Learning to Destroy and Repair Local Optima in Multilinear Graph QUBO Optimization

Anonymous authors
Paper under double-blind review

Abstract

Graph combinatorial optimization problems such as Maximum Independent Set and Maximum Cut can be written as pseudo-Boolean quadratic objectives with multilinear polynomial forms. Keeping the same multilinear objective while relaxing the variable domain from $\{0,1\}^n$ to $[0,1]^n$ preserves all 0–1 global minimizers, and every strict local minimizer of the relaxed objective is still 0–1. This exactness makes it possible to use continuous gradient-based optimization while still targeting discrete solutions. However, the relaxed landscape remains nonconvex: direct gradient optimizers such as Adam may require many iterations to stabilize near a local optimum, while amortized neural solvers trained on the relaxation can be faster but may still settle at suboptimal binary solutions. We propose LDR, a learned destroy-and-repair framework for improving such solutions. A recurrent repair network is trained by the multilinear QUBO relaxation to construct and repair continuous states that are decoded into feasible solutions of the original discrete problem. A destroy policy then masks part of the repaired continuous state before repair is applied again, and is trained with the repair network as its environment using post-repair improvement in the decoded discrete objective as a graph-level reward. The policy learns a graph- and state-dependent destruction rate; at inference time, LDR samples a random destruction mask over the vertices eligible to be perturbed using that learned rate. The repair and destroy modules can be alternated to calibrate the perturbation scale and expose repair to damaged continuous states it will see at inference time. We give a possible explanation for why destroy-and-repair can help hard-exclusion tasks such as MIS, and evaluate the method on MIS and MaxCut. On these two tasks, destroy-and-repair improves the decoded solution produced by the base repair network, with additional inference cost that we report explicitly.

1 Introduction

Many graph combinatorial optimization problems can be expressed as quadratic objectives over binary variables. Maximum Independent Set (MIS) selects vertices with no selected edge, and Maximum Cut (MaxCut) partitions vertices to maximize the number or weight of crossing edges. These problems are NP-hard and arise in applications such as scheduling and network design, biological and social network analysis, statistical physics, and circuit layout (Garey & Johnson, 1979; Nemhauser & Wolsey, 1988; Korte & Vygen, 2018; Barahona et al., 1988). They also serve as benchmark tasks for both classical combinatorial optimization and neural optimization methods (Bengio et al., 2021). Maximum Clique has the same hard-exclusion form as MIS after taking the complement graph; we mention this equivalence for completeness but do not report separate MaxClique experiments.

Neural combinatorial optimization has developed along several lines, including supervised construction policies, reinforcement-learning solvers, unsupervised energy minimization, GFlowNet-style samplers, and diffusion-based solvers (Khalil et al., 2017; Ahn et al., 2020; Qiu et al., 2022; Zhang et al., 2023; Sun & Yang, 2023; Sanokowski et al., 2024; 2025). The unsupervised line is particularly relevant here because it does not require labeled optimal solutions: Erdős Goes Neural learns distributions over feasible sets (Karalias & Loukas, 2020), physics-inspired and principled-relaxation methods train GNNs directly from differentiable graph energies (Schuetz et al., 2022; Wang et al., 2022), and recent diffusion-style methods optimize or

sample discrete solutions without supervised labels (Sanokowski et al., 2024; 2025). We focus on pseudo-Boolean QUBO objectives whose natural relaxation is the multilinear extension of the discrete objective. This extension has a useful exactness property: the discrete and relaxed problems share the same 0–1 global minimizers, and strict local minima of the relaxed objective are still binary. In principle, one could apply direct gradient optimization such as Adam to the relaxed variables, but many iterations may be needed before the trajectory stabilizes near a local optimum. Amortized neural solvers reduce this inference cost by learning a graph-to-solution map, yet preserving optima does not imply finding them: solvers trained on relaxed objectives can repeatedly converge to stable binary or near-binary local solutions that are still suboptimal.

The obstacle addressed in this paper is the structure of improvement in the relaxed QUBO landscape. For hard-exclusion tasks, a maximal independent set cannot be improved by adding a single vertex; selected vertices must be removed before alternative unselected vertices can enter. For MaxCut, all binary assignments are feasible, but improving a local partition can still require coordinated changes to several correlated vertices. In both cases, repeatedly decoding the same neural preference or applying undirected random perturbations can waste computation on changes that the repair solver immediately reverses.

We propose LDR (**L**earned **D**estroy and **R**epair), an unsupervised framework for graph QUBO optimization. The repair network constructs a local continuous state from noise and repairs damaged states, using the original multilinear QUBO relaxation as its training objective. Since the multilinear objective has no strict fractional local minimizer, stabilized repair outputs are encouraged to polarize before decoding. The destroy network observes the graph, current continuous state, and search step, predicts one Bernoulli destruction probability per node, and is trained by post-repair score improvement. Because this reward is graph-level, default inference averages these probabilities over eligible vertices to obtain a learned destruction rate, then samples a random mask at that rate.

This paper makes the following contributions:

- We formulate iterative improvement for graph QUBO solvers under the original discrete task score, covering hard-exclusion subset selection and unconstrained partitioning.
- We introduce LDR, a learned destroy-and-repair framework that trains repair by a multilinear continuous objective and trains destroy by post-repair improvement rewards.
- We analyze why multilinear relaxations still produce difficult nonconvex landscapes and provide MIS and MaxCut experiments with explicit runtime and ablations.

2 Background and Problem Formulation

2.1 Graph QUBO Objectives

Let $G = (V, E)$ be an undirected graph with $n = |V|$. A binary vector $\boldsymbol{x} \in \{0,1\}^n$ indicates which vertices are selected.

For MIS, we use the standard penalized QUBO energy

$$H_\lambda^{\mathrm{MIS}}(\boldsymbol{x}) = \lambda \sum_{(i,j)\in E} x_i x_j - \sum_{i\in V} x_i, \qquad \lambda > 1. \tag{1}$$

The linear term rewards large selected sets, while the quadratic term penalizes selecting adjacent vertices. When $\lambda > 1$, every infeasible selected edge can be improved by deleting one endpoint, so minimizers correspond to maximum independent sets.

MaxClique is the same hard-exclusion objective on the complement graph. Let $\bar{E} = \{(i,j) : i < j, (i,j) \notin E\}$ be the set of missing edges. The corresponding objective is

$$H_\lambda^{\mathrm{MaxClique}}(\boldsymbol{x}) = \lambda \sum_{(i,j)\in \bar{E}} x_i x_j - \sum_{i\in V} x_i, \qquad \lambda > 1. \tag{2}$$

This has the same mathematical form as MIS after replacing conflicts by missing edges. We include this formulation to clarify the scope of the analysis, but the present empirical section reports MIS and MaxCut only.

For MaxCut, the binary variable has a different meaning: x_i denotes the side of the cut containing vertex i. For edge weights $w_{ij} \geq 0$, the cut value is

$$C^{\text{MaxCut}}(\boldsymbol{x}) = \sum_{(i,j)\in E} w_{ij} \left[x_i(1-x_j) + x_j(1-x_i)\right]. \tag{3}$$

We minimize the negative cut value,

$$H^{\text{MaxCut}}(\boldsymbol{x}) = -\sum_{(i,j)\in E} w_{ij} \left[x_i(1-x_j) + x_j(1-x_i)\right], \tag{4}$$

which is also a quadratic pseudo-Boolean objective. Unlike MIS, every binary assignment is feasible for MaxCut; the challenge is optimizing a partition objective rather than satisfying hard exclusion constraints.

2.2 Multilinear Continuous Relaxation

We relax the discrete objective directly by replacing binary variables with continuous variables $\boldsymbol{y} \in [0,1]^n$ in its multilinear pseudo-Boolean form. If a QUBO is written with diagonal quadratic terms, those terms are first absorbed into the linear part using $x_i^2 = x_i$ on $\{0,1\}^n$. For MIS, the resulting relaxation is

$$H_\lambda^{\text{MIS}}(\boldsymbol{y}) = \lambda \sum_{(i,j)\in E} y_i y_j - \sum_{i\in V} y_i, \qquad \boldsymbol{y} \in [0,1]^n. \tag{5}$$

The same replacement applies to MaxCut using Eq. equation 4. More generally, consider a pseudo-Boolean objective whose polynomial representation has been written in multilinear form on $\{0,1\}^n$. Its continuous relaxation is the multilinear extension

$$F(\boldsymbol{y}) = \mathbb{E}_{\boldsymbol{Z}\sim\prod_i \text{Bern}(y_i)} \left[H(\boldsymbol{Z})\right]. \tag{6}$$

This probabilistic view is useful because F agrees with the discrete objective H on every binary vector and is affine in each coordinate when all other coordinates are fixed. These two facts imply that the relaxation preserves the binary optimum set, but they do not make the continuous problem convex. The following theorem records both the exactness property and the binary nature of strict local minima.

Theorem 1 (0–1 optima and strict local minima of multilinear relaxations)**.** *Let $H : \{0,1\}^n \to \mathbb{R}$ be a pseudo-Boolean objective represented by its multilinear polynomial, and let $F : [0,1]^n \to \mathbb{R}$ be its multilinear extension defined by Eq. equation 6. Then*

$$\min_{\boldsymbol{y}\in[0,1]^n} F(\boldsymbol{y}) = \min_{\boldsymbol{x}\in\{0,1\}^n} H(\boldsymbol{x}), \tag{7}$$

and the discrete problem and the relaxed problem share all global minimizers among 0–1 vectors: a binary vector is a global minimizer of H if and only if it is a global minimizer of F over $[0,1]^n$. Moreover, every strict local minimizer of F over $[0,1]^n$ is a 0–1 vector.

The proof is in Appendix C. This result explains why continuous neural optimization can yield nearly binary solutions without supervised labels. It also clarifies the remaining difficulty. For the quadratic multilinear objectives used here, the Hessian has zero diagonal because no term contains y_i^2, but it is not the zero matrix whenever the graph has at least one active pairwise interaction. A nonzero symmetric matrix with zero trace must have both positive and negative eigenvalues, so the relaxed objective is nonconvex. Consequently, there may be many stable 0–1 local solutions, and a neural solver trained on the relaxed objective can converge to a suboptimal one.

2.3 Local Improvement Objective

Let $\mathcal{X}_G \subseteq \{0,1\}^n$ denote the task-specific valid solution set and let $s_G(\boldsymbol{x})$ be the task score: independent-set size for MIS and cut value for MaxCut. For MaxCut, $\mathcal{X}_G = \{0,1\}^n$; for MIS, $\mathcal{X}_G$ contains only feasible selected subsets. Given a current binary solution $\hat{\boldsymbol{x}} \in \mathcal{X}_G$, the local-improvement problem is

$$\text{given } (G, \hat{\boldsymbol{x}}), \quad \text{find } \boldsymbol{x}' \in \mathcal{X}_G \text{ such that } s_G(\boldsymbol{x}') > s_G(\hat{\boldsymbol{x}}). \tag{8}$$

For the minimization forms in Eq. equation 1 and Eq. equation 4, the same goal is to find a binary solution with a smaller objective value. The current solution $\hat{\boldsymbol{x}}$ is not treated as a failed output to discard. It reveals the present solution structure, which constraints or pairwise interactions are active, and how large a perturbation the subsequent repair step may need.

3 Method

3.1 Overview

Throughout this section, $\boldsymbol{z}^t \in [0,1]^n$ denotes the continuous search state at outer destroy-repair iteration t. Given a graph G and the current state $\boldsymbol{z}^t$, LDR alternates two operations. A destroy policy D_ϕ predicts one Bernoulli destruction probability for each node from the current continuous graph state. Since the training reward is graph-level, LDR uses these probabilities through their average over the vertices eligible to be perturbed, obtaining a learned destruction rate ρ^t. It then samples a random binary mask $\boldsymbol{a}^t \in \{0,1\}^n$ with rate ρ^t, where $a_i^t = 1$ means that vertex i is perturbed in the continuous state. A repair network R_θ maps the damaged continuous state back to the next repaired continuous state $\boldsymbol{z}^{t+1}$. The decoded binary solution is used for reward computation, score logging, and selecting the best returned solution, but it is not fed back as the next destroy or repair input.

Figure 1 summarizes the method: repair is pretrained from the multilinear objective, destroy learns a state-dependent perturbation rate using repair as its environment, the two modules can be alternated, and inference repeats the rate-matched destroy-repair loop. The key design choice is that destroy is not trained to make the damaged state itself good. It is trained to make the repaired state better under the original task score. Since this score is observed only at the graph level, we do not assign node-specific rewards to individual mask entries. This separates two roles: destroy chooses how much of the current state to perturb, and repair reconstructs a high-quality solution from the damaged state.

3.2 Repair Network

The repair network is a learnable map from a graph and an input continuous state to a repaired continuous state. We use $\boldsymbol{y}^r$ for the internal repair state after r recurrent repair steps within one repair call; this index r is distinct from the outer destroy-repair iteration index t used for $\boldsymbol{z}^t$. Thus $\boldsymbol{y}^0$ is the input to repair, such as a random initial vector $\boldsymbol{u}$ or a damaged search state $\tilde{\boldsymbol{z}}^t$, and one repair step outputs one repaired value per vertex:

$$\boldsymbol{y}^{r+1} = R_\theta(G, \boldsymbol{y}^r) \in [0,1]^n. \tag{9}$$

The same map is applied recurrently, so a single set of parameters defines a multi-step repair process; $R_\theta^R(G, \boldsymbol{y}^0)$ denotes R such recurrent repair steps.

Repair is trained by minimizing the task's continuous multilinear energy. For a graph objective H_G, the per-graph relaxed loss is

$$\mathcal{L}_{\text{repair}}(\theta; G, \boldsymbol{y}) = F_G(\boldsymbol{y}) + \beta \sum_i y_i(1 - y_i), \tag{10}$$

where F_G is the multilinear extension of the discrete energy and the optional binary term encourages polarization. For MIS, F_G is Eq. equation 5; for MaxCut, F_G is the relaxation of Eq. equation 4. The MaxClique analogue is obtained from the MIS loss by replacing edges with missing edges.

The repair network optimizes continuous variables, but the objective geometry makes this a direct route to discrete repair. Since strict local minima of F_G are binary by Theorem 1, a stabilized recurrent repair

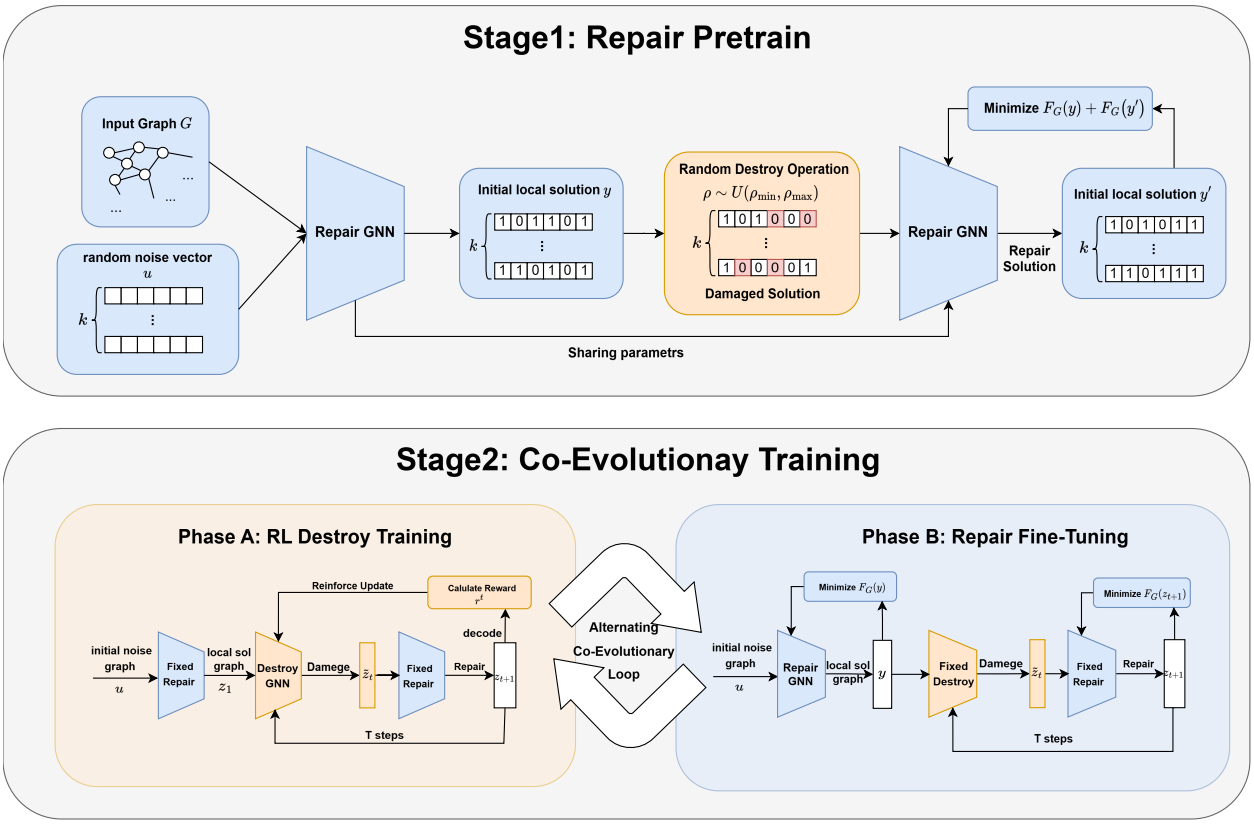

Figure 1: Overview of LDR. Stage 1 trains a recurrent repair network by the multilinear QUBO objective from both random initial states and randomly damaged continuous states. Stage 2 freezes repair and trains a graph-level destroy-rate policy whose sampled mask perturbs the current continuous state before repair. The destroy and repair stages can be alternated, and inference repeats the learned destroy-repair loop.

trajectory is encouraged to approach a 0–1 solution. We then use a fixed task decoder only when a discrete solution must be scored or returned: for MIS, vertices are greedily selected in descending repaired score while masking neighbors, producing a maximal independent set; for MaxCut, scores are thresholded to obtain a binary partition. Keeping this decoder fixed makes the learned component the destroy-and-repair dynamics rather than a changing post-processing heuristic.

Multi-source repair pretraining. Repair pretraining uses two sources of input states. First, the network maps a random initial continuous vector $\boldsymbol{u} \in [0,1]^n$ to a continuous state, giving it a base construction ability. Second, it receives randomly destroyed versions of its own outputs. Let $\boldsymbol{y} = R_\theta^R(G, \boldsymbol{u})$ be a current continuous state produced after R repair steps and let $\rho \in [\rho_{\min}, \rho_{\max}]$ be a sampled destruction rate. RandDestroy samples keep masks $m_i \sim \text{Bernoulli}(1-\rho)$ on active continuous coordinates, where activity means that vertex i has a large current state value, and returns $\boldsymbol{m} \odot \boldsymbol{y}$. For inactive coordinates, m_i is set to one so that random destruction focuses on variables that currently participate in the continuous repair state. The training loss sums the relaxed objective over recurrent repair outputs:

$$\mathcal{L}_{\text{pre}}(\theta) = F_G(R_\theta^R(G, \boldsymbol{u})) + F_G(R_\theta^R(G, \text{RandDestroy}(R_\theta^R(G, \boldsymbol{u})))). \tag{11}$$

This stage, shown as Stage 1 in Figure 1, is designed to make repair useful on partial states before it is paired with a learned destroy policy. It also makes the early training environment more stable than learning destroy against an untrained repair module. Algorithm 1 summarizes the repair pretraining procedure.

Algorithm 1 Repair Pretraining

Require: Training graphs $\mathcal{G}$; repair network R_θ; recurrent steps R; destruction-rate range $[\rho_{\min}, \rho_{\max}]$

Initialize repair parameters θ
for each training iteration **do**
  Sample a graph $G \in \mathcal{G}$ and an initial vector $\boldsymbol{u} \in [0,1]^n$
  Compute $\boldsymbol{y} = R_\theta^R(G, \boldsymbol{u})$
  Sample $\rho \sim \mathrm{Unif}[\rho_{\min}, \rho_{\max}]$
  Compute $\tilde{\boldsymbol{y}} = \textsc{RandDestroy}(\boldsymbol{y}; \rho)$
  Compute $\boldsymbol{y}' = R_\theta^R(G, \tilde{\boldsymbol{y}})$
  Update θ by minimizing $F_G(\boldsymbol{y}) + F_G(\boldsymbol{y}')$ with Eq. equation 10
end for
return pretrained repair parameters θ

3.3 Learned Destroy-Rate Policy

The destroy policy conditions on the graph, the current repaired state $\boldsymbol{z}^t$, and a scalar time or step feature τ_t. It outputs one Bernoulli destruction probability per node,

$$\boldsymbol{p}^t = D_\phi(G, \boldsymbol{z}^t, \tau_t) \in [0,1]^n, \tag{12}$$

Although Eq. equation 12 is written per node, the reward used to train it is graph-level: all mask entries for the same graph receive the same post-repair improvement signal. We therefore use the probabilities to choose the amount of destruction rather than to claim reliable node-specific credit. In both destroy training and the default LDR inference rule, the probabilities are averaged over the eligible vertex set $\mathcal{V}^t_{\text{elig}}$ defined for the current continuous state,

$$\rho^t = \frac{1}{|\mathcal{V}^t_{\text{elig}}|} \sum_{i \in \mathcal{V}^t_{\text{elig}}} p_i^t, \tag{13}$$

and the actual mask is sampled uniformly at that matched rate: $a_i^t \sim \text{Bernoulli}(\rho^t)$ for $i \in \mathcal{V}^t_{\text{elig}}$ and $a_i^t = 0$ otherwise. For MIS, the eligible set is defined from active coordinates of the current continuous state rather than from a decoded independent set; for MaxCut, all vertices are eligible. The nodewise sampling rule is kept as an ablation in the experiments, but it is not the default LDR inference rule. The damaged state for removal-style tasks is

$$\tilde{\boldsymbol{z}}^t = \boldsymbol{z}^t \odot (\boldsymbol{1} - \boldsymbol{a}^t). \tag{14}$$

For MaxCut, all vertices are eligible and the masked variables are flipped before repair. Repair then produces the next continuous state $\boldsymbol{z}^{t+1} = R_\theta^R(G, \tilde{\boldsymbol{z}}^t)$. Decoding is applied only to compute the post-repair score.

Reward. The policy reward is defined by improvement in the decoded task score:

$$r^t = s_G(\text{Decode}(\boldsymbol{z}^{t+1})) - s_G(\text{Decode}(\boldsymbol{z}^t)). \tag{15}$$

For MIS, s_G is the decoded independent-set size; for MaxCut, s_G is the decoded cut value. The decoded solutions in Eq. equation 15 are used for scoring only; the trajectory itself remains a sequence of continuous states. This reward directly matches the local-improvement objective in Eq. equation 8.

Policy optimization. Destroy is trained with a REINFORCE estimator (Williams, 1992; Sutton & Barto, 1998). For each graph, a training episode follows the same continuous rollout used at inference time: repair first constructs $\boldsymbol{z}^0 = R_\theta^R(G, \boldsymbol{u})$ from a random initial vector, and each subsequent destroy action is applied to the latest repair output $\boldsymbol{z}^t$. With repair frozen, this defines a finite-horizon environment of length T. The return is

$$G^t = \sum_{s=t}^{T-1} r^s. \tag{16}$$

Using a baseline b^t, the destroy loss is

$$\mathcal{L}_{\text{destroy}}(\phi) = -\sum_{t=0}^{T-1} \log \pi_\phi(\boldsymbol{a}^t \mid G, \boldsymbol{z}^t, \tau_t)(G^t - b^t). \tag{17}$$

Here π_ϕ denotes the rate-matched mask distribution induced by Eq. equation 13. Gradients update destroy only; repair is frozen during this stage. Because the same return multiplies the log-probabilities of all mask decisions in a trajectory, the learned probabilities tend to encode how aggressive destruction should be for the current graph state and time step. The inference rule in Eq. equation 13 deliberately uses this learned rate while avoiding an unsupported claim that graph-level rewards identify which individual nodes deserve larger credit.

3.4 Alternating Co-Evolution

Training is organized into one pretraining phase followed by the alternating destroy-training and repair-fine-tuning loop in Figure 1. Algorithm 1 first trains repair on random and randomly damaged states. After that, both alternating sub-stages use the same rollout structure as iterative inference. An episode starts with repair constructing $\boldsymbol{z}^0 = R_\theta^R(G, \boldsymbol{u})$ from noise; for $t = 0, \ldots, T-1$, destroy observes $\boldsymbol{z}^t$, samples a rate-matched mask, perturbs the continuous state, and repair maps the damaged state to $\boldsymbol{z}^{t+1}$. The two sub-stages differ only in which module is frozen and which loss is optimized. When repair is frozen, destroy is updated by the policy-gradient objective in Eq. equation 17 using decoded score improvements. When destroy is frozen, repair is fine-tuned on the same kind of destroy-repair trajectory by applying the relaxed repair loss to every repaired state in the trajectory,

$$\mathcal{L}_{\text{ft}}(\theta; G) = \sum_{t=0}^{T} \mathcal{L}_{\text{repair}}(\theta; G, \boldsymbol{z}^t). \tag{18}$$

Thus the initial constructed state $\boldsymbol{z}^0$ and all post-destroy repair outputs contribute to repair fine-tuning. In both sub-stages, the mask is computed from the continuous repair state rather than from a decoded solution; decoding is used only for the reward in destroy training. The purpose is to reduce the distribution shift between random destruction during pretraining and learned-rate destruction during search.

3.5 Iterative Inference

At test time, LDR follows the inference loop in Figure 1: it starts from an initial continuous vector $\boldsymbol{u} \in [0,1]^n$, obtains the initial search state by repair, and then repeats learned rate-matched destroy-and-repair for a fixed budget. Here t indexes outer refinement iterations:

$$\boldsymbol{z}^0 = R_\theta^R(G, \boldsymbol{u}), \tag{19}$$

$$\boldsymbol{z}^{t+1} = R_\theta^R\big(G, \text{Perturb}(\boldsymbol{z}^t, \boldsymbol{a}^t)\big), \quad a_i^t \sim \text{Bernoulli}(\rho^t), \quad \rho^t = \frac{1}{|\mathcal{V}_{\text{elig}}^t|} \sum_{i \in \mathcal{V}_{\text{elig}}^t} D_{\phi,i}(G, \boldsymbol{z}^t, \tau_t). \tag{20}$$

Here Perturb acts on the continuous state $\boldsymbol{z}^t$. Destroy never receives a decoded solution during the loop; decoding is used only to evaluate each repaired state and to maintain the returned incumbent. Algorithm 3 gives the complete inference procedure.

4 Experiments

We evaluate LDR on MIS and MaxCut. Table 1 reports external comparisons, Table 2 evaluates matched-budget destroy-policy variants, and Table 3 shows refinement trajectories. Appendix D explains the hard-exclusion rigidity that motivates destruction for MIS, while Appendix I reports hyperparameters, full statistics, selected-budget runtimes, pairwise counts, and feasibility diagnostics.

Algorithm 2 Alternating Destroy Training and Repair Fine-Tuning

Require: Training graphs $\mathcal{G}$; pretrained repair R_θ; destroy policy D_ϕ; outer rounds M; horizon T; destroy episodes N_D; repair fine-tuning episodes N_R

Initialize θ from Algorithm 1 and initialize ϕ
for $m = 1$ to M **do**
  Freeze θ
  for $i = 1$ to N_D **do**
    Sample $G \in \mathcal{G}$ and an initial vector $\boldsymbol{u} \in [0,1]^n$
    Construct $\boldsymbol{z}^0 = R_\theta^R(G, \boldsymbol{u})$
    for $t = 0$ to $T-1$ **do**
      Compute $\boldsymbol{p}^t = D_\phi(G, \boldsymbol{z}^t, \tau_t)$ and rate ρ^t by Eq. equation 13
      Sample the rate-matched mask $\boldsymbol{a}^t$, perturb $\boldsymbol{z}^t$ to $\tilde{\boldsymbol{z}}^t$, and repair $\boldsymbol{z}^{t+1} = R_\theta^R(G, \tilde{\boldsymbol{z}}^t)$
      Compute reward $r^t = s_G(\text{Decode}(\boldsymbol{z}^{t+1})) - s_G(\text{Decode}(\boldsymbol{z}^t))$
    end for
    Update ϕ using Eq. equation 17 on the collected trajectory
  end for
  Freeze ϕ
  for $j = 1$ to N_R **do**
    Sample $G \in \mathcal{G}$ and an initial vector $\boldsymbol{u} \in [0,1]^n$
    Construct $\boldsymbol{z}^0 = R_\theta^R(G, \boldsymbol{u})$
    Initialize $\mathcal{L}_{\text{ft}} \leftarrow \mathcal{L}_{\text{repair}}(\theta; G, \boldsymbol{z}^0)$
    for $t = 0$ to $T-1$ **do**
      Compute $\boldsymbol{p}^t = D_\phi(G, \boldsymbol{z}^t, \tau_t)$ and rate ρ^t by Eq. equation 13
      Sample the rate-matched mask $\boldsymbol{a}^t$, perturb $\boldsymbol{z}^t$ to $\tilde{\boldsymbol{z}}^t$, and repair $\boldsymbol{z}^{t+1} = R_\theta^R(G, \tilde{\boldsymbol{z}}^t)$
      Accumulate $\mathcal{L}_{\text{ft}} \leftarrow \mathcal{L}_{\text{ft}} + \mathcal{L}_{\text{repair}}(\theta; G, \boldsymbol{z}^{t+1})$
    end for
    Fine-tune θ by minimizing $\mathcal{L}_{\text{ft}}$
  end for
end for
return repair parameters θ and destroy parameters ϕ

4.1 Setup

Datasets. For MIS, we use RB graphs with 200–300 and 800–1200 vertices, following prior neural MIS work (Xu et al., 2005; Xu & Li, 2000). For MaxCut, we use Barabási–Albert graphs with 200–300 and 800–1200 vertices (Barabási & Albert, 1999). Each dataset contains 100 validation instances, and results are averaged over three evaluation seeds.

Baselines. The external comparison uses representative reported values for the same benchmark families. The method-type abbreviations in Table 1 are defined as follows:

- **OR** denotes optimization-based solvers. We include **Gurobi**, a general-purpose mathematical programming solver, and **KaMIS**, a specialized independent-set solver used as a strong MIS reference (Gurobi Optimization, LLC, 2024; Lamm et al., 2017).
- **SL** denotes supervised learning baselines. We include **DGL**, a tree-search and GNN-based MIS solver, and **DIFUSCO**, a graph diffusion solver trained with supervised signal when reported for MIS (Böther et al., 2022; Sun & Yang, 2023).
- **UL** denotes unsupervised learning baselines. We include **LTFT**, a GFlowNet-style graph optimizer; **DiffUCO**, an unsupervised diffusion-based optimizer; **SDDS**, a scalable discrete diffusion sampler; and **ERDOES**, the Erdős Goes Neural unsupervised framework used in the MaxCut block of Table 1 (Zhang et al., 2023; Sanokowski et al., 2024; 2025; Karalias & Loukas, 2020).

Algorithm 3 Iterative Inference

Require: Test graph G; trained repair R_θ; trained destroy policy D_ϕ; repair steps R; destroy-repair iterations T

Sample or initialize $\boldsymbol{u} \in [0,1]^n$
$\boldsymbol{z}^0 = R_\theta^R(G, \boldsymbol{u})$
$\hat{\boldsymbol{x}}^0 = \text{Decode}(\boldsymbol{z}^0)$
Set incumbent $\hat{\boldsymbol{x}}_{\text{best}} \leftarrow \hat{\boldsymbol{x}}^0$
for $t = 0$ to $T-1$ **do**
  Compute per-node destruction probabilities $\boldsymbol{p}^t = D_\phi(G, \boldsymbol{z}^t, \tau_t)$
  Define the eligible vertex set $\mathcal{V}^t_{\text{elig}}$ from $\boldsymbol{z}^t$
  Compute the rate $\rho^t = |\mathcal{V}^t_{\text{elig}}|^{-1} \sum_{i \in \mathcal{V}^t_{\text{elig}}} p^t_i$
  Sample $a^t_i \sim \text{Bernoulli}(\rho^t)$ for $i \in \mathcal{V}^t_{\text{elig}}$ and set $a^t_i = 0$ otherwise
  Perturb masked coordinates in the continuous state to obtain $\tilde{\boldsymbol{z}}^t$
  Repair: $\boldsymbol{z}^{t+1} = R_\theta^R(G, \tilde{\boldsymbol{z}}^t)$
  Decode for evaluation: $\hat{\boldsymbol{x}}^{t+1} = \text{Decode}(\boldsymbol{z}^{t+1})$
  if $s_G(\hat{\boldsymbol{x}}^{t+1}) > s_G(\hat{\boldsymbol{x}}_{\text{best}})$ **then**
    $\hat{\boldsymbol{x}}_{\text{best}} \leftarrow \hat{\boldsymbol{x}}^{t+1}$
  end if
end for
return $\hat{\boldsymbol{x}}_{\text{best}}$

- **RL** denotes reinforcement-learning baselines. We include **PPO-style** solvers and **RLNN**, which use learned policies or regularized Langevin-style dynamics for graph optimization (Ahn et al., 2020; Feng & Yang, 2025).
- **H** denotes classical heuristics. For MaxCut, we report **Greedy** decoding and **MFA** (mean-field annealing) as lightweight heuristic references (Bilbro et al., 1989).

External runtimes are not hardware-normalized; we use them for positioning and rely on fixed-budget internal tables for destroy-variant claims.

Table 2 evaluates the destroy-policy variants with the same number of repair calls:

- **Repair** repeats the fine-tuned repair network without destroying variables.
- **Fixed** $p = .6$ samples each eligible vertex with probability 0.6.
- **Uniform** $[.1, .6]$ first samples a graph-level probability $\rho_t \sim \text{Uniform}(0.1, 0.6)$ and then samples a Bernoulli mask.
- **Graph-rate** uses a learned graph-level probability shared by all eligible vertices.
- **Nodewise** samples directly from the per-node Bernoulli probabilities produced by the destroy network.
- **LDR (Rate-matched)** is our default inference rule: it averages the per-node probabilities over the eligible vertex set and samples a random mask with that learned rate. Comparing it with **Nodewise** tests whether direct nodewise sampling adds value beyond the learned destruction amount.

Metrics and budget. The primary metric is the decoded task score: independent-set size for MIS and cut value for MaxCut; higher is better. All internal methods use 256 refinement iterations, two repair steps per iteration, and 512 total repair calls. Destroy and repair operate on continuous states: MIS resets destroyed active coordinates to zero, while MaxCut flips destroyed coordinates. We log starting score, best score, best step, improvement, runtime, destroy probability, number of destroyed variables, movement diagnostics, and feasibility indicators after decoding. The binary coefficient β in Eq. equation 10 is set to zero, so these runs isolate the multilinear task objective and the fixed decoder.

Table 1: Comparison with representative external baselines on MIS and MaxCut. Scores are independent-set sizes for MIS and cut values for MaxCut; higher is better. Runtime is wall-clock seconds per instance under each method's reported inference setting. External runtimes are not hardware-normalized with the current LDR runs.

Task	Method	Type	Small		Large	
			Score ↑	Time ↓	Score ↑	Time ↓
MIS on RB graphs: Small = RB-[200–300], Large = RB-[800–1200]						
MIS	Gurobi	OR	20.21	0.41	40.90	15.624
	KaMIS	OR	20.23	10.080	**43.15**	14.760
	DGL	SL	17.36	1.534	34.50	2.868
	DIFUSCO	SL	18.52	1.926	–	–
	LTFT	UL	19.18	0.064	37.48	0.524
	DiffUCO	UL	19.59	0.056	38.87	5.924
	SDDS	UL	19.62	0.040	39.32	5.762
	PPO-style	RL	18.43	0.154	32.32	0.906
	RLNN	RL	19.52	0.197	38.06	4.325
	LDR (ours)	UL/RL	**20.28**	2.43	40.58	8.84
MaxCut on BA graphs: Small = BA-[200–300], Large = BA-[800–1200]						
MaxCut	Gurobi	OR	730.87	1.020	2944.38	9.216
	ERDOES	UL	693.45	0.092	2870.34	0.338
	LTFT	UL	704.30	0.354	2864.61	2.560
	DiffUCO	UL	727.32	0.120	2947.53	0.454
	SDDS	UL	730.76	0.028	–	–
	RLNN	RL	725.89	0.190	2907.18	0.440
	Greedy	H	688.31	0.026	2786.00	0.374
	MFA	H	704.03	0.192	2833.86	0.872
	LDR (ours)	UL/RL	**736.42**	0.46	**2963.03**	1.38

4.2 Main Results

Table 1 gives the external reference point. The main observations are:

- **LDR is competitive with reported MIS solvers, but does not dominate specialized search.** On RB-[200–300], LDR reaches 20.28, improving over the strongest listed learning baseline SDDS by 0.66 vertices and slightly exceeding the reported Gurobi/KaMIS values. On RB-[800–1200], LDR reaches 40.58, above the listed neural baselines but below Gurobi and substantially below KaMIS. This is the expected scope of the current solver: it improves neural local refinement, but it is not a replacement for a highly engineered MIS heuristic.
- **LDR is strong on MaxCut under the current comparison.** On BA-[200–300], LDR improves over Gurobi by 5.55 cut edges and over DiffUCO by 9.10. On BA-[800–1200], the corresponding improvements are 18.65 and 15.50. Since every binary partition is feasible for MaxCut, this result shows that the destroy-and-repair loop is not limited to hard-exclusion decoding.
- **External runtimes should be read as positioning, not as causal evidence.** The baselines use their own implementations and hardware assumptions. We therefore use the following internal tables, which share logs, checkpoints, budgets, and devices, to compare the destroy-and-repair variants.

4.3 Destroy-Policy Ablations

Table 2 supports three conclusions.

- **Destroy-before-repair is the dominant source of improvement.** LDR improves over repair-only inference by 2.16 vertices on RB-[200–300] and 5.91 vertices on RB-[800–1200]. On MaxCut,

Table 2: Destroy-policy variants with the same number of repair calls. Values are mean best decoded scores over 100 instances and three seeds. Repair repeats the fine-tuned repair network without destruction. Fixed and uniform random destroy do not use a learned destroy checkpoint. Graph-rate uses a learned graph-level destroy probability. Nodewise samples directly from learned per-node Bernoulli probabilities. LDR (Rate-matched) averages those probabilities over the eligible vertex set and randomizes the actual mask at the learned rate.

Task	Dataset	Repair	Fixed $p = .6$	Uniform $[.1, .6]$	Graph-rate	Nodewise	LDR
MIS	RB-[200–300]	18.12	20.15	20.05	19.86	20.27	**20.28**
MIS	RB-[800–1200]	34.67	40.08	40.02	40.18	40.39	**40.58**
MaxCut	BA-[200–300]	734.36	735.40	735.34	735.18	735.37	**736.42**
MaxCut	BA-[800–1200]	2951.71	2942.80	2957.48	2944.21	2962.98	**2963.03**

the gains are 2.06 and 11.32 cut edges on BA-[200–300] and BA-[800–1200]. These comparisons use the same fine-tuned repair network and the same number of repair calls, so they isolate the value of perturbing the current continuous state before repair.

- **The learned destruction rate is the useful part of the current destroy policy.** LDR is better than direct nodewise sampling in all four rows, with gains of 0.01, 0.19, 1.05, and 0.05 on RB-[200–300], RB-[800–1200], BA-[200–300], and BA-[800–1200], respectively. This supports the interpretation built into the method: under graph-level rewards, the destroy network is most reliable as a state-dependent rate estimator rather than as a source of node-specific causal credit.

- **Simple random destruction remains a strong baseline, but it is not the best variant under this budget.** Uniform $[.1, .6]$ random destruction improves substantially over repair-only inference on every dataset, yet LDR is higher in all four rows of Table 2. Fixed $p = .6$ is also competitive on the smaller graphs, but it is below LDR on every dataset and degrades on BA-[800–1200]. The current evidence therefore supports learned rate matching as the strongest reported inference rule under the same repair-call budget, while the broader lesson is that repair benefits from a nontrivial destruction budget.

The ablation pattern is consistent across the two tasks despite their different feasibility structure. For MIS, destruction creates room for vertices outside the current active set to enter after repair. For MaxCut, destruction instead flips a subset of continuous coordinates so that repair can coordinate a new partition. In both cases, the learned rate is useful even when the concrete mask is randomized, which is consistent with the graph-level reward used to train destroy.

4.4 Search Dynamics

Table 3 shows that repair-only inference saturates early, while destroy-and-repair continues to improve because each destroy step creates a new damaged continuous state for repair. On MIS RB-[800–1200], repair-only reaches 34.67 by step 8 and then remains flat through step 256. On MaxCut BA-[800–1200], repair-only reaches 2951.71 by step 32 and then also stops improving. Uniform $[.1, .6]$ random destruction is a strong trajectory baseline, but LDR improves over it at step 256 by 0.23, 0.56, 1.08, and 5.55 on the four datasets. This supports the rate-matched rule under the fixed repair-call budget.

5 Related Work

LDR connects three lines of work: neural combinatorial optimization, continuous QUBO relaxations, and local-search methods for escaping local optima. Learning-based solvers include supervised, reinforcement-learning, unsupervised-energy, GFlowNet-style, and diffusion-based approaches (Khalil et al., 2017; Bengio et al., 2021; Zhang et al., 2023; Sun & Yang, 2023; Sanokowski et al., 2024); continuous relaxations and physics-inspired GNNs optimize differentiable graph objectives (Karalias & Loukas, 2020; Schuetz et al.,

Table 3: Search dynamics over 256 refinement steps. Each entry is the mean best decoded score over 100 instances and three seeds. Time is the cumulative wall-clock runtime at step 256. Appendix Table 6 gives the corresponding selected-budget scores and runtimes for LDR.

Task	Dataset	Method	0	8	16	32	64	128	256	Time (s)
MIS	RB-[200–300]	Repair	18.03	18.11	18.11	18.12	18.12	18.12	18.12	2.04
		Uniform [.1, .6]	18.03	19.36	19.58	19.76	19.89	19.95	20.05	2.30
		Graph-rate	18.03	19.28	19.47	19.60	19.70	19.82	19.86	2.37
		LDR	18.03	19.68	19.87	20.02	20.12	20.24	20.28	2.43
MIS	RB-[800–1200]	Repair	34.15	34.67	34.67	34.67	34.67	34.67	34.67	7.44
		Uniform [.1, .6]	34.15	37.63	38.27	38.83	39.36	39.74	40.02	8.18
		Graph-rate	34.15	37.91	38.67	39.36	39.82	40.11	40.18	8.88
		LDR	34.15	37.93	38.71	39.38	39.92	40.30	40.58	8.84
MaxCut	BA-[200–300]	Repair	729.59	734.33	734.36	734.36	734.36	734.36	734.36	0.33
		Uniform [.1, .6]	729.59	734.73	735.07	735.16	735.20	735.23	735.34	0.37
		Graph-rate	729.59	734.72	734.94	735.02	735.10	735.16	735.18	0.47
		LDR	729.59	734.87	735.13	735.28	735.36	735.37	736.42	0.46
MaxCut	BA-[800–1200]	Repair	2914.60	2949.09	2951.25	2951.71	2951.71	2951.71	2951.71	0.97
		Uniform [.1, .6]	2914.60	2945.01	2948.59	2951.47	2953.94	2955.80	2957.48	1.00
		Graph-rate	2914.60	2925.04	2927.68	2929.77	2931.92	2934.16	2944.21	1.29
		LDR	2914.60	2951.94	2956.45	2959.52	2961.61	2962.79	2963.03	1.38

2022; Wang et al., 2022); and classical search uses restarts, annealing, tabu moves, large-neighborhood search, and repair heuristics (Shaw, 1998; Blum & Roli, 2003; Ropke & Pisinger, 2006; Nikolaev & Jacobson, 2010). LDR differs by learning a destruction rate for a continuous repair trajectory rather than only constructing a solution once or solving an explicit discrete neighborhood subproblem. Appendix B gives the expanded discussion, and Appendix E details the relation to large-neighborhood search.

6 Discussion

Limitations. LDR targets graph QUBO settings where a current binary or near-binary solution is useful for refinement. The hard-exclusion analysis applies most directly to MIS and MaxClique, while the experiments cover MIS and MaxCut. The theory does not prove global convergence, and the destroy-repair loop adds inference cost. Because the destroy reward is graph-level, the learned policy should be interpreted as a state-dependent destruction-rate estimator rather than node-specific causal credit.

Conclusion. We presented LDR, a learned destroy-and-repair framework for multilinear graph QUBO optimization. Experiments on MIS and MaxCut show that the loop improves over repair-only inference under the same repair-call budget and that rate-matched inference is stronger than fixed, uniform, graph-rate, and nodewise variants. Appendix A gives the broader-impact discussion.

References

Sungsoo Ahn, Younggyo Seo, and Jinwoo Shin. Learning what to defer for maximum independent sets. In *International Conference on Machine Learning*, pp. 134–144. PMLR, 2020.

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

Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Griff L. Bilbro, Reinhold Mann, Thomas K. Miller, Wesley E. Snyder, David E. Van den Bout, and Mark White. Optimization by mean field annealing. *Advances in Neural Information Processing Systems*, 1, 1989.

Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

Maximilian Böther, Otto Kißig, Martin Taraz, Sarel Cohen, Karen Seidel, and Tobias Friedrich. What's wrong with deep learning in tree search for combinatorial optimization. *arXiv preprint arXiv:2201.10494*, 2022.

Shengyu Feng and Yiming Yang. Regularized langevin dynamics for combinatorial optimization. *arXiv preprint arXiv:2502.00277*, 2025.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

Will Grathwohl, Kevin Swersky, Milad Hashemi, David Duvenaud, and Chris Maddison. Oops i took a gradient: Scalable sampling for discrete distributions. In *International Conference on Machine Learning*, pp. 3831–3841. PMLR, 2021.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL `https://www.gurobi.com`.

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: An unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33:6659–6672, 2020.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, 2017.

Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2018.

Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *Journal of Heuristics*, 23(4):207–229, 2017. doi: 10.1007/s10732-017-9337-x.

Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014.

Peter Merz and Bernd Freisleben. Greedy and local search heuristics for unconstrained binary quadratic programming. *Journal of Heuristics*, 8(2):197–213, 2002.

George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.

Alexander G. Nikolaev and Sheldon H. Jacobson. Simulated annealing. In Michel Gendreau and Jean-Yves Potvin (eds.), *Handbook of Metaheuristics*, pp. 1–39. Springer, 2010.

David Pisinger and Stefan Ropke. Large neighborhood search. In Michel Gendreau and Jean-Yves Potvin (eds.), *Handbook of Metaheuristics*, pp. 399–419. Springer, 2010.

Abraham P. Punnen. The quadratic unconstrained binary optimization problem. *Springer International Publishing*, 2022.

Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. Dimes: A differentiable meta solver for combinatorial optimization problems. *Advances in Neural Information Processing Systems*, 35:25531–25546, 2022.

Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

Hong Seo Ryoo and Nikolaos V. Sahinidis. Analysis of bounds for multilinear functions. *Journal of Global Optimization*, 19(4):403–424, 2001.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. *arXiv preprint arXiv:2406.01661*, 2024.

Sebastian Sanokowski, Wilhelm Berghammer, Martin Ennemoser, Haoyu Peter Wang, Sepp Hochreiter, and Sebastian Lehner. Scalable discrete diffusion samplers: Combinatorial optimization and statistical physics. *arXiv preprint arXiv:2502.08696*, 2025.

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, 2022.

Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference on Principles and Practice of Constraint Programming*, pp. 417–431. Springer, 1998.

Haoran Sun, Hanjun Dai, Wei Xia, and Arun Ramamurthy. Path auxiliary proposal for mcmc in discrete space. In *International Conference on Learning Representations*, 2021.

Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. *Advances in Neural Information Processing Systems*, 36:3706–3731, 2023.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Haoyu Peter Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. *Advances in Neural Information Processing Systems*, 35:31444–31458, 2022.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

Ke Xu and Wei Li. Exact phase transitions in random constraint satisfaction problems. *Journal of Artificial Intelligence Research*, 12(1):93–103, 2000.

Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. A simple model to generate hard satisfiable instances. *arXiv preprint cs/0509032*, 2005.

Eitan Zemel. Measuring the quality of approximate solutions to zero-one programming problems. *Mathematics of Operations Research*, 6(3):319–332, 1981.

Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron C. Courville, Yoshua Bengio, and Ling Pan. Let the flows tell: Solving graph combinatorial problems with gflownets. *Advances in Neural Information Processing Systems*, 36:11952–11969, 2023.

Ruqi Zhang, Xingchao Liu, and Qiang Liu. A langevin-like sampler for discrete distributions. In *International Conference on Machine Learning*, pp. 26375–26396. PMLR, 2022.

A Broader Impact Statement

This work studies generic graph optimization algorithms. Potential benefits include faster heuristic solvers for scheduling, resource allocation, and scientific computing. Potential risks arise if such solvers are used inside high-stakes allocation systems without validating objective design, constraints, and fairness implications. The method itself does not use personal data, but downstream deployments should audit whether the optimized graph objective faithfully represents the intended decision problem.

B Additional Related Work

Neural combinatorial optimization. Learning-based combinatorial optimization includes supervised construction policies, reinforcement learning, unsupervised energy minimization, GFlowNet-style methods, and diffusion-based solvers (Khalil et al., 2017; Bengio et al., 2021; Zhang et al., 2023; Sun & Yang, 2023; Sanokowski et al., 2024). LDR belongs to the unsupervised family because it does not require optimal labels, but it differs from one-shot construction by repeatedly perturbing the current continuous state and repairing it.

Continuous relaxations and QUBO. QUBO and multilinear relaxations connect binary graph objectives to continuous optimization (Punnen, 2022; Lucas, 2014; Zemel, 1981; Merz & Freisleben, 2002; Ryoo & Sahinidis, 2001). Physics-inspired GNN solvers and principled objective relaxations exploit this differentiability for graph problems (Karalias & Loukas, 2020; Schuetz et al., 2022; Wang et al., 2022). Our analysis emphasizes both sides of this relaxation: it preserves 0–1 optima and strict local minima are binary, but the relaxed landscape can still contain stable suboptimal binary solutions.

Escaping local optima. Classical methods use random restarts, simulated annealing, tabu search, large-neighborhood search, and repair-based perturbation heuristics (Shaw, 1998; Blum & Roli, 2003; Ropke & Pisinger, 2006; Nikolaev & Jacobson, 2010; Lamm et al., 2017; Pisinger & Ropke, 2010); recent sampling methods use gradient-informed or Langevin-like proposals (Grathwohl et al., 2021; Sun et al., 2021; Zhang et al., 2022; Feng & Yang, 2025). LDR is closest in spirit to large-neighborhood search, but its large-neighborhood move is implemented through a learned continuous repair operator rather than an explicit discrete subproblem solver.

C Proof of Theorem 1

For any $\boldsymbol{y} \in [0,1]^n$, let $\boldsymbol{Z} \sim \prod_{i=1}^n \mathrm{Bern}(y_i)$ be a random binary vector with independent coordinates. By construction of the multilinear extension,

$$F(\boldsymbol{y}) = \mathbb{E}\left[H(\boldsymbol{Z})\right]. \tag{21}$$

Because $\{0,1\}^n \subset [0,1]^n$ and $F(\boldsymbol{x}) = H(\boldsymbol{x})$ for every $\boldsymbol{x} \in \{0,1\}^n$,

$$\min_{\boldsymbol{y}\in[0,1]^n} F(\boldsymbol{y}) \le \min_{\boldsymbol{x}\in\{0,1\}^n} H(\boldsymbol{x}). \tag{22}$$

Conversely, for any $\boldsymbol{y} \in [0,1]^n$,

$$F(\boldsymbol{y}) = \mathbb{E}\left[H(\boldsymbol{Z})\right] \ge \min_{\boldsymbol{x}\in\{0,1\}^n} H(\boldsymbol{x}), \tag{23}$$

since an expectation is bounded below by the minimum value of the random variable's support. Taking the minimum over $\boldsymbol{y} \in [0,1]^n$ gives

$$\min_{\boldsymbol{y}\in[0,1]^n} F(\boldsymbol{y}) = \min_{\boldsymbol{x}\in\{0,1\}^n} H(\boldsymbol{x}). \tag{24}$$

Let $m = \min_{\boldsymbol{x}\in\{0,1\}^n} H(\boldsymbol{x})$ and let $\bar{\boldsymbol{y}}$ be a continuous global minimizer. Then $F(\bar{\boldsymbol{y}}) = m$. Since $H(\boldsymbol{Z}) \ge m$ almost surely and $\mathbb{E}[H(\boldsymbol{Z})] = m$, the random variable $H(\boldsymbol{Z})$ equals m on every binary assignment with positive probability under $\bar{\boldsymbol{y}}$. The distribution has nonempty support, so at least one supported 0–1 vector $\boldsymbol{x}^\star$ satisfies $H(\boldsymbol{x}^\star) = m$. Therefore $F(\boldsymbol{x}^\star) = H(\boldsymbol{x}^\star) = m$, and a 0–1 global minimizer of the continuous problem exists. Moreover, for any binary vector $\boldsymbol{x}$, $F(\boldsymbol{x}) = H(\boldsymbol{x})$. Thus $\boldsymbol{x}$ is a discrete global minimizer of H if and only if $F(\boldsymbol{x}) = m$, which is exactly the condition for the same 0–1 vector to be a global minimizer of the relaxed problem.

It remains to show that strict local minimizers are 0–1. Suppose $\boldsymbol{y}^\star$ is a strict local minimizer of F but is not binary. Then there exists a coordinate i with $0 < y_i^\star < 1$. Fixing all other coordinates, multilinearity implies that

$$\phi_i(t) = F(y_1^\star, \ldots, y_{i-1}^\star, t, y_{i+1}^\star, \ldots, y_n^\star) \tag{25}$$

is affine in t. If this affine function has positive slope, moving t slightly left decreases F; if it has negative slope, moving t slightly right decreases F; if it has zero slope, F is locally constant along that coordinate. Each case contradicts strict local minimality. Hence every strict local minimizer is a 0–1 vector.

D Structural Analysis

For MIS, local improvement has an additional structural obstruction beyond nonconvexity: once a solution is maximal, improving it requires first removing selected vertices. The same statement holds for MaxClique after replacing the graph by the conflict graph of missing edges. This argument does not apply to MaxCut, where every partition is feasible and improvement instead requires coordinated variable flips.

Proposition 1 (Hard-exclusion rigidity)**.** *Let S be a maximal independent set in G. If S' is an independent set with $|S'| > |S|$, then transforming S into S' requires removing at least one vertex from S. If $|S'| = |S| + n$ with $n \geq 1$, and the transition removes p vertices from S, then it must add $q = n + p$ vertices outside S.*

The proof is immediate: maximality means no outside vertex can be added while keeping all vertices in S. Thus any larger feasible set must first remove a selected vertex. The bookkeeping identity $q = n + p$ follows from the set-size difference.

This proposition is one motivation for learned destruction in hard-exclusion tasks. A repair network that simply follows its current selection preference may reconstruct the same maximal solution after decoding. A useful destroy policy must perturb enough of the continuous state representing the current selection pattern to make alternative additions possible, without destroying so much that repair degenerates into construction from scratch. For MaxCut, the same module is interpreted as learning how many variables to perturb so that repair can coordinate a new partition.

E Relation to Large Neighborhood Search

LDR can be viewed as a neural large-neighborhood-search-style method. Classical large neighborhood search (LNS) improves an incumbent solution by destroying part of it and then repairing the resulting partial solution through a problem-specific search or insertion procedure (Shaw, 1998; Pisinger & Ropke, 2010). Adaptive LNS further changes which destroy and repair operators are used during search (Ropke & Pisinger, 2006). Our method follows the same high-level template, but the state being perturbed and repaired is continuous. Decoding is used to score the post-repair state and to select the final incumbent solution.

The main difference is how the large-neighborhood improvement is carried out. In standard LNS, repair is often a handcrafted combinatorial routine, a constraint-programming subproblem, or an insertion heuristic. In LDR, repair is a learned recurrent GNN trained by optimizing the original multilinear continuous QUBO relaxation. The repair step therefore searches the neighborhood through a continuous objective whose strict local minima are binary by Theorem 1, followed by a fixed task decoder. This is the sense in which the large-neighborhood repair move is implemented through continuous optimization rather than an explicit discrete subproblem solver.

The learned destroy module also has a narrower role than a full adaptive LNS operator selector. Because its reward is graph-level, the current training signal does not provide reliable node-specific causal credit. We therefore interpret the destroy network primarily as learning the destruction amount: how large the neighborhood should be for the current graph, state, and search step. The default rate-matched inference rule in Eq. equation 13 averages the per-node probabilities over the eligible vertex set into a learned destruction rate and then samples the concrete mask randomly at that rate. This explains why the random-destroy baselines are strong: once the repair operator is effective, a nontrivial destruction budget already creates useful large neighborhoods. The benefit of LDR is to adapt that budget from data while leaving the exact destroyed vertices randomized under graph-level rewards.

F Task-Specific Objective Details

For MaxCut, there is no hard-exclusion decoder. The relaxed energy is

$$F^{\text{MaxCut}}(\boldsymbol{y}) = -\sum_{(i,j)\in E} w_{ij}\left[y_i(1-y_j) + y_j(1-y_i)\right]. \tag{26}$$

At inference time, the continuous output is thresholded to a binary partition and evaluated by Eq. equation 3. Because every binary partition is feasible, the destroy step is not "removing" selected vertices; it flips masked variables so that repair can coordinate a new partition.

G Implementation Details

The current prototype uses residual-aware GIN layers with layer normalization and sigmoid node heads for both repair and destroy. Repair is optimized with AdamW and gradient clipping. Destroy is trained with Bernoulli mask actions, sampled time-step features, a finite-horizon repair environment, normalized advantages, and gradient clipping. The reported comparisons should be read together with the runtime columns because the destroy-repair loop adds repeated network calls at inference time.

During destroy training, repair is frozen and each graph can contribute multiple sampled rollouts to the same policy update. Each rollout first constructs $\boldsymbol{z}^0$ from a random initial vector with the frozen repair network, then repeatedly applies the destroy mask and frozen repair update to produce $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^T$. We use these repeated rollouts to form a leave-one-sample baseline: the baseline for one sample is the mean return of the other samples from the same graph when available, with the batch mean used as a fallback. The resulting advantages are normalized before the REINFORCE update. This is an implementation variance-reduction device; it does not change the graph-level reward definition in Eq. equation 15.

During repair fine-tuning, destroy is frozen and the same rollout structure is used. Repair again constructs the initial state $\boldsymbol{z}^0$ from noise, then receives each damaged continuous state produced by the frozen destroy rule and maps it to $\boldsymbol{z}^{t+1}$. The fine-tuning loss is the sum of multilinear repair losses over all repaired states in the rollout, including the initial construction output $\boldsymbol{z}^0$ and every subsequent post-destroy repair output. The decoded task score is not used in this repair loss; decoding remains only for destroy rewards and evaluation.

H Model and Evaluation Details

Node features. Both repair and destroy receive node-wise feature tensors built from the current soft state $\boldsymbol{z} \in [0,1]^{n\times K}$, where K is the number of repair samples per graph. For a node i, let d_i be its degree, $N(i)$ its neighbors, $b(i)$ its graph index in a mini-batch, and

$$n_i(\boldsymbol{z}) = \sum_{j\in N(i)} z_j, \qquad T_i(\boldsymbol{z}) = \sum_{j:b(j)=b(i)} z_j, \qquad o_i(\boldsymbol{z}) = T_i(\boldsymbol{z}) - z_i. \tag{27}$$

The universal feature builder uses a shared feature basis for MIS and MaxCut. It includes the current value z_i, degree d_i, neighbor mass n_i, graph total T_i, other-node mass o_i, non-neighbor mass $o_i - n_i$, non-neighbor support/conflict ratios, and cut-style local statistics. In particular, the expected cross-side and same-side cut masses are

$$c_i^{\text{cross}} = z_i(d_i - n_i) + (1-z_i)n_i, \qquad c_i^{\text{same}} = z_i n_i + (1-z_i)(d_i - n_i). \tag{28}$$

The feature vector also contains $c_i^{\text{cross}}/(d_i+\epsilon)$, $c_i^{\text{same}}/(d_i+\epsilon)$, $c_i^{\text{cross}} - c_i^{\text{same}}$, and a graph-level balance feature $(T_i - |V_{b(i)}|/2)/(|V_{b(i)}|+\epsilon)$. When a mask-aware repair checkpoint is used, the damage mask is appended as an additional feature channel. The small constant is $\epsilon = 10^{-6}$ in feature construction.

Residual-aware GIN backbone. Each node feature vector is first projected by a two-layer MLP with ReLU and layer normalization to obtain $h_i^{(0)} \in \mathbb{R}^{d_h}$. The backbone contains L residual-aware GIN layers.

Let $\alpha = 0.5$ and let $u_i^{(\ell)}$ be the input to layer ℓ:

$$u_i^{(\ell)} = (1-\alpha)h_i^{(\ell-1)} + \alpha h_i^{(0)}. \tag{29}$$

If a repair mask m_i is supplied, the layer first gates the input as $\tilde{u}_i^{(\ell)} = m_i u_i^{(\ell)}$; otherwise $\tilde{u}_i^{(\ell)} = u_i^{(\ell)}$. When global features are enabled, the layer computes a graph mean $\bar{u}_{b(i)}^{(\ell)} = |V_{b(i)}|^{-1} \sum_{j:b(j)=b(i)} \tilde{u}_j^{(\ell)}$ and replaces $\tilde{u}_i^{(\ell)}$ by

$$q_i^{(\ell)} = \mathrm{MLP}_{\mathrm{global}}^{(\ell)} \left([\tilde{u}_i^{(\ell)}, \bar{u}_{b(i)}^{(\ell)}] \right). \tag{30}$$

Without global features, $q_i^{(\ell)} = \tilde{u}_i^{(\ell)}$. The GIN aggregation and residual MLP update are

$$a_i^{(\ell)} = \sum_{j \in N(i)} q_j^{(\ell)} + (1+\epsilon_\ell) q_i^{(\ell)}, \qquad h_i^{(\ell)} = \mathrm{ReLU}\left[\mathrm{LayerNorm}\left(\mathrm{MLP}_\ell(a_i^{(\ell)}) + q_i^{(\ell)} \right) \right], \tag{31}$$

where ϵ_ℓ is a learned scalar initialized at zero. The final node representation uses jump-knowledge concatenation followed by a linear projection:

$$h_i^{\mathrm{out}} = W_{\mathrm{JK}} \left[h_i^{(1)}; \ldots; h_i^{(L)} \right]. \tag{32}$$

The repair network uses $d_h = 128$ and $L = 5$; the destroy network uses $d_h = 64$ and $L = 4$ in the reported experiments. Since the destroy reward is shared across all node-mask actions of a graph, the node features should be read mainly as a permutation-equivariant way to summarize the current state and set a graph-conditioned destroy rate.

Repair and destroy heads. The repair head is a two-layer MLP followed by a sigmoid:

$$z_i^{\mathrm{new}} = \sigma\left(w_r^\top \mathrm{ReLU}(W_r h_i^{\mathrm{out}} + b_r) + c_r \right). \tag{33}$$

For recurrent repair, this update is applied repeatedly, with the output of one recurrent step becoming the input state of the next step. The destroy network uses the same backbone structure but additionally conditions on an inference time variable $\tau \in [0,1]$. In the checkpoints used here, τ is encoded by a sinusoidal embedding, passed through a two-layer MLP, and added to the projected node features before message passing. The destroy probabilities define a graph-level expected destroy budget $B_\phi = \sum_i p_i^{\mathrm{dest}}$ and are computed as

$$p_i^{\mathrm{dest}} = \sigma\left(w_d^\top \mathrm{ReLU}(W_d h_i^{\mathrm{out}} + b_d) + c_d \right). \tag{34}$$

The default LDR inference rule averages these probabilities over the eligible vertex set and samples the actual mask with the resulting rate. For MIS, destroyed active coordinates are reset to zero before repair. For MaxCut, destroyed coordinates are flipped, and a mask-aware repair checkpoint receives the flipped-variable mask as an input channel when the checkpoint architecture supports it.

Internal evaluation variants. The repair-only variant repeatedly applies the fine-tuned repair network with the same repair-call budget as LDR but never destroys variables. Fixed random destroy uses $\rho_t = 0.6$. Uniform random destroy samples $\rho_t \sim \mathrm{Uniform}(0.1, 0.6)$ independently for each graph and refinement step. Graph-rate destroy uses a learned graph-level probability shared by all eligible vertices. Nodewise destroy samples directly from per-node Bernoulli probabilities. LDR uses rate-matched inference: it first averages the per-node probabilities over the eligible vertex set and then samples a random mask with that average probability. The pretraining ablation compares a noise-only repair checkpoint with a multi-source repair checkpoint. The fine-tuning ablation compares learned-destroy inference with the pretrained repair checkpoint and with the repair checkpoint fine-tuned over learned-destroy trajectories, where the multilinear repair loss is applied to the initial construction output and all post-destroy repair outputs.

Table 4: Evaluation hyperparameters used for the reported internal experiments.

Hyperparameter	Value
GPU	NVIDIA A100-SXM4-40GB
Datasets	RB-[200–300], RB-[800–1200], BA-[200–300], BA-[800–1200]
Split	validation
Instances per dataset	100
Evaluation seeds	0, 1, 2
Batch size	32
Repair samples per graph	8
Destroy-repair iterations	256
Repair steps per iteration	2
Total repair calls	512
Conflict coefficient λ	1.5
Binary coefficient β	0.0
Repair hidden dimension / layers	128 / 5
Destroy hidden dimension / layers	64 / 4
Global graph features in destroy	yes
Fixed random destroy probability	0.6
Uniform random destroy range	$[0.1, 0.6]$
LDR rate matching	matched per graph and step

I Additional Experimental Details

Random destroy baseline. The fixed and uniform random baselines use explicit random masks, not untrained destroy networks. The uniform baseline samples one graph-level rate $\rho_t \sim \text{Uniform}(0.1, 0.6)$ per graph and refinement step, and then samples the concrete mask from that rate. For MIS, random masks are sampled only over active coordinates of the current continuous state; for MaxCut, random masks are sampled over all vertices and the masked variables are flipped. LDR differs from fixed and uniform random destruction because it first computes the learned destroy policy's average per-node probability at the current graph and step, then samples a random mask with that probability. This rate-matched rule reuses the learned destruction amount while avoiding reliance on nodewise probability variation.

Feasibility diagnostics. Table 8 summarizes the feasibility diagnostics. For MIS, decoding enforces feasibility and maximality. In the reported evaluation, the post-decoding conflict count is zero for every method, instance, and seed, and the feasibility and maximality indicators are both one for all MIS rows. For MaxCut, every binary partition is feasible, so the feasibility field is always one and the maximality field is unused.

Table 5: Full internal results corresponding to Table 2. Scores are mean ± standard deviation over 100 instances and three seeds. $\bar{p}$ is the mean or expected graph-level destroy probability, and $\bar{d}$ is the mean number of destroyed variables per step when logged.

Task	Dataset	Method	Best score ↑	Improvement ↑	Best step	Time (s) ↓	$\bar{p}$	$\bar{d}$
MIS	RB-[200–300]	Repair	18.12 ± 1.43	0.09	0.1	2.04	–	–
		Fixed $p = .6$	20.15 ± 1.45	2.13	29.4	2.25	0.600	11.1
		Uniform $[.1, .6]$	20.05 ± 1.43	2.02	25.4	2.30	0.350	–
		Graph-rate	19.86 ± 1.46	1.83	26.7	2.37	0.243	4.4
		Nodewise	20.27 ± 1.48	2.24	16.5	2.46	0.250	4.9
		LDR	20.28 ± 1.47	2.25	26.0	2.43	0.266	5.1
		No FT	20.12 ± 1.45	1.53	17.6	2.37	–	–
		With FT	20.26 ± 1.48	2.24	16.5	2.40	–	–
		Noise only	18.26 ± 1.65	0.37	0.7	1.80	–	–
		Multi-source	19.08 ± 1.42	0.49	0.6	2.15	–	–
MIS	RB-[800–1200]	Repair	34.67 ± 2.82	0.52	0.5	7.44	–	–
		Fixed $p = .6$	40.08 ± 3.24	5.93	78.8	8.03	0.600	21.7
		Uniform $[.1, .6]$	40.02 ± 3.32	5.87	72.4	8.18	0.350	–
		Graph-rate	40.18 ± 3.36	6.03	54.3	8.88	0.123	4.8
		Nodewise	40.39 ± 3.30	6.24	87.8	8.82	0.198	7.7
		LDR	40.58 ± 3.24	6.43	92.4	8.84	0.201	7.9
		No FT	40.39 ± 3.31	6.15	97.7	8.80	–	–
		With FT	40.43 ± 3.31	6.28	95.8	8.87	–	–
		Noise only	34.44 ± 3.15	0.28	0.3	7.42	–	–
		Multi-source	34.25 ± 3.01	0.00	0.0	7.40	–	–
MaxCut	BA-[200–300]	Repair	734.36 ± 83.51	4.77	2.4	0.33	–	–
		Fixed $p = .6$	735.40 ± 83.61	5.81	35.0	0.36	0.600	150.7
		Uniform $[.1, .6]$	735.34 ± 83.59	5.75	24.4	0.37	0.350	–
		Graph-rate	735.18 ± 83.64	5.60	13.2	0.47	0.070	17.6
		Nodewise	735.37 ± 83.64	5.78	9.3	0.47	0.101	25.3
		LDR	736.42 ± 83.70	6.83	9.7	0.46	0.101	25.4
		No FT	735.37 ± 83.64	5.78	9.3	0.43	–	–
		With FT	735.37 ± 83.64	5.78	9.3	0.43	–	–
		Noise only	733.51 ± 83.55	5.38	2.4	0.33	–	–
		Multi-source	734.36 ± 83.51	4.77	2.4	0.33	–	–
MaxCut	BA-[800–1200]	Repair	2951.71 ± 350.57	37.11	9.4	0.97	–	–
		Fixed $p = .6$	2942.80 ± 346.62	28.20	125.1	1.00	0.600	597.9
		Uniform $[.1, .6]$	2957.48 ± 349.21	42.88	174.3	1.00	0.350	–
		Graph-rate	2944.21 ± 347.51	29.61	213.1	1.29	0.449	447.9
		Nodewise	2962.98 ± 350.80	48.38	69.2	1.39	0.711	708.6
		LDR	2963.03 ± 350.68	48.43	70.7	1.38	0.711	708.7
		No FT	2963.05 ± 350.84	48.45	71.3	1.34	–	–
		With FT	2963.00 ± 350.78	48.40	72.8	1.35	–	–
		Noise only	2946.50 ± 349.35	37.42	9.3	0.96	–	–
		Multi-source	2951.69 ± 350.52	37.09	9.4	0.96	–	–

Table 6: LDR best-score and runtime at selected inference budgets. Each score row is followed by its cumulative runtime row in seconds.

Task	Dataset	0	8	16	32	64	128	256
MIS	RB-[200–300]	18.03	19.68	19.87	20.02	20.12	20.24	20.28
	runtime	0.01	0.08	0.15	0.30	0.60	1.20	2.43
MIS	RB-[800–1200]	34.15	37.93	38.71	39.38	39.92	40.30	40.58
	runtime	0.03	0.29	0.55	1.09	2.19	4.40	8.84
MaxCut	BA-[200–300]	729.59	734.87	735.13	735.28	735.36	735.37	736.42
	runtime	0.00	0.02	0.03	0.06	0.12	0.23	0.46
MaxCut	BA-[800–1200]	2914.60	2951.94	2956.45	2959.52	2961.61	2962.79	2963.03
	runtime	0.00	0.05	0.09	0.18	0.35	0.69	1.38

Table 7: Pairwise comparisons over matched instance-seed pairs. The last column reports wins/ties/losses for the first method in each comparison when pairwise counts are available.

Task	Dataset	Comparison	Mean diff.	W/T/L
MIS	RB-[200–300]	LDR − Repair	2.16	283/17/0
		LDR − Fixed	0.12	43/249/8
		LDR − Uniform	0.23	–
		LDR − Graph-rate	0.42	91/200/9
		LDR − Nodewise	0.01	23/257/20
		With FT − No FT	0.14	57/229/14
		Multi-source − Noise	0.82	181/88/31
MIS	RB-[800–1200]	LDR − Repair	5.91	300/0/0
		LDR − Fixed	0.50	159/100/41
		LDR − Uniform	0.56	–
		LDR − Graph-rate	0.41	107/164/29
		LDR − Nodewise	0.19	83/178/39
		With FT − No FT	0.04	68/175/57
		Multi-source − Noise	-0.19	85/94/121
MaxCut	BA-[200–300]	LDR − Repair	2.06	–
		LDR − Fixed	1.02	–
		LDR − Uniform	1.08	–
		LDR − Graph-rate	1.24	–
		LDR − Nodewise	1.05	–
		With FT − No FT	0.00	0/300/0
		Multi-source − Noise	0.85	155/103/42
MaxCut	BA-[800–1200]	LDR − Repair	11.32	291/6/3
		LDR − Fixed	20.23	300/0/0
		LDR − Uniform	5.55	–
		LDR − Graph-rate	18.82	299/0/1
		LDR − Nodewise	0.05	80/126/94
		With FT − No FT	-0.05	47/202/51
		Multi-source − Noise	5.19	235/16/49

Table 8: Feasibility diagnostics for the main internal comparison. Feasible and maximal columns are percentages. Pre- and post-conflict columns are mean decoded conflict counts. Maximality is not applicable to MaxCut because every binary partition is feasible.

Task	Dataset	Method	Feasible (%)	Maximal (%)	Pre-conflict	Post-conflict
MIS	RB-[200–300]	Repair	100.0	100.0	0.007	0.000
		Fixed $p = .6$	100.0	100.0	0.000	0.000
		Uniform $[.1, .6]$	100.0	100.0	0.003	0.000
		Graph-rate	100.0	100.0	0.003	0.000
		Nodewise	100.0	100.0	0.000	0.000
		LDR	100.0	100.0	0.007	0.000
MIS	RB-[800–1200]	Repair	100.0	100.0	0.197	0.000
		Fixed $p = .6$	100.0	100.0	0.287	0.000
		Uniform $[.1, .6]$	100.0	100.0	0.210	0.000
		Graph-rate	100.0	100.0	0.110	0.000
		Nodewise	100.0	100.0	0.043	0.000
		LDR	100.0	100.0	0.097	0.000
MaxCut	BA-[200–300]	Repair	100.0	–	0.000	0.000
		Fixed $p = .6$	100.0	–	0.000	0.000
		Uniform $[.1, .6]$	100.0	–	0.000	0.000
		Graph-rate	100.0	–	0.000	0.000
		Nodewise	100.0	–	0.000	0.000
		LDR	100.0	–	0.000	0.000
MaxCut	BA-[800–1200]	Repair	100.0	–	0.000	0.000
		Fixed $p = .6$	100.0	–	0.000	0.000
		Uniform $[.1, .6]$	100.0	–	0.000	0.000
		Graph-rate	100.0	–	0.000	0.000
		Nodewise	100.0	–	0.000	0.000
		LDR	100.0	–	0.000	0.000