SCoR: Unsupervised State-Conditioned Refinement for Integer Linear Programming

Anonymous authors
Paper under double-blind review

Abstract

Integer linear programming (ILP) is a general framework for combinatorial optimization. Unsupervised approaches learn to solve ILPs without solver-generated supervision, but their updates often rely on aggregate signals that obscure the prediction-dependent states of individual constraints and their effects across variables. We propose State-Conditioned Refinement (SCoR), which incorporates localized constraint feedback into unsupervised ILP optimization. Given a relaxed prediction, SCoR encodes normalized constraint residuals and propagates them to incident variables through sparse attention over the bipartite incidence graph. This feedback and the current optimization state parameterize a bounded positive diagonal preconditioner that adapts the base update to heterogeneous local conditions. We characterize the refinement's boundedness and descent compatibility and show that, under a matched update budget, variable-wise preconditioning admits a broader allocation set than uniform scaling and can yield a strictly larger local-model decrease. SCoR preserves the underlying unsupervised objective and can be integrated with different optimization schemes. Experiments on standard benchmarks and real-world instances demonstrate improvements across representative unsupervised ILP optimizers.

1 Introduction

Integer linear programming (ILP) is a fundamental modeling framework for combinatorial optimization, with broad applications in production planning (Ajagekar et al., 2022), scheduling (Klein et al., 2024), transportation (Osorio & Ouyang, 2025), and molecular machine learning (Kretschmer et al., 2025). Despite decades of progress in mathematical optimization, solving large-scale ILPs remains computationally challenging due to the discrete and combinatorial nature of their solution spaces. Modern solvers combine branch-and-bound (Land & Doig, 2009), branch-and-cut (Gomory, 2009), primal heuristics, and other sophisticated techniques. Their computational cost can nonetheless be substantial when many structurally related instances must be solved. Learning-based optimization offers a complementary approach by exploiting regularities across related instances (Nair et al., 2020). Neural models can encode the variable-constraint structure of an ILP and use the resulting representations to guide optimization or parameterize candidate solution distributions (Gasse et al., 2019). This allows learned structural patterns to transfer to unseen instances of the same problem class. However, many existing approaches rely on high-quality solutions computed by traditional solvers for supervision, tying model training to expensive solver runs (Huang et al., 2024).

Unsupervised neural optimization provides an appealing alternative by learning directly from the optimization problem rather than from precomputed target solutions (Wang et al., 2022). The objective and constraints themselves provide the training signal, allowing the model to improve its solution distribution without solver-generated supervision (Deng et al., 2026; Huang et al., 2026). Recent ILP methods such as DiffILO (Geng et al., 2025) use probabilistic formulations and surrogate objectives to optimize over discrete solution spaces. These advances establish a promising solver-free training paradigm for jointly pursuing feasibility and solution quality.

Despite this progress, constraint information is still used at a relatively coarse level during optimization. The base objective combines the effects of multiple constraints into a variable-wise optimization signal, but this aggregation does not explicitly preserve the individual states of the contributing constraints or their associations with the variables they involve. This distinction matters because the

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.

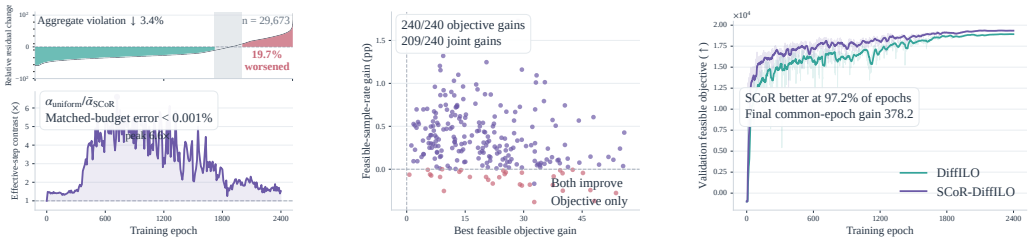


Figure 1: Motivating diagnostics for SCoR on Combinatorial Auction (CA). Aggregate violation decreases although some constraint residuals worsen (left). The lower-left panel shows how SCoR redistributes effective update magnitudes under a matched budget. Gains in feasible objective do not always increase feasible-sample rate (middle). SCoR generally achieves a higher validation objective than DiffILO during training (right).

influence of an ILP constraint changes with the predicted solution. A previously satisfied constraint may become violated, while two variables receiving similar objective-level guidance may nevertheless be embedded in markedly different local constraint environments. Our diagnostic analysis in Figure 1 illustrates this phenomenon, showing that aggregate improvement can coexist with heterogeneous local responses. This suggests that a useful base signal does not by itself specify how the resulting update should be shaped under the current constraint configuration.

To address this limitation, we propose SCoR (*State-Conditioned Refinement*) for ILP, a novel framework for refining unsupervised ILP predictions. Starting from an initial prediction, SCoR evaluates the residual state of each constraint and augments its structural representation accordingly. A State-Conditioned Constraint Feedback Attention module then propagates these augmented representations to incident variables over the sparse bipartite incidence graph, yielding localized feedback that preserves distinctions among local constraint configurations not explicitly represented in the aggregate optimization signal. Conditioned on this feedback and the current optimization state, SCoR learns a positive diagonal preconditioner that reshapes the variable-wise geometry of the update induced by the base objective. More broadly, this formulation distinguishes global optimization guidance from local update allocation, offering a general principle for integrating prediction-dependent constraint information into the optimization dynamics of unsupervised ILP solvers. Our contributions are summarized as follows.

- We identify an underexplored limitation of aggregate optimization signals in unsupervised ILP, namely that they do not explicitly retain the evolving states of individual constraints and their local associations with decision variables.
- We propose SCoR, which transforms state-dependent constraint information into localized feedback and uses a bounded positive preconditioner to shape the base update according to heterogeneous local conditions.
- We theoretically characterize the refinement's descent compatibility and allocation flexibility, and demonstrate gains in prediction quality and solver-assisted search across state-of-the-art unsupervised ILP methods, establishing state-conditioned update allocation as an effective refinement mechanism.

2 Related Work

2.1 Neural Solvers for MILP

Machine learning has become increasingly prevalent in accelerating ILP solving (Bengio et al., 2021; Scavuzzo et al., 2024). Within exact solvers, learned models have been applied to branching decisions (Gasse et al., 2019; Scavuzzo et al., 2022), node selection (He et al., 2014; Labassi et al., 2022), and cut management (Paulus et al., 2022; Ling et al., 2024; Puigdemont et al., 2024). Beyond solver components, neural models have been trained to generate complete or partial heuristic solutions (Ding et al., 2020; Nair et al., 2020). The resulting predictions can warm-start the solver, determine variables to fix, or define restricted neighborhoods for subsequent search (Han et al., 2023; Huang et al., 2024). More recent methods improve the reliability and structural consistency

of solution prediction through iterative correction and constraint-level variable modeling (Liu et al., 2025; Pu et al., 2026). Although these approaches can substantially reduce the subsequent search space, their training generally depends on high-quality solutions generated by traditional solvers.

2.2 Unsupervised Learning for Combinatorial Optimization

Unsupervised neural optimization learns from the problem objective and constraints without solution labels. Probabilistic and energy-based formulations have been developed for graph combinatorial optimization (Karalias & Loukas, 2020; Schuetz et al., 2022), followed by work on scalable distribution learning and generative sampling (Qiu et al., 2022; Sanokowski et al., 2024; Li et al., 2026). Other approaches derive learning objectives and decoding procedures for common constraint families (Bu et al., 2024), but remain tailored to specific combinatorial structures. For general ILPs, DiffILO parameterizes a probabilistic solution distribution and jointly optimizes objective value and constraint violations without solver-generated supervision (Geng et al., 2025). PUMA extends this formulation with stochastic neighborhood perturbations to preserve solution diversity and mitigate premature convergence (Deng et al., 2026). Label-free energy-based sampling has also been extended to mixed-integer problems (Huang et al., 2026). In unsupervised ILP methods, however, constraint effects are typically combined in aggregate objectives, leaving the role of each constraint's current state in shaping individual variable updates implicit.

3 Preliminaries

Binary Integer Linear Programming. A binary integer linear program (ILP) takes the form

$$\min_{\mathbf{x}\in\{0,1\}^n} \mathbf{c}^\top \mathbf{x} \qquad \text{s.t.} \qquad A\mathbf{x} \le \mathbf{b}, \tag{1}$$

where $\mathbf{x} \in \{0,1\}^n$ denotes the decision variables, $\mathbf{c} \in \mathbb{R}^n$ contains the objective coefficients, $A \in \mathbb{R}^{m\times n}$ is the constraint matrix, and $\mathbf{b} \in \mathbb{R}^m$ contains the corresponding right-hand sides. The j-th constraint is written as $\mathbf{a}_j^\top \mathbf{x} \le b_j$. Following prior learning-based ILP methods (Geng et al., 2025; Deng et al., 2026), the discussion focuses on binary decision variables.

For a solution $\mathbf{x}$, the violation of constraint j is defined as $v_j(\mathbf{x}) = \left[\mathbf{a}_j^\top \mathbf{x} - b_j\right]_+$, where $[\cdot]_+$ denotes the positive-part operator.

Bipartite Graph Representation. Following the standard graph representation for ILPs (Gasse et al., 2019), an ILP instance is represented as a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$, where $\mathcal{V} = \{1, \dots, n\}$ and $\mathcal{C} = \{1, \dots, m\}$ denote the variable and constraint node sets, respectively. An edge $(j, i) \in \mathcal{E}$ exists if and only if $A_{ji} \neq 0$, with its features characterizing the interaction between variable i and constraint j.

A bipartite GNN alternates message passing between variable and constraint nodes. A generic layer can be expressed as

$$h_j^c \leftarrow f_c\left(h_j^c, \sum_{i:(j,i)\in\mathcal{E}} g_c(h_i^v, h_j^c, e_{ji})\right), \qquad h_i^v \leftarrow f_v\left(h_i^v, \sum_{j:(j,i)\in\mathcal{E}} g_v(h_j^c, h_i^v, e_{ji})\right), \tag{2}$$

where h_i^v and h_j^c denote the hidden representations of variable and constraint nodes, respectively. After several message-passing layers, the variable representations are mapped to logits $\mathbf{z} \in \mathbb{R}^n$, which parameterize the solution distribution.

Unsupervised ILP Optimization. Unsupervised ILP methods train solution predictors directly from the problem objective and constraints, without solver-generated solution labels (Geng et al., 2025; Deng et al., 2026). Given predictor logits $z \in \mathbb{R}^n$, a common probabilistic formulation models the binary variables independently through Bernoulli marginals,

$$p_i = \sigma(z_i), \qquad x_i \sim \text{Bernoulli}(p_i), \tag{3}$$

where $p = \sigma(z)$ parameterizes a distribution over binary assignments. The ILP objective and constraint violations evaluated under this distribution provide the training signal for the predictor.

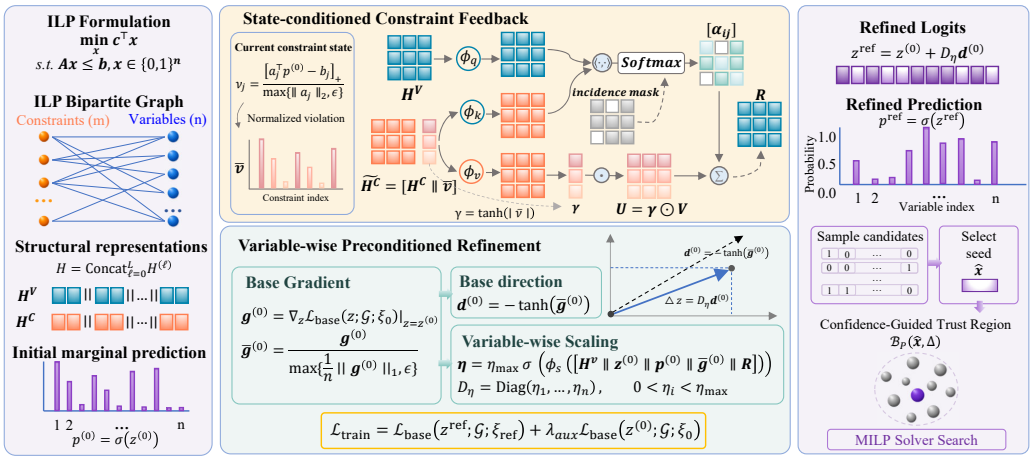


Figure 2: Overview of SCoR. State-conditioned constraint attention provides localized feedback to refine the initial prediction through variable-wise scaling of the base update. The refined marginals guide candidate sampling and subsequent MILP solver search.

Methods differ in how they construct this signal and use sampling or perturbation to explore the solution space. We denote the unsupervised objective of a base method by

$$\mathcal{L}_{\text{base}}(z; \mathcal{G}, \xi) := \mathcal{L}_{\text{unsup}}(z; \mathcal{G}, \xi), \tag{4}$$

where ξ denotes stochastic variables associated with its sampling or exploration procedure. This notation lets us describe SCoR across base methods while retaining their respective objectives and exploration strategies.

4 State-Conditioned Refinement

4.1 Overview

Given an ILP graph $\mathcal{G}$, the bipartite backbone retains variable and constraint states at multiple propagation depths. SCoR concatenates the states of each node along the feature dimension to form the structural representations H^v and H^c. A prediction head maps H^v to the initial logits $z^{(0)}$, with marginals $p^{(0)} = \sigma(z^{(0)})$. SCoR then evaluates each constraint under $p^{(0)}$ and augments its structural representation with the resulting residual state. Sparse constraint-to-variable attention propagates these state-conditioned representations to incident variables, producing localized feedback that reflects their current constraint context. The overall process is illustrated in Figure 2.

SCoR combines this feedback with the current prediction state and the signal supplied by the base unsupervised objective to construct a positive diagonal preconditioner over variables. The preconditioner assigns each variable a state-dependent scale for the base update, allowing different local constraint configurations to induce different effective refinement magnitudes.

4.2 State-Dependent Constraint Feedback

Let $h_i^{v,(\ell)}$ and $h_j^{c,(\ell)}$ denote the states retained for variable i and constraint j at depth ℓ of the bipartite backbone, respectively, where $\ell = 0$ corresponds to the embedded input features. The structural representation of each node is formed by concatenating its states across backbone depths:

$$h_i^v = \left[h_i^{v,(0)} \parallel \cdots \parallel h_i^{v,(L)}\right], \qquad h_j^c = \left[h_j^{c,(0)} \parallel \cdots \parallel h_j^{c,(L)}\right]. \tag{5}$$

The resulting collections are denoted by $H^v = \{h_i^v\}_{i=1}^n$ and $H^c = \{h_j^c\}_{j=1}^m$. Details of the backbone architectures are provided in Appendix A.1.

These multi-layer representations encode structural relationships accumulated at different backbone depths, but they do not explicitly describe how individual constraints behave under the current prediction. Since the relevance of a constraint changes as the prediction evolves, SCoR evaluates each

constraint at the relaxed prediction $p^{(0)} = \sigma(z^{(0)})$ and augments its structural representation with the resulting residual information. Specifically, SCoR characterizes constraint j by its normalized positive marginal residual

$$\nu_j = \frac{\left[a_j^\top p^{(0)} - b_j\right]_+}{\max\{\|a_j\|_2, \epsilon\}}, \qquad \bar{\nu}_j = \frac{\nu_j}{\max\left\{\frac{1}{m}\sum_{\ell=1}^{m}\nu_\ell, \epsilon\right\}}, \tag{6}$$

where $\epsilon > 0$ is a small constant for numerical stability. The row normalization reduces sensitivity to the numerical scale of individual constraints, while the instance-wise normalization reduces variation in residual magnitude across problem instances. Since ν_j depends on $p^{(0)}$, the same structural constraint can carry different state information as the prediction evolves.

To propagate the evolving constraint state to variables, SCoR employs state-conditioned constraint feedback attention over the original variable-constraint incidence structure. Let ϕ_q, ϕ_k, and ϕ_v denote learnable transformations that produce the query, key, and value representations. We define

$$\widetilde{h}_j^c = [h_j^c \| \bar{\nu}_j], \qquad q_i = \phi_q(h_i^v), \qquad k_j = \phi_k(\widetilde{h}_j^c), \qquad u_j = \gamma_j \phi_v(\widetilde{h}_j^c), \qquad \gamma_j = \tanh(|\bar{\nu}_j|). \tag{7}$$

The activity factor γ_j acts as a continuous activity gate. A constraint with zero residual therefore contributes no value message, while constraints with larger normalized residuals receive stronger activity gates.

For variable i, let $\mathcal{N}(i) = \{j \mid (j,i) \in \mathcal{E}\}$ denote its incident constraints. The attention weights and aggregated feedback are

$$\alpha_{ji} = \frac{\exp\left(q_i^\top k_j / \sqrt{d_h}\right)}{\sum_{\ell \in \mathcal{N}(i)} \exp\left(q_i^\top k_\ell / \sqrt{d_h}\right)}, \qquad r_i = \sum_{j \in \mathcal{N}(i)} \alpha_{ji} u_j, \tag{8}$$

where d_h denotes the attention dimension. The weights α_{ji} allow incident constraints to contribute differently according to their structural representations and current states. Since attention is restricted to $\mathcal{N}(i)$ and the value messages are state-gated, r_i aggregates localized feedback from constraints incident to variable i.

4.3 Variable-Wise Preconditioned Refinement

With a local constraint context available for each variable, the remaining question is how it should shape the update suggested by the base objective. SCoR then uses r_i to parameterize the variable-wise scaling of this update. The base objective $\mathcal{L}_{\text{base}}(z; \mathcal{G}, \xi)$ induces a logit-level optimization signal at the base prediction $z^{(0)}$. We define this signal and its normalized form as

$$g^{(0)} = \nabla_z \mathcal{L}_{\text{base}}(z; \mathcal{G}, \xi_0)|_{z=z^{(0)}}, \qquad \bar{g}^{(0)} = \frac{g^{(0)}}{\max\left\{\frac{1}{n}\|g^{(0)}\|_1, \epsilon\right\}}, \tag{9}$$

where $\|\cdot\|_1$ denotes the ℓ_1-norm. This normalization reduces instance-wise variation in its scale.

For each variable, SCoR combines its structural representation, current prediction state, normalized optimization signal, and localized constraint feedback to estimate a bounded positive scaling

$$\eta_i = \eta_{\max} \sigma\left(\phi_s\left([h_i^v \| z_i^{(0)} \| p_i^{(0)} \| \bar{g}_i^{(0)} \| r_i]\right)\right), \qquad 0 < \eta_i < \eta_{\max}, \tag{10}$$

where ϕ_s is a shared learnable transformation and $\eta_{\max} > 0$ sets an upper bound on the variable-wise scaling.

We define the bounded base direction and the corresponding diagonal scaling matrix by

$$d^{(0)} = -\tanh(\bar{g}^{(0)}), \qquad D_\eta = \operatorname{diag}(\eta_1, \ldots, \eta_n), \qquad z^{\text{ref}} = z^{(0)} + D_\eta d^{(0)}. \tag{11}$$

The diagonal scaling allows SCoR to reshape the relative update magnitudes across variables according to the current constraint state.

Proposition 4.1 (Descent-compatible and bounded refinement)**.** *Let $\delta z = z^{\text{ref}} - z^{(0)}$ denote the refinement in Eq. 11. Then*

$$\left\langle g^{(0)}, \delta z \right\rangle = -\sum_{i=1}^{n} \eta_i |g_i^{(0)}| \tanh(|\bar{g}_i^{(0)}|) \le 0, \qquad \|\delta z\|_\infty < \eta_{\max}. \tag{12}$$

If $g^{(0)} \neq 0$, then $\left\langle g^{(0)}, \delta z \right\rangle < 0$.

When $\mathcal{L}_{\text{base}}$ is differentiable at $z^{(0)}$, the strict inequality implies that any nonzero SCoR refinement is a descent direction for $\mathcal{L}_{\text{base}}$ at the current prediction, while the infinity-norm bound controls the magnitude of each update. This guarantee is defined with respect to the base unsupervised objective and does not assert descent for the original discrete ILP objective.

A shared scaling factor can adjust the overall strength of the update, but it cannot change how that update magnitude is distributed among variables. To isolate what is gained by allowing these scales to vary, define

$$\rho_i = |d_i^{(0)}| = \tanh(|\bar{g}_i^{(0)}|), \qquad s_i = |\delta z_i| = \eta_i \rho_i, \tag{13}$$

where ρ_i denotes the magnitude of the bounded base direction and s_i denotes the effective update magnitude assigned to variable i. For a fixed total effective update budget B, define the admissible allocation set

$$\mathcal{S}_B = \left\{ s \in \mathbb{R}_+^n \;\middle|\; \sum_{i=1}^n s_i = B,\ 0 \le s_i \le \eta_{\max} \rho_i \right\}. \tag{14}$$

Proposition 4.2 (Matched-budget variable-wise allocation)**.** *Consider a uniform scaling $\eta_i = \eta$ with $\eta \in [0, \eta_{\max}]$, and let $s_i^{\text{uni}} = \eta \rho_i$ satisfy $\sum_i s_i^{\text{uni}} = B$. Then $s^{\text{uni}} \in \mathcal{S}_B$ and*

$$\max_{s \in \mathcal{S}_B} \sum_{i=1}^n |g_i^{(0)}| s_i \ge \sum_{i=1}^n |g_i^{(0)}| s_i^{\text{uni}}. \tag{15}$$

Moreover, the inequality is strict if there exist coordinates a and b such that

$$|g_a^{(0)}| > |g_b^{(0)}|, \qquad s_a^{\text{uni}} < \eta_{\max} \rho_a, \qquad s_b^{\text{uni}} > 0. \tag{16}$$

For the SCoR allocation $s_i = |\delta z_i|$, the weighted sum in Eq. 15 equals $-\langle g^{(0)}, \delta z \rangle$, the decrease predicted by the local linear model of the base objective. Proposition 4.2 shows that variable-wise scaling admits a broader allocation set than uniform scaling under a matched budget. The condition in Eq. 16 identifies when reallocating magnitude toward a stronger base signal yields a strictly larger local-model decrease. This establishes the potential value of flexible allocation without prescribing how to achieve it. SCoR does not optimize over $\mathcal{S}_B$. Its learned preconditioner determines variable-wise scales from the current optimization state and localized constraint context. Method-specific instantiations are detailed in Appendices A.2 and A.3, with proofs in Appendix B.

4.4 Learning and Inference

SCoR is trained jointly with the underlying predictor without additional solution labels. Starting from $z^{(0)}$, the feedback and preconditioning steps produce z^{ref}, which is evaluated by the same unsupervised objective used by the base method. We additionally retain an auxiliary objective on the initial prediction,

$$\mathcal{L}_{\text{train}} = \mathcal{L}_{\text{base}}\big(z^{\text{ref}}; \mathcal{G}, \xi_{\text{ref}}\big) + \lambda_{\text{aux}} \mathcal{L}_{\text{base}}\Big(z^{(0)}; \mathcal{G}, \xi_0\Big), \tag{17}$$

where λ_{aux} controls the auxiliary term and ξ_0, ξ_{ref} denote the stochastic quantities used at the two evaluations. Optimization details are provided in Appendix C.

At inference time, SCoR outputs the refined marginals $p^{\text{ref}} = \sigma(z^{\text{ref}})$, which can be sampled directly or used to guide a downstream solver. For the solver-assisted evaluation, we use a sample-conditioned trust-region search. We first sample binary candidates from p^{ref} and select a seed solution $\hat{x}$ according to feasibility and solution quality. The seed provides a concrete binary assignment, while the marginals indicate confidence in its individual components. Specifically, we rank variables with $\hat{x}_i = 0$ by increasing p_i^{ref} and those with $\hat{x}_i = 1$ by decreasing p_i^{ref}. Selecting up to k_0 and k_1 variables from the two groups yields an index set $\mathcal{P}$. The solver then searches within

$$\mathcal{B}_{\mathcal{P}}(\hat{x}, \Delta) = \left\{ x \;\middle|\; \sum_{i \in \mathcal{P}} |x_i - \hat{x}_i| \le \Delta \right\}, \tag{18}$$

subject to the original ILP constraints.

This construction uses the sampled assignment and confidence estimates together to define the search neighborhood. It is one inference strategy for SCoR, and other variable-selection rules or adaptive neighborhoods can be used without changing the refinement mechanism. Detailed inference settings are provided in Appendix A.4.

5 EXPERIMENT

Benchmarks. We evaluate our method on three widely studied benchmarks: Set Covering (SC), Maximum Independent Set (IS), and Combinatorial Auction (CA) (Wang et al., 2024). We generate the instances following (Gasse et al., 2019), with 240 training, 60 validation, and 100 test instances for each problem (Ye et al., 2024). We further evaluate our method on real-world MILP instances from MIPLIB 2017 (Gleixner et al., 2021) to examine its applicability beyond synthetically generated benchmarks. The three benchmarks cover distinct combinatorial structures and objective characteristics, providing a common testbed for evaluating both solution prediction and solver-guided search (Wang et al., 2023). Detailed generation and dataset settings are provided in Appendix C.1.

Baselines. Our baselines span three representative categories. We include the general-purpose ILP solvers Gurobi (Gurobi Optimization, LLC, 2026) and SCIP (Achterberg, 2009). For supervised solution prediction, PaS (Han et al., 2023) uses learned marginal predictions to restrict solver search, while CoCo-MILP (Pu et al., 2026) models inter-variable contrast and intra-constraint competition. For unsupervised learning, we compare against DiffILO (Geng et al., 2025) and PUMA (Deng et al., 2026), which optimize ILP solution distributions without solver-generated supervision. To evaluate SCoR across these unsupervised optimization schemes, we instantiate it with each method while retaining its respective objective. We denote the resulting variants by $\text{SCoR}_{\text{DiffILO}}$ and $\text{SCoR}_{\text{PUMA}}$. Baseline configurations and implementation details are summarized in Appendix C.2.

Metrics. Following prior work (Han et al., 2023), we evaluate solving performance using the objective value (Obj), absolute primal gap (Gap_{abs}), and relative primal gap (Gap_{rel}). For each test instance, we initialize the best-known solution (BKS) with the result of a 3,600-second Gurobi run and update it if any evaluated method finds a better feasible solution. This gives all methods a common reference for that instance. Given a solution x and the best-known solution x^*, we compute $\text{Gap}_{\text{abs}} = |f(x) - f(x^*)|$ and $\text{Gap}_{\text{rel}} = |f(x) - f(x^*)| / \max\left\{|f(x^*)|, 10^{-10}\right\}$. We report averages of the per-instance metrics. Obj and Gap_{abs} assess final solution quality, while Gap_{rel} tracks the solving process over time.

5.1 MAIN RESULTS

Learned ILP predictions can help a solver find stronger feasible solutions within a limited search budget. We examine whether SCoR improves this process when applied to DiffILO and PUMA. Figure 4 tracks the average relative primal gap during solver-assisted search. Both refined variants generally reduce the gap relative to their base predictors, most visibly for $\text{SCoR}_{\text{DiffILO}}$ on CA.

Table 1 reports the final objective values and absolute primal gaps. On SC, several prediction-guided methods using (k_0, k_1, Δ) achieve a better average objective than Gurobi_{3600}. Both SCoR variants achieve the best 1,000-second result of 124.56, with a gap of 0.18 to the updated BKS. On CA, SCoR reduces the gap from 432.56 to 398.23 with DiffILO and from 478.62 to 451.80 with PUMA. $\text{SCoR}_{\text{DiffILO}}$ achieves the best result among methods evaluated at 1,000 seconds. Relative to Gurobi_{1000}, the best SCoR variant reduces the absolute primal gap by 80.0% on SC and 22.6% on CA. On IS, several methods attain a gap of 0.00 at the reported precision, leaving little room to distinguish their final solution quality. The improvement over Gurobi_{3600} on SC does not hold under the original Δ-only inference. Those results and the choices of (k_0, k_1, Δ) are provided in Appendix C.2.

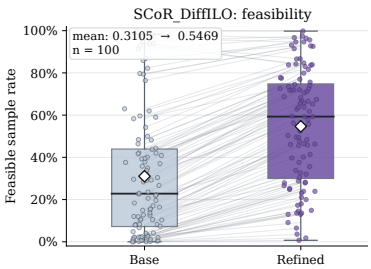
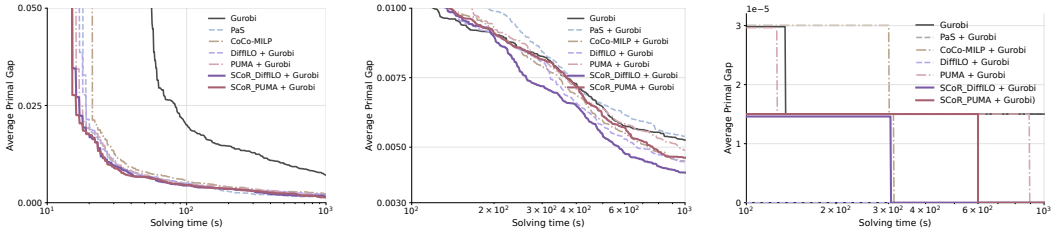


Figure 3: Feasible sample rates.

Prediction Quality Before Solver Search. We also examine whether SCoR improves samples before solver-assisted search. On SC, Figure 3 shows that DiffILO's mean feasible sample rate rises from 31.05% to 54.69%, increasing on 94 of 100 instances. The refined distribution thus provides feasible candidates more often for seed selection. Additional results on mean primal integrals and prediction quality, including feasible sampled objectives and per-instance gains for SC, CA, and IS, are presented in Appendix C.3. Results using SCIP are reported in Appendix C.4.

Table 1: Average Obj and Gap_{abs} on 100 test instances. **Bold** marks the best results at 1,000 seconds, while underlining identifies the Gurobi_{3600} reference used to initialize the BKS. Improvement is the percentage reduction in Gap_{abs} of the best SCoR variant relative to Gurobi_{1000}.

Method	SC(BKS:124.38)		CA(BKS:97714.24)		IS(BKS:684.14)	
	Obj↓	Gap_{abs} ↓	Obj↑	Gap_{abs} ↓	Obj↑	Gap_{abs} ↓
Gurobi_{3600}	<u>124.77</u>	<u>0.39</u>	<u>97395.20</u>	<u>319.04</u>	<u>684.14</u>	<u>0.00</u>
Gurobi_{1000}	125.28	0.90	97199.85	514.39	684.13	0.01
PaS	124.57	0.19	97186.62	527.62	684.13	0.01
CoCo-MILP	124.66	0.28	97273.99	440.25	**684.14**	**0.00**
DiffILO	124.65	0.27	97281.68	432.56	**684.14**	**0.00**
PUMA	124.62	0.24	97235.62	478.62	**684.14**	**0.00**
$\text{SCoR}_{\text{DiffILO}}$	**124.56**	**0.18**	**97316.01**	**398.23**	**684.14**	**0.00**
$\text{SCoR}_{\text{PUMA}}$	**124.56**	**0.18**	97262.44	451.80	**684.14**	**0.00**
Improvement	80.0%		22.6%		100.0%	



(a) Set Covering (b) Combinatorial Auction (c) Maximum Independent Set

Figure 4: Average relative primal gap Gap_{rel} to each instance's best-known objective over 1,000 seconds. Curves are averaged over 100 test instances, and lower is better.

5.2 Generalization Analysis

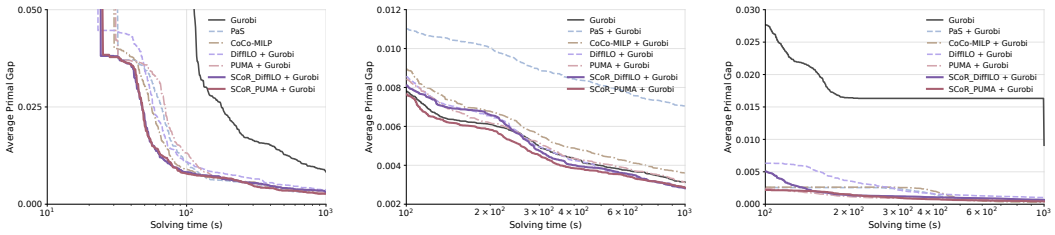
On instances with size settings different from those used in training, both SCoR variants improve the final objectives of their respective base methods across all three tasks. Figure 5 shows how the relative primal gap evolves during search. The refinement enables $\text{SCoR}_{\text{PUMA}}$ and $\text{SCoR}_{\text{DiffILO}}$ to achieve the strongest results among learning-based methods on SC and CA, respectively, at 1,000 seconds. IS remains more challenging: $\text{SCoR}_{\text{PUMA}}$ ranks second behind CoCo-MILP, while outperforming both unsupervised baselines. Together, these results suggest that the learned refinement remains useful across changes in instance size, although its gains vary by task. Complete results and further analysis are provided in Appendix C.5.

5.3 Interpretability Analysis

To examine how the proposed refinement affects optimization dynamics, we consider the toy ILP used in prior studies (Deng et al., 2026),

$$\min_{x_1,x_2\in\{0,1\}} \{-11x_1 - 10x_2 \mid x_1 + x_2 \le 1\}.$$

The global optimum $(1, 0)$ and the suboptimal solution $(0, 1)$ have close objective values, while the shared constraint induces direct competition between the two variables. With Bernoulli marginals $p = (p_1, p_2)$, the expected penalized objective is $F_\mu(p) = -11p_1 - 10p_2 + \mu p_1 p_2$. We set $\mu = 22$, initialize all methods from $p^{(0)} = (0.60, 0.64)$, and optimize for 140 steps. Each SCoR variant performs one refinement step with $\eta_{\max} = 0.9$. Results are averaged over 20 random seeds. Figure 6 shows that SCoR shifts the marginal trajectories toward the global optimum earlier under both DiffILO and PUMA. At step 40, the probability of sampling $(1, 0)$ rises from 0.26 to 0.74 for DiffILO and from 0.33 to 0.81 for PUMA. Together, the trajectories and sampling probabilities illustrate how state-conditioned variable-wise scaling changes the relative progress of the competing variables and concentrates the distribution on the optimum earlier.



(a) Set Covering (b) Combinatorial Auction (c) Maximum Independent Set

Figure 5: Average Gap_{rel} to each instance's BKS over 1,000 seconds on larger-scale instances. Curves are averaged over 100 test instances, and lower is better.

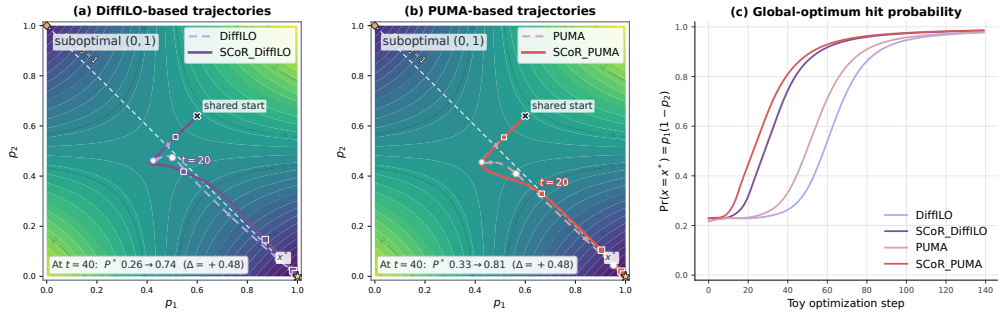


Figure 6: **Optimization dynamics on the toy ILP.** (a-b) Marginal trajectories for DiffILO and PUMA with and without SCoR. (c) Probability of sampling the global optimum, $p_1(1-p_2)$.

5.4 Ablation Study

We conduct ablations with DiffILO as the base predictor. w/o Constraint Feedback removes refinement feedback while retaining constraint information in the GNN backbone. w/o State Conditioning retains the feedback but removes its dependence on current constraint residuals. The full model improves the SC objective from 124.65 to 124.56 and the CA objective from 97281.68 to 97316.01. It outperforms every ablation on SC. On CA, removing feedback still improves upon DiffILO but falls short of the full model, whereas static feedback, uniform scaling, and removal of the auxiliary loss underperform DiffILO. These results point to interactions among the components. The static-feedback comparison supports conditioning on the current constraint state, while the uniform-scale comparison supports adapting update magnitudes across variables.

Table 2: Ablation study of $\text{SCoR}_{\text{DiffILO}}$ under a 1,000-second solving budget. Results are averaged over 100 test instances.

Method	SC	CA
	Obj↓	Obj↑
DiffILO	124.65	97281.68
w/o State Conditioning	124.57	97176.71
w/o Constraint Feedback	124.61	97308.49
w/o Variable-wise Scaling	124.58	97180.92
w/o Auxiliary Loss	124.62	97211.53
$\text{SCoR}_{\text{DiffILO}}$	**124.56**	**97316.01**

6 Conclusion

We proposed SCoR, a state-conditioned refinement mechanism for unsupervised ILP prediction. It propagates current constraint residuals to incident variables and learns a bounded positive scale for each variable's update under the base objective. Our analysis establishes descent compatibility and characterizes the allocation flexibility of variable-wise scaling under a matched budget. Integrated with DiffILO and PUMA, SCoR improves solver-assisted search on standard benchmarks and instances with different sizes. Prediction-level and held-out CVS results further support state-guided refinement without solver-generated labels. More broadly, this highlights the value of adaptive update allocation in unsupervised optimization.

REFERENCES

Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

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

Egon Balas and Andrew Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. In *Combinatorial optimization*, pp. 37–60. Springer, 2009.

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.

David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John Hooker. *Decision diagrams for optimization*, volume 1. Springer, 2016.

Fanchen Bu, Hyeonsoo Jo, Soo Yong Lee, Sungsoo Ahn, and Kijung Shin. Tackling prevalent conditions in unsupervised combinatorial optimization: Cardinality, minimum, covering, and more. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 4696–4729. PMLR, 21–27 Jul 2024.

Yufan Deng, Tianle Pu, Zhijing Hu, Zijie Geng, Li Zeng, Xingchen Hu, Kuihua Huang, Junjie Wu, and Changjun Fan. A perturbation-augmented unsupervised learning framework for integer linear programming. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 851–862, 2026.

Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the aaai conference on artificial intelligence*, volume 34, pp. 1452–1459, 2020.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Zijie Geng, Jie Wang, Xijun Li, Fangzhou Zhu, Jianye Hao, Bin Li, and Feng Wu. Differentiable integer linear programming. In *International Conference on Learning Representations*, volume 2025, pp. 28283–28309, 2025.

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. doi: 10.1007/s12532-020-00194-3. URL `https://doi.org/10.1007/s12532-020-00194-3`.

Ralph E Gomory. Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem. In *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pp. 77–103. Springer, 2009.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2026. URL `https://www.gurobi.com`.

Qingyu Han, Linxin Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. A GNN-guided predict-and-search framework for mixed-integer linear programming. In *The Eleventh International Conference on Learning Representations*, 2023.

He He, Hal Daumé, and Jason Eisner. Learning to search in branch and bound algorithms. *Advances in neural information processing systems*, 27, 2014.

Taoan Huang, Aaron M Ferber, Arman Zharmagambetov, Yuandong Tian, and Bistra Dilkina. Contrastive predict-and-search for mixed integer linear programs. In *Forty-first International Conference on Machine Learning*, 2024.

Yixin Huang, Shengyu Feng, and Yiming Yang. Unsupervised neural langevin sampler for mixed integer linear programming. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=R3za7ajCwo.

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.

Nicklas Klein, Mario Gnägi, and Norbert Trautmann. Mixed-integer linear programming for project scheduling under various resource constraints. *European Journal of Operational Research*, 319 (1):79–88, 2024.

Fleming Kretschmer, Jan Seipp, Marcus Ludwig, Gunnar W Klau, and Sebastian Böcker. Coverage bias in small molecule machine learning. *Nature communications*, 16(1):554, 2025.

Abdel Ghani Labassi, Didier Chételat, and Andrea Lodi. Learning to compare nodes in branch and bound with graph neural networks. *Advances in neural information processing systems*, 35: 32000–32010, 2022.

Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pp. 105–132. Springer, 2009.

Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pp. 66–76, 2000.

Hongpei Li, Hui Yuan, Han Zhang, Jianghao Lin, Dongdong Ge, Mengdi Wang, and Yinyu Ye. Fmip: Joint continuous-integer flow for mixed-integer linear programming. In *International Conference on Learning Representations*, volume 2026, pp. 151594–151614, 2026.

Haotian Ling, Zhihai Wang, and Jie Wang. Learning to stop cut generation for efficient mixed-integer linear programming. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 20759–20767, 2024.

Haoyang Liu, Jie Wang, Zijie Geng, Xijun Li, Yuxuan Zong, Fangzhou Zhu, Jianye Hao, and Feng Wu. Apollo-milp: An alternating prediction-correction neural solving framework for mixed-integer linear programming. In *International Conference on Learning Representations*, volume 2025, pp. 94513–94540, 2025.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid Von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Jesus Osorio and Yanfeng Ouyang. To ride or to fly: Optimal freight-on-transit operations using drones. *Transportation Research Part E: Logistics and Transportation Review*, 201:104231, 2025.

Max B Paulus, Giulia Zarpellon, Andreas Krause, Laurent Charlin, and Chris Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International conference on machine learning*, pp. 17584–17600. PMLR, 2022.

Tianle Pu, Jianing Li, Yingying Gao, Shixuan Liu, Zijie Geng, Haoyang Liu, Chao Chen, and Changjun Fan. Coco-milp: Inter-variable contrastive and intra-constraint competitive milp solution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 24882–24890, 2026.

Pol Puigdemont, Stratis Skoulakis, Grigorios Chrysos, and Volkan Cevher. Learning to remove cuts in integer linear programming. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 41235–41255. PMLR, 21–27 Jul 2024.

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.

Sebastian Sanokowski, Sepp Hochreiter, and Sebastian Lehner. A diffusion model framework for unsupervised neural combinatorial optimization. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43346–43367. PMLR, 21–27 Jul 2024.

Lara Scavuzzo, Feng Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen Aardal. Learning to branch with tree mdps. *Advances in neural information processing systems*, 35:18514–18526, 2022.

Lara Scavuzzo, Karen Aardal, Andrea Lodi, and Neil Yorke-Smith. Machine learning augmented branch and bound for mixed integer linear programming. *Mathematical Programming*, pp. 1–44, 2024.

Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022.

Haoyu Peter Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 31444–31458. Curran Associates, Inc., 2022. doi: 10.52202/068431-2280.

Haoyu Peter Wang, Jialin Liu, Xiaohan Chen, Xinshang Wang, Pan Li, and Wotao Yin. DIG-MILP: a deep instance generator for mixed-integer linear programming with feasibility guarantee. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=MywlrEaFqR.

Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *The Eleventh International Conference on Learning Representations*, 2023.

Huigen Ye, Hua Xu, and Hongyan Wang. Light-MILPopt: Solving large-scale mixed integer linear programs with lightweight optimizer and small-scale training dataset. In *The Twelfth International Conference on Learning Representations*, 2024.

A Implementation Details

This section provides additional implementation details for the neural predictor and describes how SCoR is instantiated with the two unsupervised ILP optimization schemes considered in our experiments. We focus on the model, training procedure and the solver-assisted inference pipeline.

A.1 GNN Architecture

Following standard learning-based ILP approaches (Gasse et al., 2019; Geng et al., 2025), we represent each ILP instance as a weighted bipartite graph consisting of variable nodes, constraint nodes, and edges corresponding to nonzero entries of the constraint matrix. We retain the native input features of each base implementation. The detailed feature definitions are summarized in Table 3.

Variable and constraint features are first projected into a common d-dimensional embedding space by separate MLPs. The backbone then applies L bipartite message-passing blocks. Each block consists of a variable-to-constraint half-convolution followed by a constraint-to-variable half-convolution:

$$h_j^{c,(\ell+1)} = \mathrm{GN}_c^{(\ell)}\left(f_c^{(\ell)}\left[h_j^{c,(\ell)}, \operatorname*{AGG}_{i\in\mathcal{N}(j)} g_c^{(\ell)}\left(h_i^{v,(\ell)}, h_j^{c,(\ell)}, e_{ji}\right)\right]\right),$$
$$h_i^{v,(\ell+1)} = \mathrm{GN}_v^{(\ell)}\left(f_v^{(\ell)}\left[h_i^{v,(\ell)}, \operatorname*{AGG}_{j\in\mathcal{N}(i)} g_v^{(\ell)}\left(h_j^{c,(\ell+1)}, h_i^{v,(\ell)}, e_{ji}\right)\right]\right), \tag{19}$$

where the message and update functions are implemented as two-layer MLPs with ReLU activations, and GN denotes GraphNorm. We use mean aggregation over neighboring nodes, consistent with the underlying DiffILO and PUMA implementations.

Rather than using only the final propagation layer, we retain the embedded input and the output of every message-passing block. Their concatenation forms the multi-depth representations

$$h_i^v = \mathrm{Concat}_{\ell=0}^{L} h_i^{v,(\ell)}, \qquad h_j^c = \mathrm{Concat}_{\ell=0}^{L} h_j^{c,(\ell)}.$$

A variable-wise MLP maps h_i^v to the initial logit $z_i^{(0)}$, from which $p_i^{(0)} = \sigma(z_i^{(0)})$ is obtained. The base predictors only require the variable output for solution generation. In SCoR, we additionally retain the concatenated constraint representations h_j^c, which provide the structural component of the state-conditioned constraint feedback described in the main text. All feedback-attention and variable-wise scaling modules are optimized jointly with the backbone.

A.2 Instantiation with DiffILO

Base DiffILO formulation. DiffILO parameterizes a distribution over binary solutions using the predicted marginals $p = \sigma(z)$, where $x_i \sim \mathrm{Bernoulli}(p_i)$ independently (Geng et al., 2025). At the formulation level, its unsupervised optimization balances the expected objective with expected constraint violations through a penalized probabilistic merit function, defined as

$$\mathcal{F}_{\mathrm{D}}(p) = c^\top p + \mu \sum_{j=1}^{m} \mathbb{E}_{x\sim p(\cdot|p)}\left[[a_j^\top x - b_j]_+\right]. \tag{20}$$

The objective term is evaluated directly under the Bernoulli marginals, whereas the constraint term depends on discrete samples from the predicted distribution. DiffILO employs a reparameterized sampling procedure to propagate the resulting optimization signal to the neural predictor. We use the same sampling mechanism as the original implementation.

DiffILO objective. For training, we use the normalized estimator adopted by the DiffILO implementation. Given K sampled binary solutions $\{x^{(k)}\}_{k=1}^K$, the objective and constraint terms are normalized to reduce variation across instances with different coefficient scales:

$$\mathcal{L}_{\mathrm{D}}(z) = \frac{c^\top p}{\max\{\|c\|_2, \epsilon\}} + \mu \frac{1}{|\mathcal{J}_+|} \sum_{j\in\mathcal{J}_+} \frac{\frac{1}{K}\sum_{k=1}^K [a_j^\top x^{(k)} - b_j]_+}{\max\{\|a_j\|_2, \epsilon\}}, \tag{21}$$

where $\mathcal{J}_+$ contains the constraints with nonzero sample-averaged violation. The penalty coefficient μ is adjusted during training according to the observed violation, following the base implementation.

Table 3: Input attributes of the variable-constraint bipartite graph.

Component	Attribute	Definition
Variable	Objective weight	Normalized cost coefficient associated with the variable.
	Mean coefficient	Average of its coefficients over the constraint rows.
	Incidence degree	Number of constraints involving the variable.
	Maximum coefficient	Largest coefficient associated with the variable across constraints.
	Minimum coefficient	Smallest coefficient associated with the variable across constraints.
Constraint	Mean coefficient	Average of the nonzero entries in the corresponding constraint row.
	Incidence degree	Number of variables participating in the constraint.
	RHS value	Normalized right-hand-side value of the constraint.
Edge	Coefficient	Matrix entry associated with the connected variable–constraint pair.

SCoR refinement for DiffILO. For each training instance, the GNN first produces the initial logits $z^{(0)}$ together with the retained variable and constraint representations. We evaluate the DiffILO objective at $z^{(0)}$ and obtain its logit-level gradient

$$g_{\mathrm{D}}^{(0)} = \nabla_z \mathcal{L}_{\mathrm{D}}(z)\big|_{z=z^{(0)}}.$$

In parallel, the current constraint state is evaluated directly at $p^{(0)} = \sigma(z^{(0)})$. The row-normalized positive residuals are instance-normalized as described in the main text and appended to the constraint representations. Sparse constraint-to-variable attention then propagates these states only along the original incidence edges, producing the localized feedback r_i for each variable.

The normalized DiffILO gradient, current prediction, structural variable representation, and localized feedback are passed to the SCoR scaling head. This produces the positive variable-wise scales η_i and the one-step refined logits

$$z^{\mathrm{ref}} = z^{(0)} - D_\eta \tanh\left(\bar{g}_{\mathrm{D}}^{(0)}\right). \tag{22}$$

Thus, the refinement does not introduce an additional optimization direction or a separate feasibility objective; the direction is inherited from the DiffILO loss, while the current constraint state determines how the update magnitude is distributed across variables.

The DiffILO objective is subsequently re-evaluated at z^{ref} using a new stochastic sample, and the model is trained with

$$\mathcal{L}_{\mathrm{train}}^{\mathrm{D}} = \mathcal{L}_{\mathrm{D}}\left(z^{\mathrm{ref}}; \xi_{\mathrm{ref}}\right) + \lambda_{\mathrm{aux}} \mathcal{L}_{\mathrm{D}}\left(z^{(0)}; \xi_0\right). \tag{23}$$

The auxiliary term preserves direct supervision of the initial predictor by the original unsupervised objective. In our implementation, the gradient signal and the residual state supplied to the refinement controller are stop-gradient features. This avoids introducing second-order differentiation through the base loss while retaining end-to-end learning of the backbone and refinement modules through the refined objective.

A.3 Instantiation with PUMA

Base PUMA formulation. PUMA retains the probabilistic ILP formulation of DiffILO but modifies the distribution explored during training through stochastic neighborhood perturbation (SNP) (Deng et al., 2026). Starting from a base sample $x \sim p(\cdot \mid p)$, PUMA flips each coordinate with probability ρ_t. With $\zeta_i \sim \mathrm{Bernoulli}(\rho_t)$, the perturbed sample is

$$x' = (1-\zeta) \odot x + \zeta \odot (1-x), \qquad x' \sim q_{\rho_t}(\cdot \mid x). \tag{24}$$

The perturbation magnitude follows a decaying schedule, allowing the training distribution to emphasize broader neighborhoods early in optimization and increasingly local neighborhoods later.

Up to an additive constant independent of the prediction, the SNP-based probabilistic merit function used by PUMA takes the form

$$\mathcal{F}_{\mathrm{P}}(p; \rho_t) = (1-2\rho_t) c^\top p + \mu\, \mathbb{E}_{\substack{x \sim p(\cdot|p) \\ x' \sim q_{\rho_t}(\cdot|x)}} \left[V(x')\right], \tag{25}$$

where $V(x) = \sum_j [a_j^\top x - b_j]_+$ denotes the aggregate constraint violation. PUMA estimates this expectation by first drawing base solutions and subsequently sampling perturbed neighbors around each of them.

Practical neighborhood objective. In the implementation used in our experiments, we draw K base solutions $\{x^{(k)}\}_{k=1}^K$ and generate M perturbed neighbors for each base solution. The unperturbed sample is retained together with its neighbors to form

$$\mathcal{N}_k = \{x^{(k)}, x^{(k,1)}, \dots, x^{(k,M)}\}.$$

Each candidate is evaluated using the normalized objective and constraint penalty. We aggregate the candidates associated with the same base sample through a soft minimum,

$$\mathcal{L}_{\mathrm{P}}(z; \rho_t) = \frac{1}{K} \sum_{k=1}^{K} \left[-\tau \log \sum_{x \in \mathcal{N}_k} \exp\left(-\frac{\ell_k(x)}{\tau} \right) \right], \tag{26}$$

where $\ell_k(x)$ denotes the normalized candidate-level merit within the k-th neighborhood and τ is the soft-min temperature. This retains PUMA's neighborhood-based exploration while favoring informative candidates within each sampled neighborhood.

SCoR refinement for PUMA. SCoR is inserted into PUMA without altering its neighborhood sampling or soft-min construction. A key difference from the DiffILO instantiation is that the relevant prediction state is the distribution after stochastic bit flipping. For the current perturbation probability ρ_t, the effective marginal of variable i is

$$\widetilde{p}_i^{(0)} = (1 - \rho_t) p_i^{(0)} + \rho_t (1 - p_i^{(0)}). \tag{27}$$

We therefore evaluate the state-conditioned constraint residuals using $\widetilde{p}^{(0)}$, rather than the unperturbed marginal $p^{(0)}$. This keeps the feedback consistent with the solution distribution actually explored by PUMA at the current training stage.

At the same time, the base direction is obtained from PUMA's own neighborhood objective,

$$g_{\mathrm{P}}^{(0)} = \nabla_z \mathcal{L}_{\mathrm{P}}(z; \rho_t)\big|_{z=z^{(0)}}.$$

The normalized gradient and the constraint feedback derived from $\widetilde{p}^{(0)}$ are combined by the same SCoR scaling module to produce a single variable-wise preconditioned update. Importantly, the state residual is used only to parameterize this update; it does not replace the sampled PUMA feasibility term or introduce an additional constraint loss.

After refinement, the complete PUMA sampling procedure is executed again from z^{ref}: new base solutions are sampled, stochastic neighborhoods are constructed using the current ρ_t, and the soft-min objective is re-evaluated. The resulting training objective is

$$\mathcal{L}_{\mathrm{train}}^{\mathrm{P}} = \mathcal{L}_{\mathrm{P}}\left(z^{\mathrm{ref}}; \rho_t, \xi_{\mathrm{ref}}\right) + \lambda_{\mathrm{aux}} \mathcal{L}_{\mathrm{P}}\left(z^{(0)}; \rho_t, \xi_0\right). \tag{28}$$

Consequently, PUMA continues to determine how the solution space is explored through stochastic neighborhoods, whereas SCoR acts within this optimization process by adapting the allocation of the resulting update across variables. The original perturbation schedule, neighborhood size, penalty schedule, and sampling procedure are otherwise left unchanged.

A.4 Solver-Assisted Inference

We provide additional details of the solver-assisted inference procedure introduced in Section 4.4. For an unseen instance, the trained predictor first produces the initial logits, after which the learned SCoR module performs the same refinement used during training to obtain z^{ref} and the refined marginal probabilities $p^{\mathrm{ref}} = \sigma(z^{\mathrm{ref}})$. The subsequent procedure is shared across the considered SCoR instantiations and consists of seed selection, confidence-based variable selection, and a restricted solver search.

Seed selection. We draw a collection of binary candidate solutions $\mathcal{X} = \{x^{(s)}\}_{s=1}^{N_s}$ independently from the refined Bernoulli distribution parameterized by p^{ref}. For each candidate, we evaluate its objective value and aggregate constraint violation

$$V(x) = \sum_{j=1}^{m} [a_j^\top x - b_j]_+.$$

If at least one sampled candidate is feasible, we select the feasible candidate with the best objective value as the seed $\hat{x}$. Otherwise, we choose the candidate with the smallest penalized merit,

$$\hat{x} = \arg\min_{x \in \mathcal{X}} \left\{ c^\top x + \mu_{\text{sel}} V(x) \right\}, \tag{29}$$

where μ_{sel} is used only for selecting among infeasible samples. This fallback allows the solver to receive a meaningful starting assignment even when direct sampling does not produce a feasible solution.

Confidence-guided variable selection. The selected seed determines a coherent binary assignment, while p^{ref} indicates the confidence associated with each component of that assignment. We partition the variables according to the seed,

$$\mathcal{I}_0 = \{i : \hat{x}_i = 0\}, \qquad \mathcal{I}_1 = \{i : \hat{x}_i = 1\}.$$

Among $\mathcal{I}_0$, we select up to k_0 variables with the smallest p_i^{ref}, corresponding to the most confident zero assignments. Among $\mathcal{I}_1$, we select up to k_1 variables with the largest p_i^{ref}, corresponding to the most confident one assignments. Denoting these sets by $\mathcal{P}_0$ and $\mathcal{P}_1$, respectively, the selected index set is $\mathcal{P} = \mathcal{P}_0 \cup \mathcal{P}_1$.

This seed-consistent selection is important because the sampled solution and the marginal probabilities play different roles. The seed provides a joint binary configuration, whereas the marginals determine which components of this configuration are sufficiently confident to constrain the subsequent search. We therefore do not construct the neighborhood solely from thresholded marginal probabilities.

Partial trust-region search. The complete seed $\hat{x}$ is supplied to the MILP solver as a warm start. In contrast, only variables in $\mathcal{P}$ participate in the trust-region constraint:

$$\sum_{i \in \mathcal{P}_0} x_i + \sum_{i \in \mathcal{P}_1} (1 - x_i) \le \Delta, \tag{30}$$

which is equivalent to $\sum_{i \in \mathcal{P}} |x_i - \hat{x}_i| \le \Delta$ for binary variables. The original ILP objective and constraints remain unchanged, and the solver optimizes the resulting restricted problem within the prescribed time budget.

The three inference parameters have distinct roles. The values k_0 and k_1 control how many confident zero and one assignments are used to define the restricted coordinates, while Δ controls how many of these selected assignments the solver may change. We use fixed (k_0, k_1, Δ) settings for each benchmark, with the specific values reported in the experimental configuration.

This construction differs from the all-variable local branching used in the original DiffILO and PUMA inference procedures, where the Hamming distance is measured with respect to the complete predicted solution. Here, the complete sampled seed is retained as the solver initialization, but the neighborhood restriction is imposed only on confidence-selected components. This separates the roles of initialization and search-space restriction: uncertain coordinates remain freely searchable, while the solver is encouraged to stay close to the most reliable components of the refined prediction. The resulting search rule is used for solver-assisted evaluation and is not intrinsic to SCoR; the refinement itself can be combined with other downstream decoding or search procedures.

B Proofs of the Refinement Properties

This section provides the proofs of the two propositions in Section 4.3. The first establishes the descent compatibility and boundedness of the refinement, while the second characterizes the allocation flexibility introduced by variable-wise scaling under a matched update budget. We follow the notation in the main text.

Proposition 4.1 (Descent-compatible and bounded refinement). Let $\delta z = z^{\mathrm{ref}} - z^{(0)}$ denote the refinement in Eq. 11. Then

$$\left\langle g^{(0)}, \delta z \right\rangle = -\sum_{i=1}^{n} \eta_i |g_i^{(0)}| \tanh\left(|\bar{g}_i^{(0)}|\right) \le 0, \qquad \|\delta z\|_\infty < \eta_{\max}.$$

If $g^{(0)} \neq 0$, then $\langle g^{(0)}, \delta z \rangle < 0$.

Proof. We first establish the alignment between the refinement and the negative base gradient. By the normalization in Eq. 9, let $\kappa = \max\{\frac{1}{n}\|g^{(0)}\|_1, \epsilon\} > 0$, such that $\bar{g}_i^{(0)} = g_i^{(0)}/\kappa$. The positivity of κ ensures that normalization preserves the sign of each gradient component.

Substituting $\delta z_i = -\eta_i \tanh(\bar{g}_i^{(0)})$ into the inner product and using the oddness of $\tanh(\cdot)$, we obtain

$$\begin{aligned}\left\langle g^{(0)}, \delta z \right\rangle &= -\sum_{i=1}^{n} \eta_i g_i^{(0)} \tanh\left(\bar{g}_i^{(0)}\right) \\ &= -\sum_{i=1}^{n} \eta_i |g_i^{(0)}| \tanh\left(|\bar{g}_i^{(0)}|\right) \le 0.\end{aligned}$$

The inequality follows since $\eta_i > 0$ and $\tanh(u) \ge 0$ for $u \ge 0$. Moreover, if $g^{(0)} \neq 0$, there exists at least one coordinate with $|g_i^{(0)}| > 0$. Since the corresponding $\tanh(|\bar{g}_i^{(0)}|)$ is also strictly positive, the inner product is strictly negative.

We next consider the magnitude of the refinement. For every coordinate, $|\delta z_i| = \eta_i |\tanh(\bar{g}_i^{(0)})|$. Combining $0 < \eta_i < \eta_{\max}$ with $|\tanh(u)| < 1$ for finite u gives $|\delta z_i| < \eta_{\max}$, and consequently $\|\delta z\|_\infty < \eta_{\max}$.

Finally, when the base objective is differentiable at $z^{(0)}$, its directional derivative along δz is given by

$$D\mathcal{L}_{\mathrm{base}}(z^{(0)})[\delta z] = \left\langle g^{(0)}, \delta z \right\rangle.$$

Hence, for $g^{(0)} \neq 0$, the refinement constitutes a descent direction for the base objective at the current prediction. The result characterizes the local behavior of the refinement without imposing an additional condition on the learned variable-wise scaling. □

Proposition 4.2 (Matched-budget variable-wise allocation). Consider a uniform scaling $\eta_i = \eta$ with $\eta \in [0, \eta_{\max}]$, and let $s_i^{\mathrm{uni}} = \eta \rho_i$ satisfy $\sum_i s_i^{\mathrm{uni}} = B$. Then $s^{\mathrm{uni}} \in \mathcal{S}_B$ and

$$\max_{s \in \mathcal{S}_B} \sum_{i=1}^{n} |g_i^{(0)}| s_i \ge \sum_{i=1}^{n} |g_i^{(0)}| s_i^{\mathrm{uni}}.$$

Moreover, the inequality is strict if there exist coordinates a and b such that

$$|g_a^{(0)}| > |g_b^{(0)}|, \qquad s_a^{\mathrm{uni}} < \eta_{\max}\rho_a, \qquad s_b^{\mathrm{uni}} > 0.$$

Proof. Recall that the refinement preserves the sign of the negative base gradient, with effective update magnitudes $s_i = \eta_i \rho_i$. For a given allocation s, the corresponding decrease predicted by the local linear model can therefore be written as

$$F(s) := -\left\langle g^{(0)}, \delta z \right\rangle = \sum_{i=1}^{n} |g_i^{(0)}| s_i.$$

We compare uniform and variable-wise scaling through this quantity under the same total update budget.

For uniform scaling, $s_i^{\mathrm{uni}} = \eta \rho_i$ with $0 \le \eta \le \eta_{\max}$ implies $0 \le_i^{\mathrm{uni}} \le \eta_{\max}\rho_i$ for every coordinate. Together with $\sum_i s_i^{\mathrm{uni}} = B$, these conditions establish $s^{\mathrm{uni}} \in \mathcal{S}_B$. Since $\mathcal{S}_B$ is nonempty and

compact, the continuous function F attains its maximum over this set. In particular, the uniform allocation is a feasible candidate, yielding

$$\max_{s \in \mathcal{S}_B} F(s) \geq F(s^{\mathrm{uni}}).$$

This establishes the non-strict inequality.

To establish the strict inequality, suppose that there exist coordinates a and b satisfying the stated conditions. The condition $s_a^{\mathrm{uni}} < \eta_{\max}\rho_a$ indicates that coordinate a has remaining allocation capacity, whereas $s_b^{\mathrm{uni}} > 0$ allows a positive amount of update magnitude to be removed from coordinate b. We can therefore redistribute a sufficiently small amount between these coordinates without changing the total budget.

Specifically, choose

$$0 < \tau < \min\left\{\eta_{\max}\rho_a - s_a^{\mathrm{uni}},\ s_b^{\mathrm{uni}}\right\},$$

and construct $s^\tau = s^{\mathrm{uni}} + \tau e_a - \tau e_b$, where e_i denotes the i-th standard basis vector. This transfers an update magnitude of τ from coordinate b to coordinate a, leaving all other coordinates unchanged.

By construction, the transferred magnitudes cancel in the total budget, so $\sum_i s_i^\tau = B$. The choice of τ also ensures that neither modified coordinate violates its lower or upper bound. In particular, $s_a^\tau < \eta_{\max}\rho_a$ and $s_b^\tau > 0$, while the remaining coordinates retain their original feasible values. Hence, $s^\tau \in \mathcal{S}_B$.

The change in the predicted decrease is

$$\begin{aligned} F(s^\tau) - F(s^{\mathrm{uni}}) &= \tau|g_a^{(0)}| - \tau|g_b^{(0)}| \\ &= \tau\left(|g_a^{(0)}| - |g_b^{(0)}|\right) > 0. \end{aligned}$$

The strict inequality follows from $|g_a^{(0)}| > |g_b^{(0)}|$ and $\tau > 0$. Since s^τ is feasible, we conclude that

$$\max_{s \in \mathcal{S}_B} F(s) \geq F(s^\tau) > F(s^{\mathrm{uni}}),$$

which completes the proof.

The construction shows that variable-wise scaling permits a redistribution of update magnitude that uniform scaling cannot express under a common scaling factor. The comparison concerns the admissible allocation set and its associated local linear decrease; it does not require the learned preconditioner to attain the maximizing allocation. □

C Experiment

The key backbone and training settings, including SCoR-specific parameters, are summarized in Table 4. Other base-method training settings follow the original DiffILO and PUMA implementations unless stated otherwise.

C.1 Benchmark Details

We evaluate SCoR on three widely used binary integer linear programming benchmarks: Combinatorial Auction (CA), Maximum Independent Set (IS), and Set Cover (SC). These problems involve distinct objective structures and combinatorial constraints, providing a diverse evaluation of the proposed refinement mechanism.

Combinatorial Auction (Leyton-Brown et al., 2000). The CA problem considers the allocation of indivisible items among competing bids. Given a set of items $\mathcal{I}$ and a collection of bids $\mathcal{B}$, each bid j specifies a bundle $S_j \subseteq \mathcal{I}$ with an associated value v_j. The objective is to maximize the total value of accepted bids, subject to the restriction that each item can be allocated at most once. A binary variable x_j indicates whether bid j is accepted.

Maximum Independent Set (Bergman et al., 2016). Given an undirected graph $G = (V, E)$, the IS problem seeks a subset of vertices with maximum cardinality such that no two selected vertices

Table 4: Backbone and training configurations across the three ILP benchmarks.

Parameter	SC	IS	CA	Description
Embedding dimension	32	32	32	Hidden dimension of the GNN representations.
GNN layers	3	10	10	Number of graph propagation layers.
Batch size	5	5	5	Number of ILP instances processed per batch.
Base samples (K)	15	15	15	Number of solution samples drawn per instance.
Training epochs	1,200	1,200	2,400	Maximum number of training epochs.
Optimizer	Adam	Adam	Adam	Optimization algorithm for network parameters.
Initial learning rate ($\times 10^{-5}$)	10	8	8	Initial step size of the Adam optimizer.
LR schedule period	1200	1200	2400	Period of the cosine learning-rate schedule.
Maximum refinement scale $\eta_{\max}$	0.9	0.9	0.9	Upper bound on each variable-wise refinement scale.
Auxiliary loss weight λ_{aux}	0.25	0.25	0.25	Weight of the base loss evaluated at the initial prediction.

are adjacent. Each binary variable x_i indicates whether vertex i belongs to the independent set, with the constraint $x_i + x_j \leq 1$ imposed on every edge $(i, j) \in E$.

Set Cover (Balas & Ho, 2009). The SC problem aims to cover a universe of elements using the smallest possible number of subsets. Given a universe $\mathcal{U}$ and a family of subsets $\mathcal{S} = \{S_1, \ldots, S_n\}$, the objective is to select a minimum-cardinality subcollection whose union covers every element in $\mathcal{U}$. Each binary variable x_j indicates whether subset S_j is selected, and every element must be contained in at least one selected subset.

The objectives of the three benchmarks can be summarized as

$$f_{\text{CA}}(x) = \sum_{j \in \mathcal{B}} v_j x_j, \quad f_{\text{IS}}(x) = \sum_{i \in V} x_i, \quad f_{\text{SC}}(x) = \sum_{j=1}^{n} x_j.$$

CA and IS are maximization problems, whereas SC is a minimization problem. For consistency with the ILP formulation in Section 3, maximization objectives are converted into minimization objectives by negating their coefficients.

C.2 Experiment details

Instance sizes. Each ILP is represented by a bipartite graph with binary-variable and constraint nodes. Table 5 reports the training and inference sizes for the standard benchmarks. SC and IS retain the same node counts, while CA has more constraints at inference. To examine generalization across instance sizes, we evaluate the models trained at the standard settings on the large-size instances in Table 6 without retraining.

Table 5: Training and inference instance sizes and inference search settings for the standard benchmarks.

Dataset	Training		Inference		
	Binary Variables	Constraint Number	Binary Variables	Constraint Number	(k_0, k_1, Δ)
SC	5000	3000	5000	3000	(0,100,40)
CA	1500	576	1500	2593	(0,80,40)
IS	1500	5943	1500	5943	(300,300,100)

Search neighborhood. The parameters k_0 and k_1 select confident zero and one assignments, while Δ bounds the number of changes among the selected variables. We also evaluate the full-variable Δ-only search used by DiffILO and PUMA in their original papers, setting $\Delta = 200$ for SC and $\Delta = 500$ for CA in this comparison. As shown in Table 7, SCoR improves both base methods on both tasks. On SC, however, neither refined variant surpasses Gurobi$_{3600}$. Improving a feasible cover may require coordinated changes to the selected subsets, which a Hamming bound over all

Table 6: Generalization inference instances and their search settings.

Dataset	Large size		
	Number of Binary Variables	Number of Constraint	(k_0, k_1, Δ)
SC	4000	5000	(0,100,40)
CA	5000	1951	(0,250,150)
IS	4000	199006	(800,800,250)

Table 7: Average objective on 100 test instances at 1,000 seconds using full-variable Δ-only search. Gurobi_{3600} is shown for reference. **Bold** marks the best result at 1,000 seconds.

Method	SC($\Delta = 200$)	CA($\Delta = 500$)
	Obj↓	Obj↑
Gurobi_3600	124.77	97395.20
Gurobi_1000	125.28	97199.85
DiffILO	125.27	97265.42
PUMA	125.23	97267.98
$\text{SCoR}_{\text{DiffILO}}$	125.21	**97306.23**
$\text{SCoR}_{\text{PUMA}}$	**125.18**	97280.75

variables can impede. This motivated the confidence-guided partial neighborhood used in our main experiments, where variables outside the selected set remain unrestricted.

C.3 Complementary Results of Main Experiment

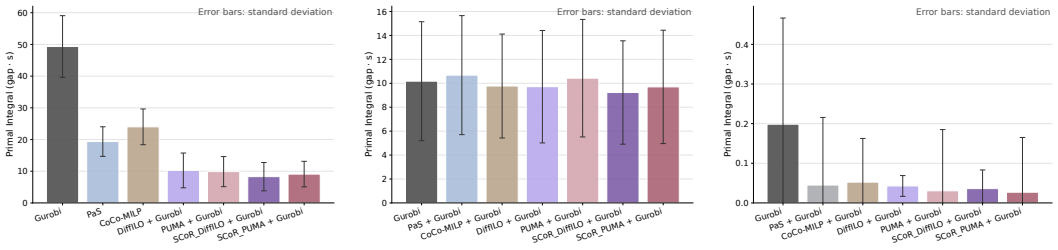
Primal integral. The final objective captures solution quality only at the end of the time limit. To assess the full search trajectory, we also report the primal integral, computed as the area under the relative primal gap curve over 1,000 seconds. Lower values indicate better primal performance over time. Figure 7 shows that both SCoR variants reduce the mean integral relative to their base methods on SC, with $\text{SCoR}_{\text{DiffILO}}$ attaining the lowest value. The same paired improvements appear on CA, although they are smaller relative to the variation across instances.

Prediction quality before solver search. We further examine the effect of SCoR on predictions before any solver search. For each test instance in SC, CA, and IS, we draw 1,000 samples from the DiffILO distribution before and after refinement as shown in Figure 8. We report the fraction of feasible samples, the best objective among feasible samples, and the relative change in that objective for instances with a feasible sample in both sets. These measurements characterize the sampled predictions rather than the final solutions returned by Gurobi.

On SC, the mean feasible sample rate rises from 31.05% to 54.69%, increasing on 94 of 100 instances. At least one feasible sample is found on 98 instances before refinement and all 100 afterward. Among the 98 instances with feasible samples on both sides, the mean best feasible objective decreases from 178.61 to 175.40; 71 instances improve, with a mean per-instance relative gain of 1.63%. Refinement therefore increases the frequency of feasible samples while also improving the best sampled objective on most comparable instances.

On CA, the mean feasible sample rate increases from 48.48% to 56.82%, with an increase on 76 of 100 instances. Among the 97 instances with feasible samples on both sides, the mean best feasible objective changes from -4317.14 to -4786.93 in the plotted minimization convention. This corresponds to a higher objective in the original maximization problem. The best sampled objective improves on 91 of these 97 instances, with a mean per-instance relative gain of 11.64%.

On IS, both distributions produce feasible samples on all 100 instances. The mean feasible sample rate changes only from 96.13% to 96.15%, increasing on two instances, decreasing on seven, and remaining unchanged on 91. In the plotted minimization convention, the mean best feasible objective changes from -666.98 to -667.06, corresponding to an increase from 666.98 to 667.06 in the original maximization problem. The best sampled objective improves on eight instances and ties on



(a) Set Covering (b) Combinatorial Auction (c) Maximum Independent Set

Figure 7: Primal integral over 1,000 seconds on SC, CA, and IS. Bars show the mean across 100 test instances, and error bars indicate standard deviation. Lower values indicate smaller gaps accumulated during search.

the other 92, yielding a mean per-instance relative gain of 0.012%. Thus, refinement has a modest effect on IS prediction quality, consistent with the nearly identical final solver objectives.

C.4 Extended Evaluation with SCIP

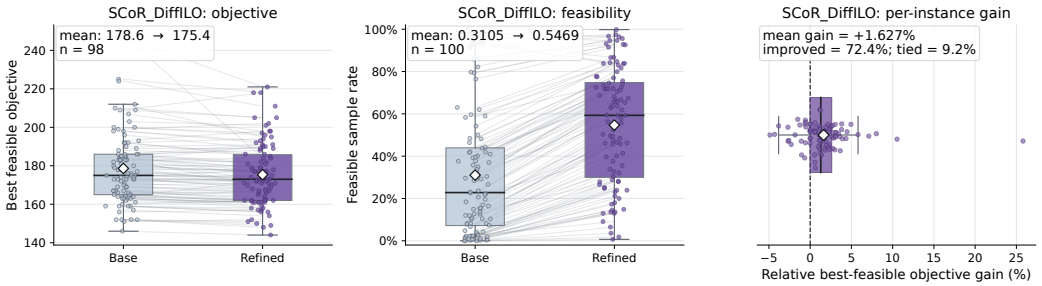
We additionally evaluate SCoR with SCIP under the same 1,000-second search budget to examine whether the gains observed with Gurobi extend to another downstream solver. As shown in Table 8, SCoR improves both base methods on SC and CA. On SC, $\text{SCoR}_{\text{PUMA}}$ achieves the best objective of 124.63, compared with 124.71 for PUMA and 125.69 for the SCIP3600 reference. On CA, $\text{SCoR}_{\text{DiffILO}}$ raises the objective from 96065.41 to 96123.21, the best result at 1,000 seconds. On IS, the refined variants tie with their base methods at the reported precision. These results show that the gains on SC and CA also carry over to SCIP.

Table 8: Average Obj and Gap_{abs} on 100 test instances using SCIP. **Bold** marks the best results at 1,000 seconds. Underlining marks the SCIP_{3600} reference used to initialize the instance-wise BKS.

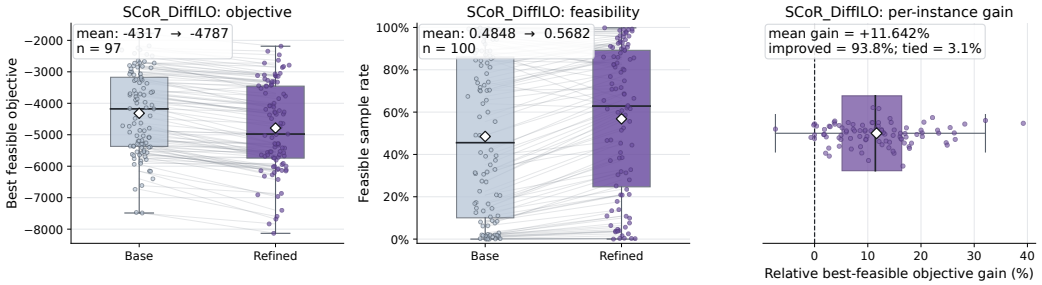
Method	SC(BKS:124.40)		CA(BKS:96771.87)		IS(BKS:684.10)	
	Obj↓	Gap_{abs} ↓	Obj↑	Gap_{abs} ↓	Obj↑	Gap_{abs} ↓
SCIP_3600	<u>125.69</u>	<u>1.29</u>	<u>96771.87</u>	<u>0.00</u>	<u>684.10</u>	<u>0.00</u>
SCIP_1000	127.05	2.65	96034.82	737.05	684.05	0.05
PaS	124.65	0.25	95909.14	862.73	684.05	0.05
CoCo-MILP	124.67	0.27	96091.75	680.12	**684.06**	**0.04**
DiffILO	124.66	0.26	96065.41	706.46	**684.06**	**0.04**
PUMA	124.71	0.31	96046.54	725.33	**684.06**	**0.04**
$\text{SCoR}_{\text{DiffILO}}$	124.65	0.25	**96123.21**	**648.66**	**684.06**	**0.04**
$\text{SCoR}_{\text{PUMA}}$	**124.63**	**0.23**	96060.48	711.39	**684.06**	**0.04**

Table 9: Average Obj and Gap_{abs} on 100 test instances with large sizes. **Bold** and *italics* mark the best and second-best results at 1,000 seconds, respectively.

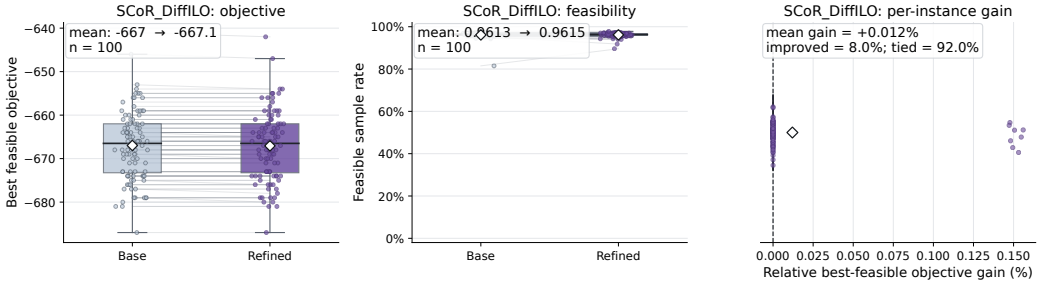
Method	SC(BKS:168.95)		CA(BKS:74108.73)		IS(BKS:1679.45)	
	Obj↓	Gap_{abs} ↓	Obj↑	Gap_{abs} ↓	Obj↑	Gap_{abs} ↓
Gurobi_{3600}	<u>169.74</u>	<u>0.79</u>	<u>74008.09</u>	<u>100.64</u>	<u>1676.55</u>	<u>2.90</u>
Gurobi_{1000}	170.38	1.43	73876.47	232.26	1664.24	15.21
PaS	169.45	0.50	73586.43	522.30	1678.76	0.69
CoCo-MILP	*169.43*	*0.48*	73841.10	267.63	**1679.04**	**0.41**
DiffILO	169.58	0.63	73895.66	213.07	1677.79	1.66
PUMA	169.48	0.53	73878.02	230.71	1678.63	0.82
$\text{SCoR}_{\text{DiffILO}}$	169.49	0.54	**73900.06**	**208.67**	1678.38	1.07
$\text{SCoR}_{\text{PUMA}}$	**169.42**	**0.47**	*73897.11*	*211.62*	*1678.78*	*0.67*



(a) Set Covering



(b) Combinatorial Auction



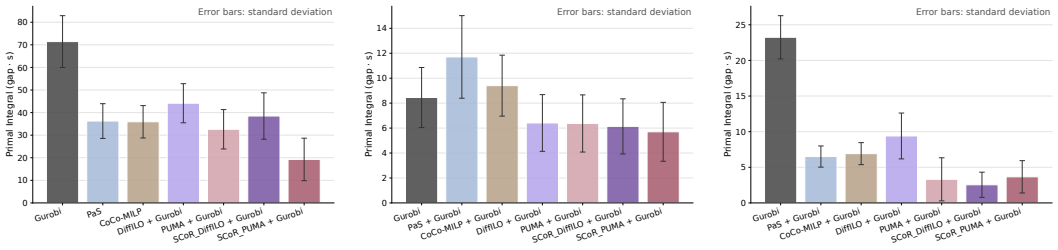
(c) Maximum Independent Set

Figure 8: Prediction quality before solver search on three benchmarks. For each task, the left and middle panels compare the best feasible objective and feasible sample rate before and after SCoR refinement. Lines connect results from the same instance, boxes show their distributions, and diamonds mark the means. The right panel shows the per-instance relative gain in the best feasible objective, with the dashed line marking zero gain. Objective and gain panels include only instances with feasible samples on both sides.

C.5 Generalization

Large Size Generalization. We evaluate generalization at large size on test instances with more variables, more constraints, or both than those used for training, as detailed in Table 6. Table 9 reports the mean final objective and absolute primal gap over 100 instances after 1,000 seconds of search. The mean instance-wise BKS values are 168.95, 74108.73, and 1679.45 for SC, CA, and IS, respectively. Figure 5 shows the corresponding relative primal-gap trajectories.

Both SCoR variants improve the final objectives of their respective base methods across all three tasks. On SC, $\text{SCoR}_{\text{PUMA}}$ achieves the lowest mean objective among the evaluated methods at 169.42, leaving an absolute gap of 0.47 to the BKS. On CA, $\text{SCoR}_{\text{DiffILO}}$ raises the mean objective from 73895.66 to 73900.06 and obtains the strongest final result among learning-based methods. On IS, $\text{SCoR}_{\text{PUMA}}$ improves PUMA from 1678.63 to 1678.78 and ranks second behind CoCo-MILP. $\text{SCoR}_{\text{DiffILO}}$ also improves upon DiffILO, from 1677.79 to 1678.38.



(a) Set Covering (b) Combinatorial Auction (c) Maximum Independent Set

Figure 9: Primal integrals on test instances with generalization size for three benchmarks. Each bar shows the mean integral of the relative primal gap to the instance-wise BKS over 1,000 seconds and 100 test instances. Error bars indicate standard deviations. Lower values indicate smaller gaps accumulated during search.

Figure 9 evaluates performance throughout the 1,000-second search using the primal integral. $\text{SCoR}_{\text{PUMA}}$ has the lowest mean integral on SC and CA, while $\text{SCoR}_{\text{DiffILO}}$ has the lowest on IS. The IS ranking differs from the final objectives in Table 9, where CoCo-MILP ranks first and $\text{SCoR}_{\text{PUMA}}$ ranks second. The primal integral accumulates relative gaps over time, so a method that maintains smaller gaps during search can have a lower integral without producing the best objective at the endpoint. These results show that the two refined predictors can benefit different parts of the search trajectory under a change in instance size.

MIPLIB. Following PUMA (Deng et al., 2026), we use the capacitated vertex separator (CVS) subset of MIPLIB 2017. The instances `cvs08r139-94`, `cvs16r70-62`, and `cvs16r89-60` are used for training, while `cvs16r106-72` and `cvs16r128-89` are held out for testing. We follow the same solver-assisted inference protocol and compare final objective values after 1,000 seconds. On `cvs16r106-72`, all methods attain -81. On `cvs16r128-89`, $\text{SCoR}_{\text{DiffILO}}$ improves upon both Gurobi and DiffILO, reaching -97 instead of -96. This matches the objective reported by PUMA on the same instance and shows that the refinement can improve DiffILO on a held-out CVS instance.

Table 10: Final objective values on the held-out CVS instances after 1,000 seconds.

Instance	Gurobi	DiffILO	$\text{SCoR}_{\text{DiffILO}}$
cvs16r106-72	-81.00	-81.00	-81.00
cvs16r128-89	-96.00	-96.00	-97.00

C.6 Sensitivity Analysis

We follow the original hyperparameter choices of the DiffILO and PUMA base methods and focus this analysis on parameters introduced by SCoR. We vary the maximum refinement scale $\eta_{\max}$ while holding the remaining settings fixed, and use 0.9 for all three tasks. This choice gives the best objective among the tested values on SC (124.56) and CA (97316.01). The SC difference from 1.2 is small, whereas CA shows a clearer advantage for 0.9 over both 0.6 and 1.2. On IS, all three settings yield 684.14 at the reported precision, so the results do not distinguish their effects on that task.

Table 11: Sensitivity to the maximum refinement scale $\eta_{\max}$.

Method	SC Obj↓	CA Obj↑	IS Obj↑
η_{max} : 0.6	124.66	97236.17	**684.14**
η_{max} : 0.9	**124.56**	**97316.01**	**684.14**
η_{max} : 1.2	124.57	97231.84	**684.14**