.

REPRODUCIBILITY STATEMENT

The architecture and the recurrent-feature training loop are specified in Section 4 and Algorithm 1; all hyperparameters, iteration budgets, stopping rules and the multi-seed protocol are listed in Appendix B. All benchmarks are public (Gset, COLOR, Cora/Citeseer/Pubmed, BA graphs, the canonical DIMES ER and RB test sets); QUBO formulations are given in Appendix A. Sources of every baseline number and the protocol of every table are stated in Appendix D. Code will be released upon acceptance.

AI USE STATEMENT

Generative AI tools were used for language editing of the manuscript and for assistance with plotting and auxiliary analysis scripts. All AI-assisted text and code were reviewed and verified by the authors, who take full responsibility for the content of this work.

REFERENCES

Utku Umur Acikalin, Aaron M. Ferber, and Carla P. Gomes. Learning to explore and exploit with GNNs for unsupervised combinatorial optimization. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=vaJ4FObpXN.

Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Ismail R. Alkhouri, Cedric Le, George K. Atia, Alfred O. Hero, and Alvaro Velasquez. Differentiable quadratic optimization for the maximum independent set problem. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research. PMLR, 2025.

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.

Maria Chiara Angelini and Federico Ricci-Tersenghi. Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set. *Nature Machine Intelligence*, 5(1):29–31, January 2023. doi: 10.1038/s42256-022-00589-y. URL https://doi.org/10.1038/s42256-022-00589-y.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286 (5439):509–512, 1999.

Una Benlic and Jin-Kao Hao. Breakout local search for the max-cutproblem. *Engineering Applications of Artificial Intelligence*, 26(3):1162–1173, 2013. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2012.09.001. URL https://www.sciencedirect.com/science/article/pii/S0952197612002175.

S. Boettcher. Numerical results for ground states of spin glasses on Bethe lattices. *The European Physical Journal B - Condensed Matter*, 31(1):29–39, January 2003. doi: 10.1140/epjb/e2003-00005-y. URL https://doi.org/10.1140/epjb/e2003-00005-y.

Stefan Boettcher. Inability of a graph neural network heuristic to outperform greedy algorithms in solving combinatorial optimization problems. *Nature Machine Intelligence*, 5(1):24–25, January 2023. doi: 10.1038/s42256-022-00587-0. URL https://doi.org/10.1038/s42256-022-00587-0.

Stefan Boettcher and Allon G. Percus. Optimization with extremal dynamics. *Physical Review Letters*, 86(23):5211–5214, June 2001. doi: 10.1103/physrevlett.86.5211. URL https://doi.org/10.1103/physrevlett.86.5211.

Endre Boros and Peter L. Hammer. The max-cut problem and quadratic 01 optimization; polyhedral aspects, relaxations and bounds. *Annals of Operations Research*, 33:151–180, 1991.

Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2013.02.041. URL https://www.sciencedirect.com/science/article/pii/S0020025513001588. Prediction, Control and Diagnosis using Advanced Neural Computations.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107–117, 1998. ISSN 0169-7552. doi: https://doi.org/10.1016/S0169-7552(98)00110-X. URL https://www.sciencedirect.com/science/article/pii/S016975529800110X. Proceedings of the Seventh International World Wide Web Conference.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Quentin Cappart, Didier Chételat, Elias Khalil, Andrea Lodi, Christopher Morris, and Petar Velikovi. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023. URL http://jmlr.org/papers/v24/21-0449.html.

Hejie Cui, Zijie Lu, Pan Li, and Carl Yang. On positional and structural node features for graph neural networks on non-attributed graphs. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, pp. 38983902, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392365. doi: 10.1145/3511808.3557661. URL https://doi.org/10.1145/3511808.3557661.

Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Graph reinforcement learning for combinatorial optimization: A survey and unifying perspective. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=HduK51xNtS. Survey Certification.

Shengyu Feng and Yiming Yang. Regularized langevin dynamics for combinatorial optimization. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research. PMLR, 2025.

Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 01 1999. doi: 10.1023/A:1009823419804.

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. doi: 10.1007/s10479-022-04634-2. URL https://doi.org/10.1007/s10479-022-04634-2.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 10251035, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Peter L. Hammer and Sergiu Rudeanu. Pseudo-boolean programming. *Operations Research*, 17 (2):233–261, 1969. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/168831.

Yuma Ichikawa. Controlling continuous relaxation for combinatorial optimization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=ykACV1IhjD.

Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27(1-2):70–98, April 2022. doi: 10.1007/s10601-022-09327-y. URL https://doi.org/10.1007/s10601-022-09327-y.

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.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

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.

Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *Journal of Heuristics*, 23(4):207229, May 2017. ISSN 1572-9397. doi: 10.1007/s10732-017-9337-x. URL http://dx.doi.org/10.1007/s10732-017-9337-x.

R. Lewis. *Guide to Graph Colouring: Algorithms and Applications*. 01 2021. ISBN 978-3-030-81053-5. doi: 10.1007/978-3-030-81054-2.

Muheng Li and Ruqi Zhang. Reheated gradient-based discrete sampling for combinatorial optimization. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=uPCvfyr2KP.

Wei Li, Ruxuan Li, Yuzhe Ma, Siu On Chan, David Pan, and Bei Yu. Rethinking graph neural networks for the graph coloring problem. *ArXiv*, abs/2208.06975, 2022.

Yang Li, Jinpei Guo, Runzhong Wang, and Junchi Yan. From distribution learning in training to gradient search in testing for combinatorial optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=JtF0ugNMv2.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Neural Information Processing Systems*, 2018. URL https://api.semanticscholar.org/CorpusID:53027872.

Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2, 2014. doi: 10.3389/fphy.2014.00005. URL https://doi.org/10.3389/fphy.2014.00005.

Ankur Nath and Alan Kuhnle. MaxCutBench: Revisiting and benchmarking graph neural networks for maximum cut. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856.

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.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. DIMES: A differentiable meta solver for combinatorial optimization problems. In *Advances in Neural Information Processing Systems 35*, 2022.

T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *ArXiv*, abs/2303.10993, 2023. URL https://api.semanticscholar.org/CorpusID:257632346.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research. PMLR, 2024.

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.

Martin J. A. Schuetz, J. Kyle Brubaker, and Helmut G. Katzgraber. Reply to: Inability of a graph neural network heuristic to outperform greedy algorithms in solving combinatorial optimization problems. *Nature Machine Intelligence*, 5(1):26–28, January 2023. doi: 10.1038/s42256-022-00588-z. URL https://doi.org/10.1038/s42256-022-00588-z.

Haoran Sun, Katayoon Goshvadi, Azade Nova, Dale Schuurmans, and Hanjun Dai. Revisiting sampling for combinatorial optimization. In *Proceedings of the 40th International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2023a.

Haoran Sun, Etash Kumar Guha, and Hanjun Dai. Annealed training for combinatorial optimization on graphs, 2023b. URL https://openreview.net/forum?id=YHCR6CFAK6v.

Zhiqing Sun and Yiming Yang. DIFUSCO: Graph-based diffusion solvers for combinatorial optimization. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=JV8Ff0lgVV.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015. doi: 10.1109/CVPR.2015.7298594.

Jan Tönshoff, Berke Kisin, Jakob Lindner, and Martin Grohe. One model, any CSP: Graph neural networks as fast global search heuristics for constraint satisfaction. In *Proceedings of the 32nd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4280–4288, 2023.

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.

M. Trick. Color dataset, 2002. URL https://mat.tepper.cmu.edu/COLOR02/.

Haoyu Peter Wang and Pan Li. Unsupervised learning for combinatorial optimization needs meta learning. 2023. URL https://openreview.net/forum?id=-ENYHCE8zBp.

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.

Qinghua Wu, Yang Wang, and Zhipeng Lü. A tabu search based hybrid evolutionary algorithm for the max-cut problem. *Applied Soft Computing*, 34:827–837, 2015. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc.2015.04.033. URL https://www.sciencedirect.com/science/article/pii/S1568494615002604.

Yikai Wu, Hao Zhao, and Sanjeev Arora. Unrealized expectations: Comparing AI methods vs classical algorithms for maximum independent set. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856.

Ke Xu and Wei Li. Many hard examples in exact phase transitions. *Theoretical Computer Science*, 355(3):291–302, 2006. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2006.01.001. URL https://www.sciencedirect.com/science/article/pii/S0304397506000181.

Weichi Yao, Afonso S. Bandeira, and Soledad Villar. Experimental performance of graph neural networks on random instances of max-cut. In Dimitri Van De Ville, Manos Papadakis, and Yue M. Lu (eds.), *Wavelets and Sparsity XVIII*, volume 11138, pp. 111380S. International Society for Optics and Photonics, SPIE, 2019. doi: 10.1117/12.2529608. URL https://doi.org/10.1117/12.2529608.

Y. Ye. The G-set dataset, 2003. URL https://web.stanford.edu/~yyye/yyye/Gset/.

Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with GFlownets. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=sTjW3JHs2V.

A QUBO FORMULATIONS OF EVALUATED CO PROBLEMS

For completeness, this section gives the QUBO objective minimised by QRF-GNN on each of the three benchmark problems considered in Section 5.

Maximum Cut. The Max-Cut problem partitions the vertices of an undirected graph $G = (V, E)$ into two subsets so as to maximise the number of edges with endpoints in different subsets (or, in the weighted case, the total weight of such edges). Its QUBO objective is

$$\mathcal{F}(x) = \sum_{i<j} A_{ij}(2x_i x_j - x_i - x_j), \quad \forall i \in V, \tag{6}$$

where A is the adjacency matrix of G and the decision variable $x_i \in \{0, 1\}$ indicates which of the two subsets vertex i belongs to. We consider only unweighted graphs ($A_{ij} = 1$ for all $(i, j) \in E$).

Maximum Independent Set. For a graph $G = (V, E)$, MIS seeks a subset $S \subset V$ of pair-wise nonadjacent vertices of maximum size. The QUBO cost function is

$$\mathcal{F}(x) = -\sum_{i \in V} x_i + P \sum_{(i,j) \in E} x_i x_j, \quad x_i \in \{0, 1\}, \tag{7}$$

where $x_i = 1$ iff $i \in S$ and $P > 0$ penalises independence violations; we use an adaptive penalty schedule (Appendix D.2).

Graph Coloring. We consider two variants on $G = (V, E)$: **(i)** given a fixed number of colours k, minimise the number of monochromatic (constraint-violating) edges; **(ii)** find the smallest k admitting a valid colouring with zero violations. Both share the QUBO objective

$$\mathcal{F}(x) = \sum_{i \in V} \left(1 - \sum_{c=1}^{k} x_{i,c}\right)^2 + \sum_{(i,j) \in E} \sum_{c=1}^{k} x_{i,c} x_{j,c}, \tag{8}$$

where in variant (ii) k is also part of the optimisation. The loss reduces to the second term because the GNN's softmax output enforces that each node selects a unique colour, so the first term vanishes.

B TECHNICAL DETAILS AND ALGORITHM

B.1 GENERAL EXPERIMENTAL SETUP

All random graphs in this work were generated with NetworkX[1], and the QRF-GNN architecture is implemented with the DGL library[2]. Algorithm 2 gives the pseudocode of one QRF-GNN iteration. We use Adam without a learning-rate schedule, with learning rate 0.014 (set empirically) and the remaining optimiser parameters at their defaults; gradients are clipped at Euclidean norm 2.

We cap the number of iterations at 5×10^4 for random regular graphs and 10^5 otherwise, although convergence is often reached much earlier: training stops when the loss varies by less than 10^{-5} over the last 500 iterations. For graph coloring we additionally stop when the absolute loss falls below 10^{-3}.

Dropout is 0.5, the random part of the input has dimension 10, and the hidden width is 50 for Max-Cut and 140 for coloring.

Due to the stochasticity of the algorithm, multiple seeds are preferable. Separate runs can be done in parallel, possibly across several GPUs; if memory allows, the RUN-CSP scheme of (Tönshoff J & M, 2021) can be used, where one composite graph containing duplicates of the original is built for the input. We trained on a single NVIDIA H100 GPU; conventional heuristics ran on a machine with two Intel Xeon E5-2670 v3 @ 2.30GHz.

We tested three ways to recursively use the probability data, namely raw values before the sigmoid layer, values after it, or both concatenated. Different choices led to minor improvements on some graphs and slight degradations on others; we report the concatenated variant.

Algorithm 2 Forward propagation of the QRF-GNN algorithm at iteration t

Input: Graph $G(V, E)$, static nodes features $\{a_i, \forall i \in V\}$
Output: Probability p_i, hidden state $h_i^t, \forall i \in V$
$h_i^{t,0} \leftarrow \left[a_i \; h_i^{t-1}\right], \quad \forall i \in V$
for $i \in V$ **do**
  $h_{N(i)}^{t,1} \leftarrow \rho_{\text{mean}}\left(\left\{h_j^{t,0}, \forall j \in N(i)\right\}\right)$
  $h_i^{t,1} \leftarrow f\left(W^1\left[h_i^{t,0} \; h_{N(i)}^{t,1}\right]\right)$
  $h_{N(i)}^{t,2} \leftarrow \rho_{\text{pool}}\left(\left\{h_j^{t,0}, \forall j \in N(i)\right\}\right)$
  $h_i^{t,2} \leftarrow f\left(W^2\left[h_i^{t,0} \; h_{N(i)}^{t,2}\right]\right)$
end for
$\{h_i^{t,1}\} \leftarrow \text{BN}_{\gamma 1,\beta 1}(\{h_i^{t,1}, \forall i \in V\})$
$\{h_i^{t,2}\} \leftarrow \text{BN}_{\gamma 2,\beta 2}(\{h_i^{t,2}, \forall i \in V\})$
for $i \in V$ **do**
  $h_i^{t,12} \leftarrow f(h_i^{t,1} + h_i^{t,2})$
  $h_i^{t,12} \leftarrow \text{Dropout}(h_i^{t,12})$
  $h_{N(i)}^{t,\text{out}} \leftarrow \rho_{\text{mean}}\left(\left\{h_j^{t,12}, \forall j \in N(i)\right\}\right)$
  $h_i^{t,\text{out}} \leftarrow f\left(W^{\text{out}}\left[h_i^{t,12} \; h_{N(i)}^{t,\text{out}}\right]\right)$
end for
$p_i, h_i^t \leftarrow \sigma(h_i^{t,\text{out}}), h_i^{t,\text{out}}, \quad \forall i \in V$

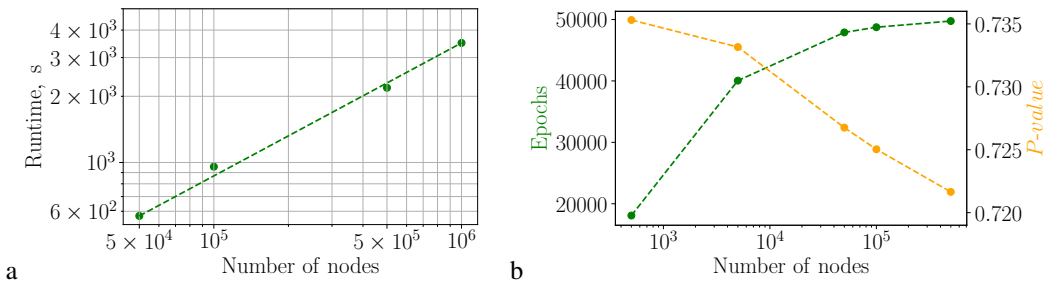

Figure 5: a) The computation time of 5×10^4 iterations of QRF-GNN on random regular graphs with $d = 5$ in the sparse format depending on the number of nodes. b) The iteration number averaged over 20 graphs at which the algorithm found the best solution during the training process (green) and the mean *P-value* for 1 run (orange).

C Analysis

C.1 Convergence and learning-rate robustness

The number of runs and iterations was chosen for a more fair comparison with other algorithms rather than for absolute optimality; more runs or iterations can lead to better results. We conducted additional experiments on 20 random regular graphs with $d = 5$ and up to one million nodes, with one run per graph and at most 5×10^4 iterations. Training times for large sparse-format graphs on a single GPU are in Figure 5a. We also analysed how the iteration count affects solution quality. As the number of vertices grows, the iteration at which the last best solution was saved moves closer to the cap (Figure 5b), while the mean *P-value* of one run drops from 0.735 to 0.722 – partly explained by the limited training budget. For example, training QRF-GNN for 10^5 iterations on $n = 5 \times 10^4$ graphs raises the mean *P-value* from 0.726 to 0.728, whereas on $n = 500$ graphs no such effect is observed. Hence convergence on large instances is difficult to assert under the given constraint; we recommend tracking the latest best solution and terminating once it has not improved for a sufficiently large ($> 10^4$) number of iterations.

To study the algorithm's robustness to hyperparameters, we run the default QRF-GNN with two parallel layers on Gset Max-Cut instances, varying only the learning rate (other hyperparameters as in Section 5). As Figure 6 shows, small learning rates fail to converge within 10^5 iterations; for values above 0.01 the results are relatively stable, with the best cut sizes reached between 0.01 and 0.02.

C.2 Recurrent-feature training dynamics on additional Gset instances

In Section 6 we compared QRF-GNN with and without the recurrent feature on a single Gset instance (`G14`) using a four-panel diagnostic: per-epoch cut size, per-snapshot variable flips, PCA of visited bitstrings, and cut size as a function of Hamming distance to the variant's own best. We repeat the same diagnostic on six more Gset instances (`G15, G22, G49, G50, G55, G70`), with everything else fixed (same seed, hyperparameters and warm-start; the only difference between the **with rec** and **without rec** runs is whether the recurrent input from (4) is fed to the GNN or replaced by zeros).

Figure 7 reveals two qualitatively different but mutually consistent regimes induced by the recurrent feature, both leading to substantially higher final cuts than the non-recurrent variant.

Persistent-walk regime (G15, G22, G55, G70). The picture is essentially the same as on G14 (Fig. 4 of the main text): the with-rec variant keeps reassigning a non-trivial fraction of variables (panel b), spreads its visited bitstrings widely along the leading PCA directions (panel c), and sustains near-best cut quality at Hamming distances of several hundred to a thousand from its current

[1] `https://networkx.org/`
[2] `https://www.dgl.ai/`

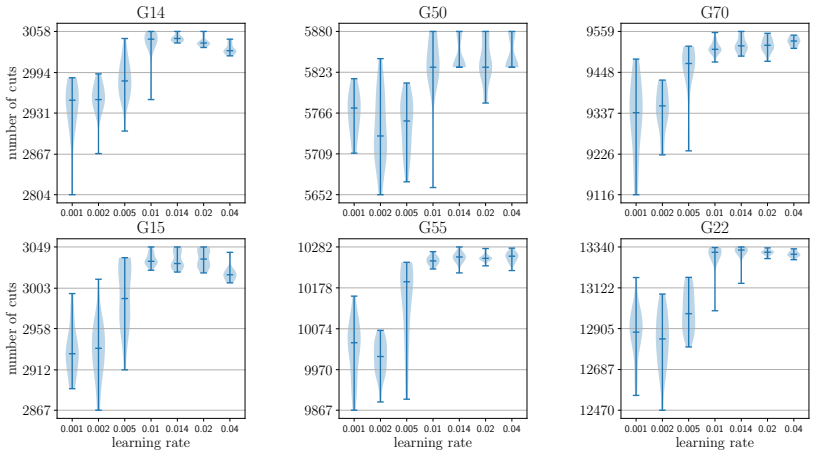

Figure 6: Results distribution for 20 runs of the default QRF-GNN architecture depending on the learning rate for the Max-Cut problem on several instances from Gset. The number of iterations in all cases was fixed at 10^5. Horizontal lines mean maximum, median and minimum values.

best (panel d). The without-rec variant rapidly freezes into a single configuration with very few flips, very little PCA spread, and lower cut size that does not recover even with more epochs (panel a).

Direct-lock-on regime (G49, G50). The recurrent feature instead allows QRF-GNN to commit early to a high-quality basin and stay there: in panel (a), the with-rec variant jumps to a near-optimal cut within the first few epochs and remains essentially flat for the rest of training, while the without-rec variant slowly climbs toward a much lower plateau over 20,000 epochs. Panel (b) shows that both variants are quiescent for these graphs (few flips), but the trajectories that the recurrent variant settles into are systematically high-cut, while the non-recurrent variant settles into low-cut basins. Panels (c) and (d) reflect the same effect: the with-rec PCA cluster is small but located in a region of consistently high cut size, the without-rec cluster is equally tight but at lower quality. We attribute this to the fact that on `G49` and `G50` the QUBO has a small number of high-quality basins reachable from a generic initialisation, and the recurrent input lets the network identify the right basin in the first few iterations and stop wandering.

Both regimes are consistent with the interpretation in Section 6: rather than locking the network into a fixed exploration or exploitation budget, the recurrent feature lets QRF-GNN *adapt its search behaviour to the structure of the instance*. Where the loss landscape requires broad exploration, the with-rec variant performs an extended persistent walk through high-quality solutions; where a good direction is identifiable from the first few iterations, it commits to it immediately. In both cases the recurrent variant ends up at a strictly higher cut size than the non-recurrent baseline.

C.3 Budget-matched comparison inside the instance-wise family

PI-GNN, CRA and QRF-GNN have the same cost per iteration up to small constants: one forward pass, one backward pass and one Adam step on the same graph. QRF-GNN adds one input column (the previous logits, detached, so no extra pass and no extra gradient) and CRA adds an elementwise penalty with a scalar weight. In our implementation, with the whole step captured in a CUDA graph, one iteration on `G14` takes 0.333 ms with the recurrent feature and 0.330 ms without it on an RTX 5090; in a plain eager loop the costs are 2.64, 2.64 and 2.25 ms for QRF-GNN, CRA and PI-GNN respectively (2.69, 3.03 and 2.32 ms on `G70`). The iteration of QRF-GNN is thus not the cheaper one, and iteration-matched budgets do not favour it. Wall-clock differences between papers come from implementations: the code released with PI-GNN and CRA is built on DGL, which no longer builds for current PyTorch and GPUs; whenever we ran these baselines ourselves, we re-implemented them in our codebase.

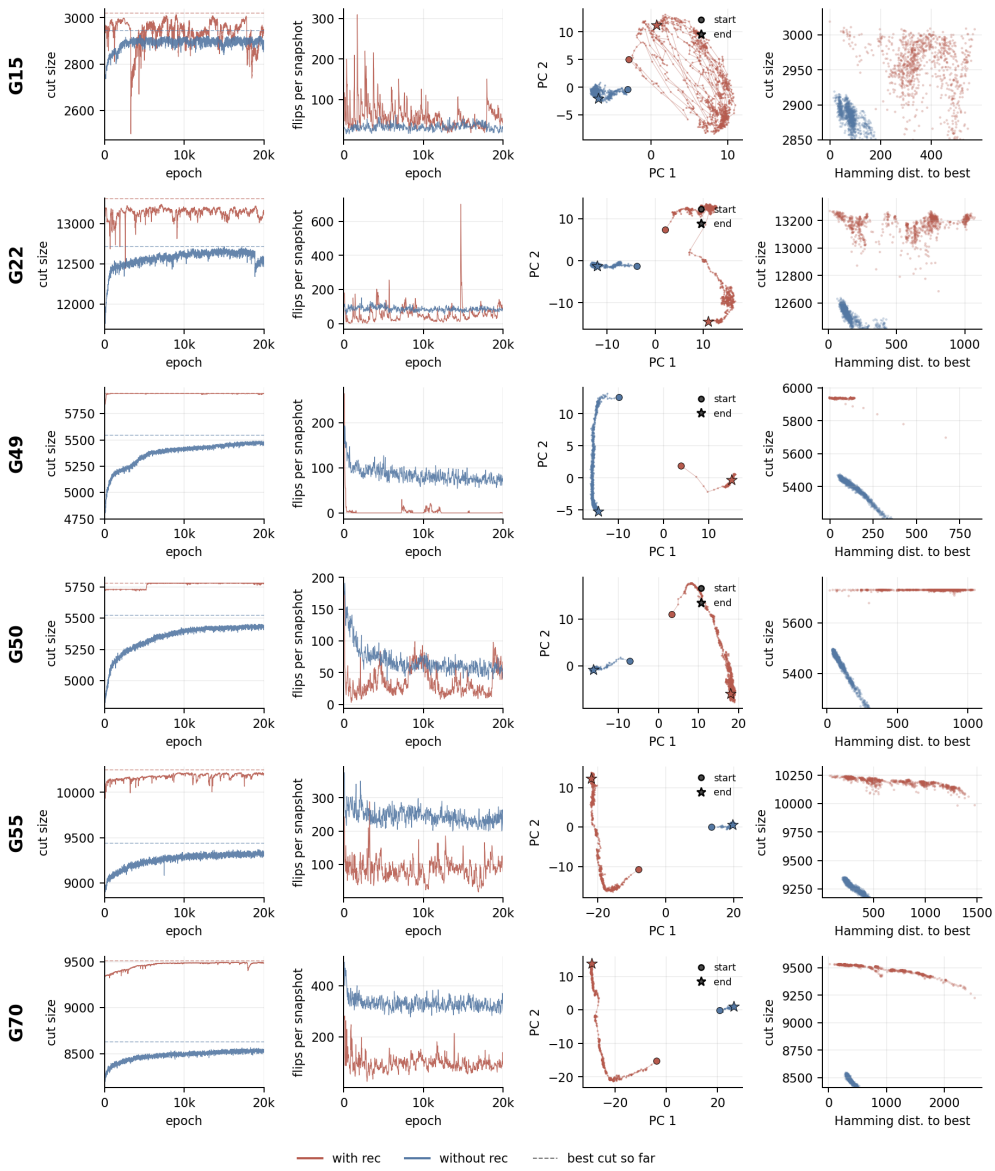

Figure 7: Same four-panel training-dynamics diagnostic as Fig. 4 (main text), here applied to six additional Gset instances (`G15, G22, G49, G50, G55, G70`; one row per instance). Within each row, columns from left to right show: **(a)** per-epoch cut size; **(b)** variables flipped between consecutive 10-epoch snapshots; **(c)** visited bitstrings projected onto two shared leading PCA directions, with start (•) and end ($\star$) markers; **(d)** each visited bitstring as one dot, x = Hamming distance to the variant's own best, y = cut size. **red** = QRF-GNN *with* the recurrent feature, **blue** = same architecture and seed *without* it. The recurrent feature is consistently beneficial across all six instances, but in two qualitatively different regimes (see text): broad persistent-walk exploration on `G15, G22, G55, G70`; immediate lock-on to a high-quality basin on `G49, G50`.

Table 3 reports the best cut reached within a wall-clock budget by the three methods on `G14`, all run in the same eager loop of our codebase on one RTX 5090 with the same architecture, static features, optimiser and iteration cap (10^5; 3 seeds; for CRA we tried three annealing rates and report the best). QRF-GNN is above the final value of both baselines after two seconds, so the ranking does not depend on where the budget is cut. CRA is late by construction, since its annealing schedule spans the whole budget; its final value here is below the tuned best-of-5 value reported in (Ichikawa, 2024), so its row is a lower bound. Time to a target cut: 2900 in 6.9 s for PI-GNN and 1.4 s for

QRF-GNN; 2950 in 1.4 s for QRF-GNN, reached by PI-GNN in one of three seeds only and never by CRA; 3000 only by QRF-GNN, in 2.3 s.

Table 3: Best cut reached within a wall-clock budget on Gset G14 (best known cut 3064): mean over 3 seeds / best of 3. All methods run in the same implementation on one RTX 5090 with the same architecture and settings.

WALL-CLOCK	PI-GNN	CRA	QRF-GNN
2 S	2847 / 2866	2369 / 2422	2993 / 3001
10 S	2912 / 2918	2369 / 2422	3035 / 3039
30 S	2922 / 2929	2419 / 2470	3036 / 3042
60 S	2931 / 2934	2522 / 2538	3037 / 3042
120 S	2939 / 2949	2585 / 2609	3042 / 3047
260 S (10^5 ITERATIONS)	2942 / 2957	2831 / 2846	3044 / 3047

C.4 STRUCTURAL DIAGNOSTICS OF THE TRAINING DYNAMICS

We instrumented runs on G14, G22 and G49 (10^5 iterations, a snapshot every 10 iterations, with and without the recurrent feature, default hyperparameters, single seeds; two seeds on G14). At every snapshot we record the fraction of probabilities outside $[0.05, 0.95]$ (saturation), the relaxed objective $p^\top Qp$ against the discrete cut of the rounded vector, the cut of the current and of the best rounded iterate, the best single-flip gain of the current rounded vector, Hamming distances, and a hash of the rounded vector. At the end we run greedy single-flip ascent from the returned solution. Table 4 summarises the results.

Onset. Both variants become near-binary within the first tens of iterations ($> 90\%$ saturation after 10–30 iterations, $> 99\%$ after 30–80 with the recurrent feature) and stay there for the rest of the run. Saturation is a property of the relaxation, not of the mechanism.

Tightness. After the onset the relaxed objective and the discrete cut of the rounded vector differ by less than 0.01% with the recurrent feature, and their per-step changes correlate at 0.97–0.99. A gradient step on the relaxed loss is therefore a step on the discrete objective, and rounding loses nothing.

Monotonicity. Entering the near-binary regime does not make the discrete objective monotone. On G14 and G22 the cut of the current iterate drops in about half of the snapshot steps, by up to 220 edges; it falls more than 1% below the best about 300 times per run, and the best iterate is found after tens of thousands of iterations and after many such drops. The walk reaches Hamming distance 460–720 from its own best, against 140–280 without the feature. On G49 the run instead locks on early and is nearly monotone, as in Figure 7.

Local optimality. The visited iterates are almost never 1-flip local optima on G14 and G22 (44–56 improving flips are available on average), so the run does not stop at the first local optimum it meets. The returned solution, however, is one up to 0.2% of the cut: greedy single-flip ascent adds 5 edges out of 3042 on G14 and 8 out of 13319 on G22, against 35 and 240 without the recurrent feature. On G49, 89% of the visited iterates are local optima and the returned solution is one exactly. The number of distinct configurations within 1% of the best cut found in these runs is 846–981, 7993 and 2280 with the recurrent feature, and 0 without it, on the three graphs respectively.

Mechanism statistics (Section 6). On G14 (3×10^4 iterations, 3 seeds) the margin $|h_i|$ of a flipping node, relative to the median margin, is 0.36 eight iterations before the flip and 0.13 one iteration before it, while non-flipping nodes stay above 1; for the non-recurrent variant the curve is flat at 0.98. With the rounded solution fed back instead of the logits the precursor weakens to 0.60, the rounded solution stands still in 93% of the iterations, and the cut drops from 3042 to 2871 (non-recurrent: 2954). With logit feedback, 20% of the iterations contain a multi-node move whose gain exceeds the best single flip available from the previous state, and the flipped nodes are $1.3\times$ more connected to each other than random node sets of the same size ($0.8\times$ for the non-recurrent variant, whose rounded output changes at every iteration).

Table 4: Structural diagnostics of 10^5-iteration runs with (*rec*) and without (*no rec*) the recurrent feature. *Onset*: first iteration with $> 99\%$ saturated probabilities. *Gap*: median relative difference between the relaxed objective and the discrete cut after the onset. *Drops*: fraction of snapshot steps in which the current cut decreases. *Best at*: iteration of the returned solution. *1-flip gain*: cut gained by greedy single-flip ascent from the returned solution, relative to it. *Near-best*: number of distinct visited configurations within 1% of the best cut found on the graph. On `G14` the two seeds are given as a range.

GRAPH	VARIANT	ONSET	GAP (%)	DROPS (%)	BEST AT	1-FLIP GAIN (%)	NEAR-BEST
G14	REC	60–70	0.007	47	20K–41K	0.03–0.16	846–981
G14	NO REC	130–320	0.05	49	23K–61K	0.8–1.2	0
G22	REC	80	0.002	48	19K	0.06	7993
G22	NO REC	1030	0.09	49	94K	1.9	0
G49	REC	30	0.000	9	2K	0.0	2280
G49	NO REC	620	0.06	46	72K	0.0	0

D ADDITIONAL EXPERIMENTS

D.1 MAX-CUT

BA baseline sources and time conventions. The Max-Cut numbers on Barabási–Albert graphs in main Table 1a (left panel of Table 1) are sourced as follows. GUROBI, SDP, EO and GFlowNet are reproduced from (Zhang et al., 2023) on the same 500-graph BA-[200-300] and BA-[800-1200] sets. The four amortised baselines Tabu Search (Nath & Kuhnle, 2025), ANYCSP (Tönshoff et al., 2023), DiffUCO (Sanokowski et al., 2024) and X^{2}GNN (Acikalin et al., 2025) are taken from (Acikalin et al., 2025), Table 4; the times there are per-graph wall-clock, which we rescale to total wall-clock over the 500 test instances of each size set so that all rows are directly comparable as total time budgets. PI-GNN, CRA and QRF-GNN are run on the same 500 instances per size set and reported in total wall-clock too.

Gset baseline run-protocols. The Gset comparison in Table 2 mixes baselines with heterogeneous best-of-N conventions: BLS and TSHEA report the best of 20 independent runs (Benlic & Hao, 2013; Wu et al., 2015); RUN-CSP takes the best of 64 runs (for G70 we re-ran the authors' code[3] with the parameters of (Tönshoff J & M, 2021)); PI-GNN uses graph-specific hyperparameter tuning per instance (Schuetz et al., 2022a); EO and CRA each report a single best across multiple runs. Our τ-EO reimplementation follows (Boettcher & Percus, 2001) with τ=1.3 and at most 10^7 single-spin updates; we run 20 independent restarts per graph to partially compensate for the limited budget (the optimal scaling suggested by the authors, $\sim O(|V|^3)$, is infeasible here). Within this budget, τ-EO takes ~ 6800 s per run for G70. For QRF-GNN we run 20 independent seeds per graph and report both their best (matching the baselines' best-of-N convention) and their empirical mean $\pm$ standard deviation (an estimate of typical single-run quality); the seeds are independent and parallelisable on a single GPU at no extra wall-clock cost.

For synthetic instances we use the number of cut edges as evaluation metric. For random regular graphs with $n \to \infty$ there is a theoretical estimate of the maximum cut size that depends on n, vertex degree d and a universal constant $P_* \simeq 0.7632$, which motivates the *P-value* $= \sqrt{\frac{4}{d}}\left(\frac{z}{n} - \frac{d}{4}\right)$, where z is the obtained cut size (Yao et al., 2019). Higher P-values indicate better cuts, with P_* as an upper bound on optimality.

Table 5 reports average P-values over 200 random d-regular graphs with $n = 500$ for QRF-GNN, RUN-CSP, and extremal optimisation (EO) (Boettcher, 2003). For QRF-GNN and PI-GNN we report the best of 5 runs; PI-GNN uses the GCN-based setup from (Schuetz et al., 2022a) for $d > 5$, with other values from the original source. EO and RUN-CSP numbers are from (Tönshoff J & M, 2021), where RUN-CSP used 64 runs per instance. QRF-GNN consistently outperforms RUN-CSP and PI-GNN and achieves the best result overall for $d \geq 5$.

Table 5: *P-value* of EO, PI-GNN, RUN-CSP and QRF-GNN for d-regular graphs with 500 nodes and different degree d averaged over 200 randomly generated instances.

d	HEUR	UNSUPERVISED LEARNING		
	EO	**PI-GNN**	**RUN-CSP**	**QRF-GNN**
3	**0.727**	0.612	0.714	0.725
5	0.737	0.608	0.726	**0.738**
10	0.735	0.658	0.710	**0.737**
15	0.736	0.644	0.697	**0.739**
20	0.732	0.640	0.685	**0.735**

In Table 5, EO and RUN-CSP results are taken from (Tönshoff J & M, 2021) (where *P-value* was averaged over 1000 graphs, RUN-CSP used 64 runs per graph, and EO the best of two (Yao et al., 2019)). PI-GNN values depend on the architecture: those for $d = 3$ and 5 are from (Schuetz et al., 2022a) with a GCN layer (cut sizes bootstrap-averaged over 20 random graphs, with up to 5 shots).

[3] https://github.com/toenshoff/RUN-CSP

(Schuetz et al., 2023) shows that switching to a SAGE layer improves results on $d = 3$ by 10.78%, but we observed no improvement over GCN on higher-degree graphs.

Full Gset benchmark. Table 6 reports cut sizes obtained by QRF-GNN on 36 Gset instances, together with three classical heuristics (BLS (Benlic & Hao, 2013), DSDP, KHLWG), the amortised learning solvers X^{2}GNN (Acikalin et al., 2025) and ANYCSP at two compute budgets each (64×256 and 64×2560, in the notation of (Acikalin et al., 2025)), and the instance-wise unsupervised baselines RUN-CSP (Tönshoff J & M, 2021), PI-GNN (Schuetz et al., 2022a) and CRA (Ichikawa, 2024). Numbers for BLS, DSDP, KHLWG, X^{2}GNN and ANYCSP are reproduced from (Acikalin et al., 2025), Table 12; numbers for RUN-CSP, PI-GNN and CRA are reproduced from (Ichikawa, 2024), Table 6, where they are reported as approximation ratios with respect to BLS. We multiply each ratio by the BLS column to recover absolute cut sizes.

Table 6: Number of cut edges on the full Gset benchmark (36 instances; $|V|$ nodes, $|E|$ edges). BLS, DSDP, KHLWG, X^{2}GNN and ANYCSP are reproduced from (Acikalin et al., 2025), Table 12; RUN-CSP, PI-GNN and CRA are reproduced from (Ichikawa, 2024), Table 6 (ApR with respect to BLS, multiplied by the BLS column to recover absolute cuts; rounded to the nearest integer). *Time (s)* is QRF-GNN's wall-clock per graph in seconds. A dash (–) indicates that no value is reported by the corresponding source.

Graph	$\|V\|$	$\|E\|$	Classical heuristics			X^{2}GNN		ANYCSP		UL_{tw}			QRF-GNN	Time (s)
			BLS	DSDP	KHLWG	64×256	64×2560	64×256	64×2560	RUN-CSP	PI-GNN	CRA	(Ours)	
G1	800	19176	11624	–	11624	11583	11598	11582	11616	11380	11368	11624	11624	42.9
G2	800	19176	11620	–	11620	11575	11588	11585	11601	11399	11341	11597	11608	42.6
G3	800	19176	11622	–	11620	11583	11591	11581	11614	11413	11297	11622	11616	42.9
G4	800	19176	11646	–	11646	11597	11617	11612	11633	11413	11390	11634	11608	42.7
G5	800	19176	11631	–	11631	11596	11594	11583	11623	11398	11375	11631	11622	43.1
G14	800	4694	3064	2922	3061	3054	3058	3021	3035	2929	3027	3046	3056	39.7
G15	800	4661	3050	2938	3050	3043	3036	3011	3024	2904	2989	3026	3048	39.9
G16	800	4672	3052	–	3052	3040	3045	3011	3024	2909	2945	3021	3040	39.7
G17	800	4667	3047	–	3046	3037	3039	3002	3018	2913	2946	3017	3037	39.7
G22	2000	19990	13359	12960	13359	13339	13347	13263	13285	12985	13185	13332	13330	39.0
G23	2000	19990	13344	13006	13342	13319	13329	13247	13309	12984	12917	13304	13316	39.1
G24	2000	19990	13377	12933	13337	13310	13319	13273	13276	13016	12829	13350	13350	39.2
G25	2000	19990	13340	–	13332	13311	13327	13248	13287	12993	12993	13313	13340	38.2
G26	2000	19990	13328	–	13328	13296	13306	13236	13272	12981	12862	13301	13320	38.8
G35	2000	11778	7684	–	7678	7648	7646	7543	7575	7323	7438	7607	7642	39.4
G36	2000	11766	7678	–	7670	7646	7654	7529	7566	7317	7417	7609	7626	39.9
G37	2000	11785	7689	–	7682	7665	7666	7546	7593	7305	7428	7666	7666	39.7
G38	2000	11779	7687	–	7683	7649	7658	7547	7570	7295	7426	7618	7644	39.7
G43	1000	9990	6660	–	6660	6655	6659	6622	6651	6500	6434	6627	6647	34.7
G44	1000	9990	6650	–	6639	6645	6646	6622	6631	6504	6437	6637	6639	34.9
G45	1000	9990	6654	–	6652	6640	6652	6625	6641	6514	6394	6641	6633	34.8
G46	1000	9990	6649	–	6649	6639	6644	6622	6637	6489	6476	6636	6631	34.8
G47	1000	9990	6657	–	6665	6651	6654	6626	6647	6497	6471	6637	6647	34.7
G48	3000	6000	6000	6000	6000	6000	6000	6000	6000	6000	5472	6000	6000	23.3
G49	3000	6000	6000	6000	6000	6000	6000	6000	6000	6000	5916	6000	6000	38.6
G50	3000	6000	5880	5880	5880	5872	5880	5846	5876	5874	5821	5880	5880	46.4
G51	1000	5909	3848	–	3847	3838	3837	3791	3810	3671	3709	3813	3837	35.4
G52	1000	5916	3851	–	3849	3835	3840	3792	3805	3678	3701	3812	3840	35.4
G53	1000	5914	3850	–	3848	3841	3844	3796	3812	3658	3719	3819	3829	35.3
G54	1000	5916	3852	–	3851	3842	3841	3791	3802	3667	3736	3806	3842	35.2
G55	4969	12498	10294	9960	10236	10241	10283	10066	10141	10068	10119	10201	10278	49.9
G58	5000	29570	19263	–	19248	19167	19186	18843	18937	18878	18608	19051	19192	63.1
G60	6957	17148	14176	13610	14057	14121	14146	13855	13952	13892	13396	14048	14155	52.8
G63	7000	41459	26997	8017	26963	26863	26905	26405	26512	25566	26133	26700	26948	68.5
G70	8646	9999	9541	9456	9458	9492	9525	9189	9290	9255	9369	9465	9548	52.6

Toy example of a Max-Cut problem. We use a toy Max-Cut instance with 10 nodes and 12 edges (Figure 1) to illustrate QRF-GNN against PI-GNN. QRF-GNN uses the default setup of Section 5; PI-GNN uses a trainable embedding layer and one hidden layer of width 50 to match QRF-GNN. Across 100 random seeds, QRF-GNN reaches the optimal cut ($= 12$) in 10.96 iterations on average, while PI-GNN needs 532.1, motivating the "$\sim50\times$" qualification used in Section 4.1.

D.2 MIS

Extended ER MIS comparison. Table 7 extends Table 1b on the canonical DIMES (Qiu et al., 2022) ER instances with the supervised and RL baselines we omit from the main table for space: Intel (Li et al., 2018), DIFUSCO (Sun & Yang, 2023), T2TCO (Li et al., 2023) (supervised) and LwD (Ahn et al., 2020) (RL). The gradient-based instance-wise samplers iSCO (Sun et al., 2023a) and ReSCO (Li & Zhang, 2025) in the main table are originally reported in two regimes (single chain and 32 parallel chains as a best-of-32); for fair single-solution comparison with QRF-GNN we use only the single-chain (S=1) variant. DIFUSCO and T2TCO could not scale to ER-[9000-11 000] (Zhang et al., 2023) and are omitted on that column.

Table 7: Additional ER MIS baselines complementing Table 1b; numbers are taken from the corresponding source papers as reported there. Times are total wall-clock over the entire test set.

METHOD	TYPE	ER-[700-800]		ER-[9000-11000]	
		SIZE	TIME	SIZE	TIME
INTEL	SL	34.86	6:04	284.63	5:02
DIFUSCO	SL	40.35	32:58	–	–
T2TCO	SL	41.37	29:44	–	–
LWD	RL	41.17	6:33	345.88	1:02:29

Penalty schedule. A small P leads to solutions with too many violations under a fixed iteration budget, while a large P causes rapid convergence to the trivial empty set. To avoid tuning P per graph type, we linearly increase it from 0.01 to 2 over all iterations: the algorithm starts in the space of large sets with violations and gradually narrows toward feasible sets.

RB-model graphs. Following (Tönshoff J & M, 2021) we also evaluate on the RB-model graphs of (Xu & Li, 2006), which are hard instances with hidden optima. Table 8 reports MIS sizes (mean $\pm$ std over 5 seeds) for QRF-GNN, the GNN baseline RUN-CSP, a greedy heuristic, and the classical solver KaMIS. QRF-GNN outperforms RUN-CSP and greedy on all four RB instances and approaches KaMIS.

Table 8: MIS results of QRF-GNN, the GNN baseline RUN-CSP, a greedy baseline, and the SOTA classical solver KaMIS on RB-model graphs (Xu & Li, 2006). Average MIS sizes over 5 runs with standard deviations.

GRAPH	$\lvert V\rvert$	$\lvert E\rvert$	HEURISTICS		UNSUPERVISED LEARNING	
			KAMIS	GREEDY	RUN-CSP	QRF-GNN
FRB30-15	450	18K	30 ± 0.0	24.6 ± 0.5	25.8 ± 0.8	28.4 ± 0.4
FRB40-19	790	41K	39.4 ± 0.5	33.0 ± 1.2	33.6 ± 0.5	36.8 ± 0.7
FRB50-23	1150	80K	48.8 ± 0.4	42.2 ± 0.8	42.2 ± 0.4	45 ± 0.6
FRB59-26	1478	126K	57.4 ± 0.9	48.0 ± 0.7	49.4 ± 0.5	54.6 ± 1

Structured MIS benchmark. Table 9 reports MIS sizes and per-graph runtimes on the benchmark of 32 structured instances (groups 1dc, 1et, 1tc, 1zc, 2dc; $|V|$ from 8 to $4{,}096$, $|E|$ from 6 to $504{,}451$) used by (Acikalin et al., 2025), Table 8. For each instance we report the best-known size BKS, KaMIS (Lamm et al., 2017), X^{2}GNN (Acikalin et al., 2025) at its 64×2560 search budget, and QRF-GNN. Baseline numbers are taken from (Acikalin et al., 2025), Table 8; bold entries match BKS.

X^{2}GNN on the canonical ER-[9000-11 000] test set. The X^{2}GNN paper (Acikalin et al., 2025) releases three pretrained MIS checkpoints: `er750.pt` (trained on ER-[700-800]), `rb200-300.pt` (RB-250) and `rb800-1200.pt` (RB-1000). *No checkpoint is released for $n \approx 10\,000$ ER graphs*, and the authors do not evaluate X^{2}GNN on the canonical 16 ER-[9000-11 000] instances of (Qiu et al., 2022) we use in Table 1b. To still produce a fair X^{2}GNN entry, we adopt the same out-of-distribution convention X^{2}GNN itself uses elsewhere (Table 7 of (Acikalin et al., 2025) evaluates `rb800-1200.pt` on DIMACS Max Clique graphs of size 125–4000; Table 8 on coding-theory MIS instances of size 8–4096; Table 12 trains on ER-200–500 and evaluates on Gset up to 10 000). Concretely, we run *both* closest checkpoints, `er750.pt` (matching distribution but $12\times$ smaller than the test) and `rb800-1200.pt` (different distribution but matching size), on the 16 ER-[9000-11 000] instances, with the two configurations from the official `mis/pretrained.sh`: 8×256 (k_2=8 hypotheses, 256 recurrence steps) and 32×256 (also used in Tables 7–8 of the paper). All runs use a single H100-class GPU with the authors' default thresholding.

Table 10 reports the aggregate results. The same-distribution checkpoint `er750.pt` collapses to $\sim$245 ($\approx$ 35% below KaMIS) and the additional refinement of 32×256 does not help: the model converges to a small basin and stops moving. The matching-size checkpoint `rb800-1200.pt`

Table 9: MIS sizes and per-graph wall-clock times (in seconds) on the structured MIS benchmark of 32 instances used by (Acikalin et al., 2025) (Table 8). BKS denotes the best-known solution size. KaMIS and X^{2}GNN columns (the latter at the 64×2560 search budget) are reproduced from (Acikalin et al., 2025), Table 8.

Graph	$\lvert V\rvert$	$\lvert E\rvert$	BKS	KaMIS		X^{2}GNN (64×2560)		QRF-GNN (Ours)	
				Size	Time	Size	Time	Size	Time
1dc.64	64	543	10	**10**	5.1	**10**	5.1	**10**	18.0
1dc.128	128	1471	16	**16**	4.0	**16**	3.9	**16**	16.9
1dc.256	256	3839	30	**30**	4.5	**30**	4.4	**30**	15.8
1dc.512	512	9727	52	**52**	6.5	**52**	6.3	**52**	22.0
1dc.1024	1024	24063	94	93	11.4	**94**	11.3	**94**	23.7
1dc.2048	2048	58367	172	**172**	24.4	**172**	24.3	**172**	30.8
1et.64	64	264	18	**18**	1.3	**18**	3.8	**18**	12.3
1et.128	128	672	28	**28**	3.9	**28**	3.8	**28**	16.7
1et.256	256	1664	50	**50**	3.9	**50**	3.8	**50**	17.4
1et.512	512	4032	100	**100**	4.7	98	4.7	97	28.2
1et.1024	1024	9600	171	**171**	7.2	165	7.1	167	27.5
1et.2048	2048	22528	316	**316**	14.0	300	14.0	311	26.1
1tc.8	8	6	4	**4**	0.5	**4**	3.7	**4**	4.9
1tc.16	16	22	8	**8**	0.8	**8**	3.8	7	4.9
1tc.32	32	68	12	**12**	0.3	**12**	3.8	**12**	5.2
1tc.64	64	192	20	**20**	1.1	**20**	3.8	**20**	8.8
1tc.128	128	512	38	**38**	1.2	**38**	3.8	37	13.0
1tc.256	256	1312	64	63	3.9	63	3.8	61	17.4
1tc.512	512	3264	110	**110**	4.5	109	4.4	**110**	21.0
1tc.1024	1024	7936	196	**196**	6.7	189	6.6	192	22.9
1tc.2048	2048	18944	352	**352**	13.0	332	12.9	339	25.8
1zc.128	128	1120	18	**18**	3.9	**18**	3.8	16	18.7
1zc.256	256	2816	36	**36**	4.1	**36**	4.0	**36**	19.6
1zc.512	512	6912	62	**62**	5.6	**62**	5.5	**62**	21.6
1zc.1024	1024	16640	112	**112**	9.3	109	9.1	108	26.7
1zc.2048	2048	39424	198	195	18.9	181	18.8	173	35.4
1zc.4096	4096	92160	379	353	40.4	326	40.2	354	66.0
2dc.128	128	5173	5	**5**	1.5	**5**	4.5	**5**	13.7
2dc.256	256	17183	7	**7**	10.1	**7**	8.2	6	16.4
2dc.512	512	54895	11	**11**	22.3	**11**	19.1	**11**	38.2
2dc.1024	1024	169162	16	**16**	57.8	15	52.7	15	80.9
2dc.2048	2048	504451	24	**24**	162.7	11	152.6	22	263.2

is much stronger and reaches ≈ 346, $\approx 8\%$ below KaMIS and QRF-GNN. We therefore quote the stronger `rb800-1200.pt` (32×256) result, 346.56, as the X^{2}GNN entry for the ER-[9000-11 000] column of Table 1b, with a footnote indicating the OOD setting.

Table 10: X^{2}GNN on the canonical 16 ER-[9000-11 000] instances of (Qiu et al., 2022) (mean $\pm$ standard deviation across the 16 graphs; per-graph wall-clock time on a single H100-class GPU). The `er750.pt` checkpoint matches the test *distribution* but is trained on $12\times$ smaller graphs and collapses; the `rb800-1200.pt` checkpoint matches the test *size* but is trained on the RB distribution. We report both configurations from the official `mis/pretrained.sh`. For comparison, KaMIS and QRF-GNN entries are taken from Table 1b.

Method (config)	mean MIS	std	time / graph
X^{2}GNN `er750.pt` (8×256)	245.38	10.40	28.2 s
X^{2}GNN `er750.pt` (32×256)	251.69	9.84	108.3 s
X^{2}GNN `rb800-1200.pt` (8×256)	345.62	5.43	28.3 s
X^{2}GNN `rb800-1200.pt` (32×256)	**346.56**	**4.17**	108.2 s
KaMIS (from Table 1b)	381.31	–	$\approx$1710 s
QRF-GNN (from Table 1b)	381.1	–	330.6 s

D.3 Coloring

QRF-GNN achieves the best result on every chromatic-number-colouring instance (outperforming HybridEA on several), and is the strongest GNN on the smallest-k formulation, losing only to HybridEA on one instance.

Table 11: Monochromatic edges at $k = \chi$.

Graph	heur	Unsupervised Learning			
	HEA	**GNN-1N**	**PI-GNN**	**GDN**	**QRF-GNN**
homer	**0**	**0**	**0**	**0**	**0**
myciel6	**0**	**0**	**0**	**0**	**0**
queen5-5	**0**	**0**	**0**	**0**	**0**
queen6-6	**0**	**0**	**0**	**0**	**0**
queen7-7	**0**	**0**	**0**	9	**0**
queen8-8	**0**	1	1	-	**0**
queen9-9	**0**	1	1	-	**0**
queen8-12	**0**	**0**	**0**	**0**	**0**
queen11-11	14	13	17	21	**7**
queen13-13	18	15	26	33	**15**
Cora	**0**	1	**0**	**0**	**0**
Citeseer	**0**	**0**	**0**	**0**	**0**
Pubmed	**0**	-	17	21	**0**

Table 12: Smallest k admitting a valid colouring; χ is the known chromatic number.

Graph	χ	heur	Unsupervised Learning		
		HEA	**PI-GNN**	**RUN-CSP**	**QRF-GNN**
homer	13	**13**	**13**	17	**13**
myciel6	7	**7**	**7**	8	**7**
queen5-5	5	**5**	**5**	**5**	**5**
queen6-6	7	**7**	**7**	8	**7**
queen7-7	7	**7**	**7**	10	**7**
queen8-8	9	**9**	10	11	**9**
queen9-9	10	**10**	11	17	**10**
queen8-12	12	**12**	**12**	17	**12**
queen11-11	11	**12**	14	>17	**12**
queen13-13	13	**14**	17	>17	15
Cora	5	**5**	**5**	-	**5**
Citeseer	6	**6**	**6**	-	**6**
Pubmed	6	**8**	9	-	**8**

Per-method run protocols. HybridEA results use the implementation[4] based on (Lewis, 2021). GNN-1N on citation graphs was obtained by re-implementing the algorithm from the original paper; lacking hyperparameter tuning instructions, we took them close to PI-GNN and chose the best of 10 runs. For QRF-GNN we did up to 10 runs per graph (most required only one). To find the smallest k admitting a valid colouring (Table 12), we successively increased the number of colours until a valid colouring was found among 10 seeds. QRF-GNN convergence took up to 6000 iterations on citation graphs and ranged from ~ 200 to 9×10^4 on the COLOR dataset. Graphs anna, david, games120, myciel5, huck and jean were omitted from the main tables because all algorithms find violation-free solutions on them. Graph sizes are in Table 13.

Table 13: Number of Vertices and Edges in coloring graphs.

Graph	$\lvert V\rvert$	$\lvert E\rvert$
anna	138	493
david	87	406
games120	120	638
homer	561	1629
huck	74	301
jean	80	254
myciel5	47	236
myciel6	95	755
queen5-5	25	160
queen6-6	36	290
queen7-7	49	476
queen8-8	64	728
queen9-9	81	1056
queen8-12	96	1368
queen11-11	121	1980
queen13-13	169	3328
Cora	2708	5429
Citeseer	3327	4732
Pubmed	19717	44338

E Ablations

Other components. We also analyse which other components of QRF-GNN contribute most to Max-Cut performance. The default architecture has two intermediate SAGEConv layers and the recurrent dynamic feature; as Figure 8 already shows, dropping the recurrent feature causes the most dramatic quality drop.

[4] http://rhydlewis.eu/gcol/

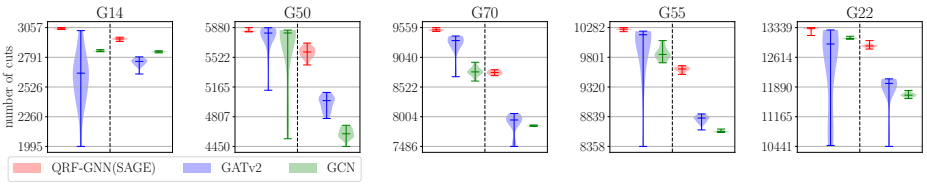

Figure 8: Distribution of cut sizes over 20 random seeds for three GNN backbones (**QRF-GNN (SAGE)**, the default; **GATv2**; **GCN**) on a representative subset of Gset instances. Within each panel the dashed vertical line separates runs *with* the recurrent feature (left) from runs *without* it (right); horizontal lines mark the max, median and min cut sizes.

Removing one of the intermediate convolutional layers is less harmful (Figure 10). Removing the pool-aggregation layer lowers the median, upper bound, and increases dispersion on almost all graphs. Removing the mean-aggregation layer can increase the median cut and even reduce variance in some cases, but the upper bounds either stay the same or decrease, even when we double the parameters of the remaining hidden layer. Further ablation on random regular graphs (Figure 11) shows that removing any convolutional layer hurts; Table 14 (single-seed and best-of-5 averages) confirms the benefit of the two-layer combination.

For input features we settled on a combination of a random vector, a shared vector (Cui et al., 2022) and PageRank (Brin & Page, 1998). Figure 9 shows results when removing the random or PageRank components: the median sometimes improves, but the upper bound generally decreases with fewer features. Using an embedding layer brings no benefit over the default.

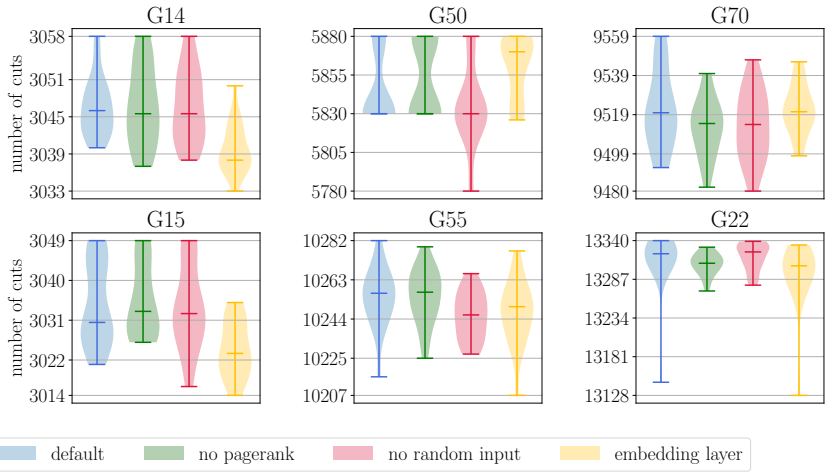

Figure 9: Results distribution for 20 runs of QRF-GNN with the default architecture on several instances from Gset. Input feature vectors varied as follows: the default choice corresponds to the blue color; the exclusion of the pagerank component corresponds to the green color; the exclusion of the random part corresponds to the red color. The use of a trainable embedding layer instead of artificial features is indicated in yellow.

Table 14: The first row contains the average *P-value* over 200 random regular graphs with $d = 5$ for 1 run of QRF-GNN with same configurations as in Figures 10 and 11. The second row shows the average *P-value* over 200 graphs when the best cut out of 5 runs is taken.

Runs	**Default**	**no SAGEConv (mean)**	**no SAGEConv (pool)**
1	0.734	0.732	0.725
5	0.738	0.737	0.734

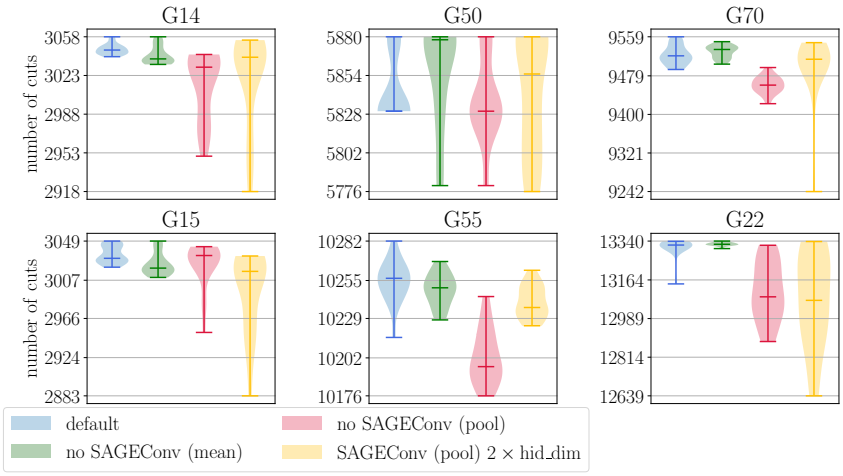

Figure 10: Results distribution for 20 runs of QRF-GNN of the default architecture (blue), with the absence of one SAGE layer with a mean aggregation function (green) or the SAGE layer with a pool aggregation function (red). Horizontal lines mean maximum, median and minimum values.

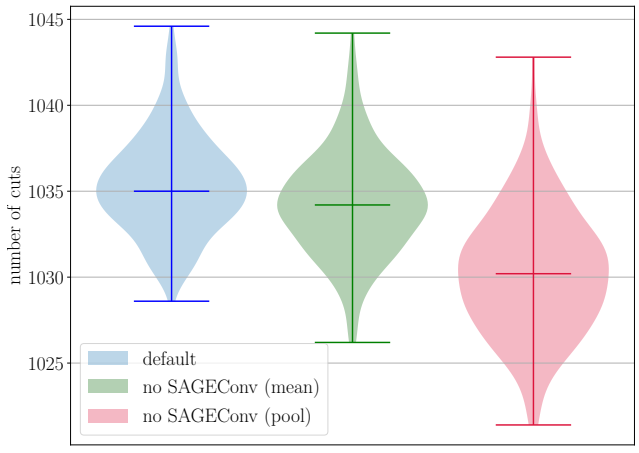

Figure 11: Results distribution for the mean of 5 runs on 200 random regular graphs with $d = 5$ and 500 nodes. QRF-GNN had the same configurations as in the Fig. 10.