PREFERENCE-GUIDED TREE SEARCH FOR FEASIBLE SOLUTION GENERATION IN INTEGER LINEAR PROGRAMMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Integer linear programming (ILP) arises in many critical decision-making tasks. While recent learning-based methods have shown promise in producing feasible solutions for ILPs, many rely on variable predictions that, on their own, do not necessarily yield high-quality solutions or even feasible solutions during inference. To address this gap, we propose a preference-guided tree search (PgTS) framework for high-quality feasible solution generation. Building on an underlying tree search that guarantees finding a feasible solution, PgTS jointly learns two guidance components to improve search efficiency and solution quality: reference values, which indicate preferred variable assignments, and variable ordering, which reflects the sequence in which variables are explored along the search. To align training with inference, PgTS evaluates each guidance pair through the same tree-search procedure used at inference, so that the learning signal directly reflects the quality of the resulting feasible solution. Based on the search outcomes, objective-based preference learning derives contrastive signals from their objective values to favor better guidance pairs, without requiring optimal-solution labels. Extensive experiments across ILP benchmarks show that PgTS achieves state-of-the-art solution quality and provides strong initial solutions that improve long-term solver performance over conventional and learning-based methods.

1 INTRODUCTION

Integer linear programming (ILP) provides a foundational mathematical framework for formulating decision-making problems that arise across a wide range of real-world applications, such as scheduling (Deenen et al., 2023), transportation (Yang et al., 2026), energy systems (Wirtz et al., 2021), and network design (Magnanti & Wong, 1984). Despite its expressive power, solving ILP is NP-hard, and large-scale instances can be prohibitively time-consuming in practice. At the same time, ILP instances from the same application domain often exhibit recurring structural patterns. This observation has sparked growing interest in machine learning (ML) methods for ILP solving, where patterns are learned from historical instances and exploited to improve solution efficiency on new instances from the same domain (Bengio et al., 2021).

Learning-based solution generation methods for ILPs aim to learn variable assignments that are close to optimal feasible solutions, and can be broadly categorized into prediction-guided search methods and feasibility-aware solution generation methods. Prediction-guided search methods typically learn to imitate optimal or high-quality solutions by predicting variable values. The high-confidence predicted values are assigned to a subset of variables, thereby restricting the search space or yielding a reduced ILP that can be solved more efficiently (Nair et al., 2021; Han et al., 2023; Liu et al., 2025). However, even high-confidence assignments do not inherently satisfy all constraints of the original problem and can cause infeasibility. Feasibility-aware solution generation methods address this issue by incorporating constraint violations into the loss function or reward design during training, thereby improving the empirical feasibility rate (Zeng et al., 2024; Geng et al., 2025; Lee & Kim, 2026). However, they still decouple predictions from the solution search process and rely on external post search to enforce feasibility.

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.

To address the aforementioned limitation, we propose a novel preference-guided tree search (PgTS) framework that guides the tree search towards high-quality feasible solution generation. Inspired by classical backtracking and tree search methods for constraint satisfaction problems (Haralick & Elliott, 1980; Cappart et al., 2025), we use a tree search with backtracking as the base, which can guarantee finding a feasible solution. The tree search relies on two key guidance components: reference values and variable ordering. The variable ordering specifies the sequence in which variables are assigned, and the reference values specify the preferred assignment values to try first. In contrast to prior methods that directly output variable assignments based on predicted values, the tree search incrementally assigns variables while checking feasibility at each step and backtracks when constraint violations occur. Hence the solution obtained via tree search is guaranteed to be feasible.

Although the underlying tree search enforces feasibility, the quality of the generated feasible solution mostly depends on how the search is guided by the two critical components: reference values and variable ordering. Reference values that are close to feasible high-quality solutions help steer the tree search toward promising variable assignments, while variable ordering enables efficient recovery from imperfect reference values through backtracking. Enhancing these two components is naturally suited to objective-based preference learning: obtaining optimal solutions for ILP instances can be computationally expensive, whereas comparing feasible solutions (once they are generated through tree search) by their objective values can be readily performed. In the PgTS framework, we employ a graph neural network with two output heads: a value head sampling reference values and an ordering head determining the assignment sequence of variables. During training, we sample multiple candidate pairs from the two heads, generate feasible solutions via tree search under their paired guidance, and evaluate the objective values of the resulting solutions. The neural network is optimized by adjusting joint likelihoods of the pairs according to objective values in their resulting solutions, rewarding pairs that yield better solutions and penalizing those that lead to poorer ones.

Our contributions are threefold. First, we propose PgTS, a preference-guided tree search framework that jointly learns reference values and variable ordering to guide a tree search toward high-quality feasible solutions. Second, we align training with inference by evaluating each guidance pair through the same tree-search procedure employed at inference, ensuring the learning signal directly reflects the quality of the returned first feasible solution. Third, we introduce an objective-based preference learning approach that compares the guidance pairs of reference values and variable ordering through the quality of their resulting feasible solutions, without requiring optimal-solution labels. Extensive experiments across ILP benchmarking demonstrate that PgTS consistently generates high-quality feasible solutions and achieves strong long-term performance.

2 Related Work

2.1 Learning-augmented Branch-and-Bound

Branch-and-bound, often implemented together with cutting planes and primal heuristics, is the core algorithmic framework in typical ILP solvers. Its performance depends heavily on a set of heuristic rules, and machine learning (ML) has been used to improve different heuristic components of branch-and-bound-based solvers (Scavuzzo et al., 2024; Zhang et al., 2023). A significant body of research focuses on improving branch-and-bound search to reduce the size of search trees for solving ILPs. These improvements include branching variable selection, where ML policies choose variables to branch on (Gasse et al., 2019; Gupta et al., 2020; Zarpellon et al., 2021; Shen et al., 2021; Gupta et al., 2022; Scavuzzo et al., 2022; Wang et al., 2026; Mhamed et al., 2026), and node selection, where ML policies decide which open branch should be explored next (He et al., 2014; Labassi et al., 2022; Zhang et al., 2025). Beyond these direct search decisions in branch-and-bound solvers, ML has also been applied to other modules, including cutting-plane selection and generation (Tang et al., 2020; Paulus et al., 2022; Wang et al., 2023; Li et al., 2023; Ling et al., 2024; Cheng & Basu, 2024; Deza et al., 2025), presolving and model reduction (Liu et al., 2024; Li et al., 2025; 2026b), primal heuristic scheduling (Khalil et al., 2017; Chmiela et al., 2021), and neighborhood design (Huang et al., 2023; Wu et al., 2021). In contrast to these works, mainly improving internal decisions of a solver, we focus on jointly learning reference values and variable ordering to guide a separate feasibility-guaranteed tree search, which directly constructs high-quality feasible solutions that can also serve as strong warm-starts for long-term ILP solving.

2.2 Learning-based Solution Generation

Beyond learning within branch-and-bound, another line of work learns to predict complete or partial variable assignments in ILP instances. Current learning-based solution generation methods are broadly divided into prediction-guided search methods and feasibility-aware solution generation methods. Prediction-guided search methods typically learn from optimal or high-quality solutions and employ predictions to fix variables, thereby constructing reduced problems and accelerating the subsequent search for feasible solutions (Ding et al., 2020; Nair et al., 2021; Han et al., 2023; Huang et al., 2024; Liu et al., 2025). This idea has been implemented in different ways. Neural Diving generates multiple partial assignments for integer variables and solves the resulting smaller ILPs with an off-the-shelf solver (Nair et al., 2021). Predict-and-Search predicts marginal probabilities of variables and searches for feasible solutions within a neighborhood around the predicted solution (Han et al., 2023). Apollo further improves this paradigm by alternating prediction and correction: it iteratively uses trust-region search to obtain corrected reference solutions and then fixes variables whose predictions are estimated to be reliable (Liu et al., 2025). Although effective, these methods mainly rely on predictions to approximate variable assignments. The predictions themselves do not inherently guarantee that constraints are satisfied, and therefore, solver-based correction, refinement, or additional search is usually required to recover feasible solutions. Instead, feasibility-aware solution generation methods attempt to account for feasibility during solution generation. For example, Guided Diffusion incorporates constraints and objectives into the sampling process (Zeng et al., 2024), while DiffILO reformulates ILPs into continuous unconstrained problems and trains the model through an unsupervised learning objective (Geng et al., 2025). These approaches improve the empirical feasibility rate of generated solutions but still do not provide a guarantee that the learned outputs are feasible. To remedy this issue, we design a preference learning-based framework, in which feasible solutions are guaranteed via tree search, and the resulting search outcomes are naturally used as learned signals to enhance the solution quality.

3 Preliminaries

An integer linear programming (ILP) problem can be formulated as follows:

$$\min_x \{g(x) = c^T x \mid Ax \leq b, x \in \mathbb{Z}^n\}, \tag{1}$$

where x denotes the n-dimensional vector of integer decision variables, $c \in \mathbb{R}^n$ is the vector of objective coefficients, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ denote the coefficient matrix and the right-hand-side values of the constraints, separately. A solution is feasible when it satisfies all constraints. In this paper, we focus on ILP with binary variables following most prior literature (Nair et al., 2021; Han et al., 2023; Geng et al., 2025; Liu et al., 2025), i.e., $x \in \{0, 1\}^n$. They still capture a broad and representative class of ILP since all bounded integer variables can be binarized (Dantzig, 2016). Without loss of generality, we assume the objective function $c^T x > 0$ for all feasible x.

Bipartite graph representations are commonly used to encode the structural information of ILP instances. It was first introduced in (Gasse et al., 2019), where one set of vertices represents variables and the other disjoint set represents constraints, with edges encoding variable-constraint incidences in the coefficient matrix. This graph representation provides a natural interface for the bipartite graph neural network (GNN), which typically processes the bipartite graph through two half-convolutions: one from variable vertices to constraint vertices and the other from constraint vertices back to variable vertices. Subsequent learning-based methods for ILPs are largely built on similar bipartite graphs and neural networks. More details of them are provided in App. C.

4 PgTS: Preference-guided Tree Search

Strictly satisfying constraints in ILP remains challenging for learning-based solution generation methods. To bridge this gap, a preference-guided tree search framework (PgTS) is proposed, consisting of two main components: the feasibility-guaranteed tree search and the objective-based preference learning. The overview of PgTS is illustrated in Figure 1.

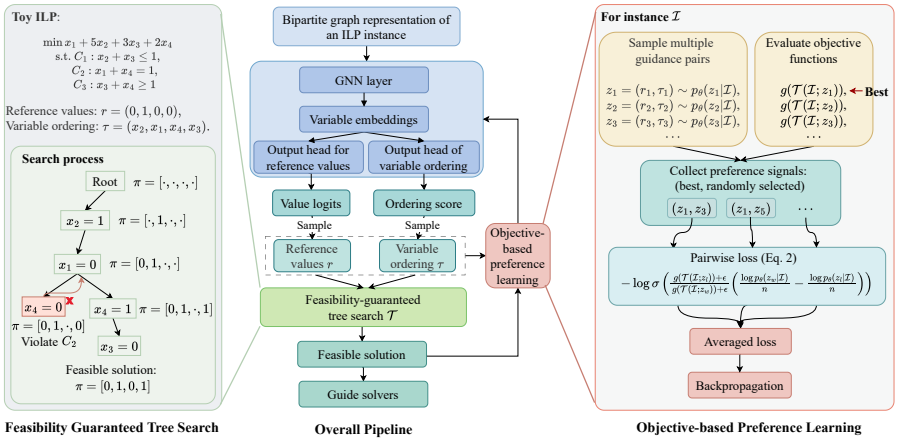


Figure 1: **Overview of PgTS.** As shown in the pipeline (**the middle**), the bipartite graph representation of an ILP instance is first encoded by a GNN to obtain variable embeddings. Then, two output heads generate value logits and ordering scores, which are used to sample reference values and variable ordering, respectively. The sampled values and ordering guide the feasibility-guaranteed tree search to generate feasible solutions (**the left**). During training (**the right**), we sample multiple candidate guidance pairs (reference values and variable ordering), guiding tree search to generate feasible solutions, and evaluate their objective values. Preference learning with a pairwise loss is used to train the model, increasing the likelihood of guidance pairs that lead to better solutions.

4.1 Feasibility-guaranteed Tree Search

A tree search is adopted to enforce solution feasibility in the PgTS framework. The learned binary reference values $\mathbf{r} = (r_1, \ldots, r_n)$ and variable ordering $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$ are taken as input, guiding the tree search process. The variable ordering specifies the sequence in which values are assigned to variables, while the reference values indicate which values should be assigned first to each variable.

Starting from an empty assignment, the tree search proceeds in a depth-first manner. At depth i, it selects the next variable τ_i according to the variable ordering and first assigns it the corresponding reference value r_{τ_i}. The resulting partial assignments are then checked against the ILP constraints. Specifically, for each constraint, the lower bound of the left-hand side is calculated and checked to determine whether it already exceeds the right-hand-side value. If no constraint violation is detected, the tree search proceeds to the next variable at depth $i+1$. Otherwise, the violated constraints cannot be satisfied by continuing variable assignments. In this case, the search backtracks and tries the alternative assignment $(1 - r_{\tau_i})$. This process continues until all feasible variable assignments are finished or until all branches of the search tree have been exhausted (indicating no feasible solutions exist). Uninterrupted execution of the tree search procedure is guaranteed to return a feasible solution whenever the input ILP instance is feasible (see the proof in App. A). The pseudocode and implementation details are provided in App. C.1.

The worst-case complexity of tree search is exponential in the number of variables, since exhaustive exploration may require visiting all possible variable assignments. In practice, however, the search cost and the quality of the generated feasible solution depend heavily on the guidance of reference values and variable ordering. Reference values specify the values assigned to variables. When these values are close to high-quality feasible solutions, they guide the tree search toward a promising region of solution space and reduce backtracks to restore feasibility. Ideally, reference values coincide exactly with a high-quality solution, such that this solution is always reached by tree search regardless of variable ordering. Variable ordering determines the sequence in which variables are assigned values and therefore influences the efficiency of both pruning and backtracking. When reference values correspond to infeasible solutions but remain close to good feasible ones, an effective variable ordering can steer backtracking toward assignments that quickly restore feasibility while preserving most reference values (and thus the objective value). These intuitions motivate the learning of reference values and variable ordering in the next subsection. Please refer to App. B for examples illustrating how reference values and variable ordering guide the search process.

4.2 Objective-based Preference Learning

To provide effective tree-search guidance, we optimize reference values and variable ordering jointly through objective-based preference learning. For each ILP instance $\mathcal{I}$, we encode it as a bipartite graph $\mathcal{G}(\mathcal{I})$, as introduced in Sec. 3. A bipartite GNN processes $\mathcal{G}(\mathcal{I})$ and produces variable embeddings. We build two output heads based on the embeddings: a value head that parameterizes Bernoulli distributions over binary reference values, and an ordering head that produces scores for variables, which determine the assignment sequence of variables (i.e., variable ordering). Details of the bipartite graph representation, GNN, and output heads are provided in App. C.2 and App. C.3.

Let $z = (\mathbf{r}, \boldsymbol{\tau})$ denote a guidance pair obtained from the output heads, with $\mathbf{r} = (r_1, \ldots, r_n)$ for reference values and $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$ for variable ordering. Each reference value r_i is sampled from the Bernoulli distribution predicted by the value head. The variable ordering $\boldsymbol{\tau}$ is sampled according to scores from the ordering head using sequential softmax sampling without replacement. Given a pair z, the proposed tree search $\mathcal{T}$ returns its first complete feasible solution $\mathcal{T}(\mathcal{I}; z)$.

Most prior methods obtain optimal solutions for ILP instances to supervise solution generation (Han et al., 2023; Huang et al., 2024; Liu et al., 2025), which can be computationally expensive. Alternatively, we propose to compare generated feasible solutions with their objective values, and develop an effective preference learning framework to construct objective-based preferences over candidate guidance pairs. For an instance $\mathcal{I}$, each sampled pair z (reference values and variable ordering) is used to guide the feasibility-guaranteed tree search and return a feasible solution $\mathcal{T}(\mathcal{I}; z)$. The corresponding objective value $g(\mathcal{T}(\mathcal{I}; z))$ provides a direct criterion for comparing different candidate guidance pairs. Assuming candidate guidance pairs z_w and z_l, z_w is preferred to z_l if z_w leads to a better generated feasible solution, i.e., $g(\mathcal{T}(\mathcal{I}; z_w)) < g(\mathcal{T}(\mathcal{I}; z_l))$ for minimization problems. Given the two pairs (z_w, z_l), we define the following pairwise loss for preference learning:

$$\ell_{\text{pref}}(\theta; \mathcal{I}, z_w, z_l) = -\log \sigma \left(\frac{g(\mathcal{T}(\mathcal{I}; z_l)) + \epsilon}{g(\mathcal{T}(\mathcal{I}; z_w)) + \epsilon} \left(\frac{\log p_\theta(z_w \mid \mathcal{I})}{n} - \frac{\log p_\theta(z_l \mid \mathcal{I})}{n} \right) \right), \quad (2)$$

where $p_\theta(z \mid \mathcal{I})$ is the joint likelihood of the sampled reference values and variable ordering obtained from the two output heads, σ is the sigmoid function, and ϵ is a small constant for numerical stability.

For each training instance, we first sample candidate guidance pairs and evaluate them via tree search. From them, we select pair z^* whose returned feasible solution has the best objective value as an anchor and randomly select $\{z_1, \cdots, z_k\}$ from the remaining pairs, where k is set to 8 in our experiments. Then we construct a collection of preference signals, $\mathcal{P}(\mathcal{I}) = \{(z^*, z_1), \cdots, (z^*, z_k)\}$, and calculate the loss for instance $\mathcal{I}$ by averaging pairwise loss over the collection:

$$\mathcal{L}_{\text{pref}}(\theta; \mathcal{I}) = \frac{1}{|\mathcal{P}(\mathcal{I})|} \sum_{(z_w, z_l) \in \mathcal{P}(\mathcal{I})} \ell_{\text{pref}}(\theta; \mathcal{I}, z_w, z_l). \quad (3)$$

Since $p_\theta(z \mid \mathcal{I})$ contains both the likelihood of the reference values and the likelihood of the variable ordering, the preference learning is used to jointly train the value head and the ordering head to assign higher probability to tree-search guidance that leads to better generated feasible solutions. The preference target is determined by the objective value of the feasible solution returned by the same search procedure used at inference, thereby aligning learning with solution quality after search.

4.3 Parameter Initialization

Since the efficiency of the proposed tree search strongly depends on the quality of the reference values, we initialize part of the GNN and the value head using a pretrained ILP solution prediction model. More specifically, we load the graph convolution layers and the prediction head from DiffILO (Geng et al., 2025) as the warm-start parameters. This initialization provides informative reference-value predictions at the beginning of preference learning. The ordering head is then trained jointly with the value head by the objective-based preference learning framework, thereby potentially enhancing the performance of the solution prediction model. The parameter initialization is not tied to DiffILO: any model that learns variable embeddings and produces reference-value predictions can in principle be used to initialize the corresponding components of our neural network. As shown in App. E.2, initializing our framework with other pretrained solution prediction models also leads to substantially better feasible solutions than their direct predictions.

4.4 Inference and Solver Integration

During inference, the trained model is used to sample multiple candidate guidance pairs consisting of reference values and variable orderings. Each pair is passed to the proposed tree search to obtain a feasible solution. As presented in the experiments, the obtained feasible solutions themselves are already competitive feasible solutions. Moreover, they are capable of improving long-term performance of downstream ILP solvers/algorithms by serving as warm-start solutions or by guiding search-region reduction. In our implementation, we follow the neighborhood-generation strategy in DiffILO (Geng et al., 2025) to reduce the search region around the generated feasible solution and provide the solution as the initial incumbent. Given a generated feasible solution $\hat{x}$, the Hamming neighborhood is constructed by adding a local branching constraint:

$$\sum_{\hat{x}_i=0} x_i + \sum_{\hat{x}_i=1} (1 - x_i) \leq \Delta, \tag{4}$$

to the original ILP instance, where Δ is a hyperparameter controlling the maximum number of binary variables allowed to differ from $\hat{x}$. This restricts the downstream solver (such as Gurobi or SCIP) to a smaller region while allowing the incumbent to be improved.

5 Experiments

We present empirical results demonstrating the effectiveness of the proposed PgTS in feasible solution generation (Sec. 5.2 and Sec. 5.4), long-term performance integrated with solvers (Sec. 5.3), and generalization performance (Sec. 5.5). Further evaluations are also provided, including ablation studies (App. E.2), sensitivity analysis (App. E.3), and real-world applicability (App. E.5). More details of training and inference are given in App. D.

5.1 Setup

Benchmarks We evaluate our method mainly on three widely used ILP benchmarks: combinatorial auction (CA) (Leyton-Brown et al., 2000), maximum independent set (IS) (Bergman et al., 2016), and set covering (SC) (Balas & Ho, 2009). Details are provided in App. D.1.

Baselines We select baselines to cover both conventional exact solvers and the two main classes of learning-based methods discussed in Sec. 2.2. First, we include SCIP (Hojny et al., 2025) and Gurobi (Gurobi Optimization, LLC, 2026), the representative open-source and commercial exact solvers, respectively. Second, we include four prediction-guided search methods, including Predict-and-Search (PaS) (Han et al., 2023), Contrastive Predict-and-Search (ConPaS) (Huang et al., 2024), Apollo (Liu et al., 2025), and FMIP (Li et al., 2026a). Third, we include feasibility-aware solution generation methods, including Diffusion (Zeng et al., 2024) and DiffILO (Geng et al., 2025). To ensure a fair comparison, we evaluate all baselines on the same hardware platform as our method.

Evaluation protocol We evaluate all methods in two settings: direct feasible solution generation and long-term solving performance. The former aims to assess PgTS's ability to directly generate high-quality solutions. For each test instance, PgTS creates five candidate guidance pairs to execute the feasibility-guaranteed tree search, resulting in the corresponding solutions for assessment. Feasibility-aware solution generation baselines are included for comparison as they also aim to generate feasible solutions. We implement their models and also generate five predicted solutions per instance for fairness. In addition, we compare PgTS with exact solvers that are evaluated by the first feasible solutions they find. We run each solver only once per test instance, as they generally produce the same results across multiple runs under the same configuration. When using SCIP, we disable trivial heuristics to avoid trivial feasible solutions[1], while Gurobi avoids such trivial solutions by default. By comparing with the first solutions from exact solvers, we aim to highlight the advantage of learning-based methods as feasible solution seekers. We do not include prediction-guided search methods because they are not designed as standalone feasible solution generation methods.

[1]The trivial heuristic in SCIP is applied at a very early stage of the solving process, attempting to generate feasible solutions using simple global assignments, such as all zeros. While efficient, it typically yields poor-quality solutions. We therefore disable it to allow SCIP to focus on finding higher-quality solutions.

Long-term performance comparison evaluates the effectiveness of PgTS in guiding exact solvers to run for a long time. We use the solutions generated by our method to guide Gurobi and SCIP as described in Sec. 4.4. We include all learning-based baselines for a comprehensive comparison. We evaluate these baselines in a similar manner: using their predicted or generated solutions to guide Gurobi and SCIP, following their specific solver-guidance procedures. Gurobi and SCIP are also evaluated as stand-alone exact solver baselines. They are run directly on the original test instances under the same time budget. In all long-term solving experiments, the time limit is set to 1000 seconds, and solver performance is recorded at 10, 100, and 1000 seconds.

Metrics We use different metrics for evaluating feasible solution generation and guiding long-term performance. For direct feasible solution generation, we report feasible ratio (FR), running time (Time), and primal gap (PG). FR measures the fraction of test instances for which a method returns at least one feasible solution, and Time records the wall-clock time in seconds. PG measures the percentage relative gap between the objective value of a returned solution and the best-known objective value, denoted by BK. In our experiment, BK is obtained by solving each test instance with Gurobi under a 10-hour time limit, under which optimality is obtained for most test instances. Following the primal-gap convention used by SCIP, we define

$$\text{PG} = \begin{cases} \frac{|OBJ-BK|}{\max(|OBJ|,|BK|,10^{-10})} \times 100, & \text{if a feasible solution is available,} \\ 100, & \text{otherwise,} \end{cases} \tag{5}$$

where OBJ is the objective value of the returned feasible solution or the best feasible solution found so far. By the positive-objective assumption introduced in Sec. 3, PG lies between 0 and 100, with lower values indicating objective values closer to BK. Infeasible outputs in direct feasible solution generation, as well as solver states before any feasible solution is found, are assigned PG $= 100$, corresponding to the worst value under this metric.

For evaluating long-term performance, we report the PG together with the primal integral (PI) at three time points: 10, 100, and 1000 seconds. PI is defined as the integral of the PG over the solving time. A lower PI indicates better overall anytime performance, meaning the solver rapidly reduces the primal gap and spends less total time sitting on poor-quality solutions.

5.2 Feasible Solution Comparison

This experiment investigates the effectiveness of our method in directly producing high-quality feasible solutions. Table 1 reports the feasible ratio, primal gap, and running time on the three benchmarks. The results show that our method is the only evaluated learning-based method that achieves a 100% feasible ratio on all benchmarks. Beyond feasibility, our method also produces substantially higher-quality feasible solutions. It achieves the lowest primal gap on all three benchmarks. The improvement is particularly pronounced on SC, where the primal gaps of all baselines are above 50, while our method reaches 11.61, much closer to the best-known objective values. These results demonstrate the superior performance of PgTS in the quality of its generated solutions.

Although it incurs additional search time in several settings, PgTS remains sub-second on all benchmarks while consistently producing feasible solutions with substantially lower primal gaps. The additional search cost is a favorable trade-off when high-quality feasible solutions are needed in a short time. In App. E.4, we further show that PgTS still achieves the best solution quality when allowing the baselines to run with a larger wall-clock budget.

Table 1: Direct feasible solution generation results, with the best results highlighted in bold. FR is not reported for exact solvers because they are evaluated by the first feasible solution returned.

Method	**CA**			**IS**			**SC**		
	FR ↑	PG ↓	Time ↓	FR ↑	PG ↓	Time ↓	FR ↑	PG ↓	Time ↓
Gurobi	–	28.530	**0.006**	–	39.448	**0.009**	–	96.378	**0.027**
SCIP	–	17.305	0.046	–	38.995	0.081	–	94.335	1.468
Diffusion	99.0%	36.974	0.546	49.0%	62.122	2.648	97.2%	93.977	0.889
DiffILO	73.0%	35.526	0.031	93.8%	9.042	0.032	43.4%	71.715	0.282
PgTS	**100.0%**	**7.904**	0.192	**100.0%**	**1.227**	0.214	**100.0%**	**11.611**	0.825

Table 2: Long-term performance integrated with Gurobi (left half) and SCIP (right half). The best and second-best results are highlighted in bold and italics, respectively. "Exact" means only using the backend solver (Gurobi or SCIP), without any machine learning integration. * indicates the result is statistically different from the best one under Wilcoxon signed-rank test ($p < 0.05$).

Backend solver		**Gurobi**						**SCIP**					
Problem	**Model**	**10s**		**100s**		**1000s**		**10s**		**100s**		**1000s**	
		PG ↓	PI ↓	PG ↓	PI ↓	PG ↓	PI ↓	PG ↓	PI ↓	PG ↓	PI ↓	PG ↓	PI ↓
CA	Exact	0.48	18.47*	0.11	37.67*	0.00	62.70	*4.10*	182.22*	0.22	246.28*	*0.01*	272.24*
	PaS	0.51	*17.25**	0.14*	40.26*	0.02	74.57*	5.62*	185.90*	**0.18**	250.09*	**0.00**	270.08*
	ConPaS	0.47	18.15*	0.11	*36.27**	*0.00*	*54.84*	4.82	188.70*	*0.21*	254.77*	0.01	281.12*
	Diffusion	0.50	20.61*	0.12*	40.64*	0.03*	81.40*	8.40*	166.52*	0.44*	294.44*	0.04*	370.67*
	Apollo	0.47	19.77*	*0.10*	38.27*	0.01	57.08*	**4.06**	187.82*	0.23	257.95*	0.01*	297.60*
	DiffILO	0.66*	23.36*	0.26*	56.67*	0.10*	178.42*	5.98*	*110.07**	0.35*	*217.97**	0.02*	*262.34**
	FMIP	*0.47*	17.84*	0.11*	36.93*	0.00	56.47*	4.59*	170.69*	0.24*	248.35*	0.01*	279.33*
	PgTS	**0.47**	**14.53**	**0.09**	**31.31**	**0.00**	**46.20**	4.98*	**70.46**	0.36*	**175.57**	0.01*	**217.76**
IS	Exact	0.12*	22.21*	0.00	23.22*	*0.00*	23.29*	4.30*	53.32*	0.17*	137.52*	0.16*	282.11*
	PaS	0.13*	12.11*	*0.00*	12.85*	*0.00*	12.85*	4.41*	62.76*	1.21*	217.26*	1.20*	1297.26*
	ConPaS	*0.11**	22.83*	**0.00**	23.61*	**0.00**	24.09*	4.30*	75.56*	0.17*	160.75*	0.16*	303.37*
	Diffusion	8.51*	91.52*	8.51*	856.75*	8.51*	8511.76*	12.45*	148.52*	12.34*	1259.52*	12.34*	12369.34*
	Apollo	0.14*	*4.97**	0.14*	17.26*	0.14	140.23*	**0.35**	*28.92**	**0.14**	*44.13**	**0.14**	**167.10**
	DiffILO	0.76*	19.36*	0.71*	83.98*	0.70	717.48*	1.54*	118.29*	*0.16**	141.23*	0.15*	277.41*
	FMIP	0.13	12.36	**0.00**	*13.38**	*0.00*	*13.93**	4.14*	60.53*	0.17*	149.30*	0.16*	293.91*
	PgTS	**0.01**	**1.35**	*0.00*	**1.70**	*0.00*	**2.27**	*0.90**	**12.12**	0.17*	**30.75**	*0.15*	*167.72*
SC	Exact	62.04*	906.24*	0.31	1175.54*	*0.01*	1224.68*	40.19*	607.89*	6.67	2237.30*	*2.79**	5599.87*
	PaS	27.06*	798.73*	0.29	913.27*	0.03	1008.94*	*25.27**	602.07*	6.65	*1864.57**	3.06*	5118.90*
	ConPaS	49.10*	893.58*	*0.28*	1099.34*	0.05	1184.76*	45.63*	678.65*	7.51	2601.13*	3.72*	6694.55*
	Diffusion	37.81*	846.63*	2.07*	1164.73*	1.75*	2799.22*	45.93*	679.61*	9.17	2771.11*	5.53*	8411.67*
	Apollo	88.50*	932.13*	**0.27**	1285.33*	**0.00**	1341.18*	39.51*	642.28*	6.70	2271.37*	**0.53**	**2940.98**
	DiffILO	*12.39**	*360.36**	0.29	*447.71**	0.04*	*527.60**	30.22*	*556.91**	*6.29*	2041.13*	3.10*	5356.56*
	FMIP	72.82*	853.98*	0.31	1170.06*	0.03	1241.67*	38.25*	648.38*	6.52	2200.68*	2.88*	5367.20*
	PgTS	**5.30**	**100.96**	0.30	**179.12**	0.04*	**262.88**	**11.24**	**113.64**	**6.18**	**888.11**	2.95*	*4105.74**

5.3 Long-term Performance Comparison

The preceding results show PgTS's capability in generating high-quality feasible solutions. Here we evaluate the effectiveness of these solutions in long-term performance when integrated with Gurobi and SCIP, as described in Sec. 4.4. The results of PgTS and the baselines are presented in Table 2. Our method achieves consistently strong performance in primal integral, obtaining the best results across all benchmarks and time points with Gurobi, and the best or close-to-best results in most settings with SCIP. This indicates that our method effectively improves the anytime quality of exact solvers. For primal gap, our method also achieves either the best or close-to-best performance in most settings. Overall, these results show that the proposed method is not only effective for feasible solution generation, but also achieves strong long-term performance in guiding exact solvers.

5.4 Comparison with Primal & Ordering Heuristics

Beyond comparisons with exact solvers and learning-based solution generation methods, we further evaluate the proposed PgTS against two additional groups of conventional baselines for feasible solution generation. The first group consists of primal heuristics implemented in SCIP, including rounding (SCIP-RD), diving (SCIP-DV), feasibility pump (SCIP-FP), and relaxation enforced neighborhood search (SCIP-RENS). For each variant, we disable all other primal heuristics and keep only the corresponding heuristic family active. We also include SCIP-Triv that denotes SCIP with trivial heuristics enabled, contrasting with experiments in Sec. 5.2 where trivial heuristics are disabled. The second group consists of variable ordering rules embedded in the same feasibility-guaranteed tree search mechanism. Keeping reference values the same, these baselines replace our learned variable ordering with heuristic ordering rules. MinDom selects the variable with the smallest feasible domain, while Impact selects the variable yielding the largest domain reduction.

All methods use the same stopping criterion and terminate once they find the first feasible solution. Table 3 reports the primal gap of this first feasible solution and the corresponding running time. The results show that our method largely outperforms both SCIP primal heuristics and hand-designed variable ordering rules across most settings. Although SCIP-RENS manages to reach a lower primal gap on CA, it requires a much longer running time than our method; SCIP-based heuristics generally produce first feasible solutions with large primal gaps. The ordering-heuristic baselines

Table 3: Comparison against SCIP primal heuristics and heuristic variable ordering rules for feasible solution generation. The best results are highlighted in bold.

	PG↓			Time↓		
Method	CA	IS	SC	CA	IS	SC
SCIP	17.30	38.99	94.34	0.05	0.08	1.47
SCIP-TRIV	100.00	100.00	99.91	**0.01**	**0.01**	**0.08**
SCIP-RD	61.85	99.78	81.36	0.77	0.14	1.70
SCIP-DV	62.45	99.70	81.39	1.54	14.73	2.48
SCIP-FP	25.39	36.50	94.06	0.85	0.31	56.61
SCIP-RENS	**4.24**	99.70	62.42	3.63	14.90	9.44
MINDOM	11.75	2.99	58.02	1.01	2.72	6.42
IMPACT	11.83	2.99	57.86	12.25	152.21	2328.15
PgTS	7.90	**1.23**	**11.61**	0.19	0.21	0.82

achieve strong solution quality on IS, but they still remain inferior and require a longer running time compared to our learned tree-search guidance overall. These additional comparisons provide further empirical evidence for the effectiveness of PgTS in feasible solution generation.

5.5 GENERALIZATION

We next evaluate whether PgTS generalizes to substantially larger instances without retraining. For each of CA, IS, and SC, we generate three larger variants: "-2×" and "-4×" contain twice and four times as many variables as the corresponding training instances, with proportionally increased constraint counts, while the "-12k" instances increase the number of variables to 12,000. We apply the models trained on the original-scale datasets to generate guidance pairs for these large test instances and then execute the tree search to obtain feasible solutions. We compare against DiffILO integrated with SCIP and SCIP alone, giving each baseline an instance-specific wall-clock budget equal to PgTS's end-to-end runtime on that instance. Table 4 shows that PgTS achieves the lowest reported mean PG in all nine settings. Moreover, the end-to-end inference time of PgTS grows modestly as the instance size increases, as shown in the "Time" row of Table 4. App. E.1 presents detailed tree search statistics, further showing modest growth in visited nodes and backtracks over the evaluated scales. These results indicate that PgTS can efficiently generalize beyond the training scale, allowing our method to work on substantially larger instances.

Table 4: Generalization performance on substantially larger testing instances. Row 3-5 records the average primal gap, while NA means that no feasible solution was found within the matched budget.

Task	**CA-2×**	**CA-4×**	**CA-12k**	**IS-2×**	**IS-4×**	**IS-12k**	**SC-2×**	**SC-4×**	**SC-12k**
#Variables	3000	6000	12000	3000	6000	12000	4000	8000	12000
Time	0.73	2.40	8.69	2.91	5.53	22.14	2.22	5.56	14.07
SCIP	62.58	64.39	62.95	4.23	4.43	4.19	NA	NA	NA
DiffILO	16.94	12.39	11.68	3.00	1.83	3.68	32.09	58.44	71.73
PgTS	**8.18**	**8.17**	**7.86**	**1.38**	**1.70**	**2.18**	**20.58**	**11.74**	**24.05**

6 CONCLUSION

This paper introduces PgTS, a preference-guided tree search framework for feasible solution generation in ILPs. Our approach contributes three key innovations: Joint learning of reference values and variable ordering to guide the search toward high-quality feasible solutions; training–inference alignment by evaluating each guidance pair through the same tree-search procedure used at inference; and objective-based preference learning that favors guidance pairs yielding better solutions without requiring optimal-solution labels. Comprehensive evaluations across multiple ILP benchmarks and different time limits demonstrate that our method consistently generates feasible solutions with strong primal gaps and provides effective warm-start solutions for exact ILP solvers.

Two promising directions remain for the future improvement of the proposed framework. First, the tree search mechanism is inherently sequential and therefore cannot fully exploit GPU parallelism, which affects both training and inference efficiency. Second, it is worth extending the proposed framework to more general optimization problems, such as MILPs with continuous variables and integer programs involving nonlinear expressions.

AI USE STATEMENT

In this paper, we use large language models (LLMs) to assist with identifying relevant literature and polishing the writing. We do not use AI tools in the development of the core methodology, ideas, or contributions of this work. We take full responsibility for the final content of the paper, including any text, claims, or artifacts produced with the assistance of generative AI.

REPRODUCIBILITY STATEMENT

We provide detailed information to support the reproducibility of our results. A complete description of our method is provided in Sec. 4. Components that are only briefly introduced in the main text are described in further detail in App. C, including the tree search procedure (App. C.1), the bipartite graph representation (App. C.2), and the model architecture (App. C.3). Experimental details are provided in App. D, including the datasets (App. D.1), training configurations (App. D.2), and inference settings (App. D.3). To further facilitate reproducibility, we will release our code and data on GitHub upon acceptance, enabling researchers to reproduce our experiments and results.

REFERENCES

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.

Quentin Cappart, Tias Guns, Michele Lombardi, Gilles Pesant, and Dimos Tsouros. Combining constraint programming and machine learning: From current progress to future opportunities. *Journal of Artificial Intelligence Research*, 84, 2025.

Hongyu Cheng and Amitabh Basu. Learning cut generating functions for integer programming. In *Advances in Neural Information Processing Systems*, volume 37, pp. 61455–61480, 2024.

Antonia Chmiela, Elias Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. In *Advances in Neural Information Processing Systems*, volume 34, pp. 24235–24246, 2021.

George B. Dantzig. *Linear Programming and Extensions*. Princeton Landmarks in Mathematics and Physics. 2016. ISBN 9781400884179.

Patrick Deenen, Wim Nuijten, and Alp Akcay. Scheduling a real-world photolithography area with constraint programming. *IEEE Transactions on Semiconductor Manufacturing*, 36(4):590–598, 2023.

Arnaud Deza, Elias B. Khalil, Zhenan Fan, Zirui Zhou, and Yong Zhang. Learn2Aggregate: Supervised generation of Chvatal-Gomory cuts using graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 26947–26954, 2025.

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. In *Advances in neural information processing systems*, volume 32, pp. 15554–15566, 2019.

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

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, et al. MIPLIB 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.

Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 18087–18097. Curran Associates, Inc., 2020.

Prateek Gupta, Elias Boutros Khalil, Didier Chételat, Maxime Gasse, Andrea Lodi, Yoshua Bengio, and M. Pawan Kumar. Lookback for learning to branch. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856.

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.

Robert M Haralick and Gordon L Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980.

William E Hart, Carl D Laird, Jean-Paul Watson, David L Woodruff, Gabriel A Hackebeil, Bethany L Nicholson, John D Siirola, et al. *Pyomo-optimization modeling in Python*, volume 67. Springer, 2017.

He He, Hal Daumé, and Jason Eisner. Learning to search in branch and bound algorithms. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

Christopher Hojny, Mathieu Besançon, Ksenia Bestuzheva, Sander Borst, Antonia Chmiela, João Dionísio, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, Adrian Göß, Alexander Hoen, Rolf van der Hulst, Dominik Kamp, Thorsten Koch, Kevin Kofler, Jurgen Lentz, Stephen J. Maher, Gioni Mexi, Erik Mühmer, Marc E. Pfetsch, Sebastian Pokutta, Felipe Serrano, Yuji Shinano, Mark Turner, Stefan Vigerske, Matthias Walter, Dieter Weninger, and Liding Xu. The SCIP Optimization Suite 10.0. Technical report, Optimization Online, November 2025.

Taoan Huang, Aaron M Ferber, Yuandong Tian, Bistra Dilkina, and Benoit Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In *International conference on machine learning*, pp. 13869–13890, 2023.

Taoan Huang, Aaron M Ferber, Arman Zharmagambetov, Yuandong Tian, and Bistra Dilkina. Contrastive predict-and-search for mixed integer linear programs. 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. 19757–19771. PMLR, 21–27 Jul 2024.

Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

Elias B. Khalil, Bistra Dilkina, George L. Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 659–666, 2017.

Abdel Ghani Labassi, Didier Chetelat, and Andrea Lodi. Learning to compare nodes in branch and bound with graph neural networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 32000–32010. Curran Associates, Inc., 2022.

Tae-Hoon Lee and Min-Soo Kim. RL-SPH: Learning to achieve feasible solutions for integer linear programs. In *Forty-third International Conference on Machine Learning*, 2026.

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 *The Fourteenth International Conference on Learning Representations*, 2026a.

Jiajun Li, Yixuan Li, Ran Hou, Yu Ding, Shisi Guan, Jiahui Duan, Xiongwei Han, Tao Zhong, Vincent Chau, Weiwei Wu, Zhiyuan Liu, and Wanyuan Wang. Constraint matters: Multi-modal representation for reducing mixed-integer linear programming. In *The Fourteenth International Conference on Learning Representations*, 2026b.

Sirui Li, Wenbin Ouyang, Max B. Paulus, and Cathy Wu. Learning to configure separators in branch-and-cut. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, 2023.

Yixuan Li, Can Chen, Jiajun Li, Jiahui Duan, Xiongwei Han, Tao Zhong, Vincent Chau, Weiwei Wu, and Wanyuan Wang. Fast and interpretable mixed-integer linear program solving by learning model reduction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25):27090–27098, Apr. 2025.

Haotian Ling, Zhihai Wang, and Jie Wang. Learning to stop cut generation for efficient mixed-integer linear programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38 (18):20759–20767, Mar. 2024.

Chang Liu, Zhichen Dong, Haobo Ma, Weilin Luo, Xijun Li, Bowen Pang, Jia Zeng, and Junchi Yan. L2P-MIP: Learning to presolve for mixed integer programming. In *The Twelfth International Conference on Learning Representations*, 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 *The Thirteenth International Conference on Learning Representations*, 2025.

Thomas L Magnanti and Richard T Wong. Network design and transportation planning: Models and algorithms. *Transportation science*, 18(1):1–55, 1984.

Abdelouahed Ben Mhamed, Assia Kamal Idrissi, and Amal Seghrouchni. Learning branching policies for MILPs with proximal policy optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(29):24379–24387, Mar. 2026.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks, 2021.

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 *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 17584–17600, 2022.

Laurent Perron and Frédéric Didier. CP-SAT, 2024. URL `https://developers.google.com/optimization/cp/cp_solver/`.

Lara Scavuzzo, Feng Chen, Didier Chetelat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen Aardal. Learning to branch with tree MDPs. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 18514–18526. Curran Associates, Inc., 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.

Yunzhuang Shen, Yuan Sun, Andrew Eberhard, and Xiaodong Li. Learning primal heuristics for mixed integer programs. In *2021 international joint conference on neural networks (ijcnn)*, pp. 1–8. IEEE, 2021.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9367–9376. PMLR, 13–18 Jul 2020.

Ruobing Wang, Xin Li, Yangchuan Wang, Zijian Zhang, and Mingzhong Wang. Generative branching for mixed-integer linear programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(17):14352–14360, Mar. 2026.

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 *International Conference on Learning Representations*, 2023.

Marco Wirtz, Maria Hahn, Thomas Schreiber, and Dirk Müller. Design optimization of multi-energy systems using mixed-integer linear programming: Which model complexity and level of detail is sufficient? *Energy Conversion and Management*, 240:114249, 2021. ISSN 0196-8904.

Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning large neighborhood search policy for integer programming. *Advances in Neural Information Processing Systems*, 34:30075–30087, 2021.

Jinfeng Yang, Jiliu Li, Hu Qin, E Su, and Rui Zhang. The freight multimodal transport problem with buses and drones: An integrated approach for last-mile delivery. *Transportation Research Part E: Logistics and Transportation Review*, 206:104538, 2026.

Giulia Zarpellon, Jason Jo, Andrea Lodi, and Yoshua Bengio. Parameterizing branch-and-bound search trees to learn branching policies. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3931–3939, May 2021.

Hao Zeng, Jiaqi Wang, Avirup Das, Junying He, Kunpeng Han, Haoyuan Hu, and Mingfei Sun. Effective generation of feasible solutions for integer programming via guided diffusion. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 4107–4118, 2024.

Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519: 205–217, 2023.

Sijia Zhang, Shuli Zeng, Shaoang Li, Feng Wu, and Xiangyang Li. Learning to select nodes in branch and bound with sufficient tree representation. In *The Thirteenth International Conference on Learning Representations*, 2025.

Preference-Guided Tree Search for Feasible Solution Generation in Integer Linear Programming (Appendix)

A Proof of Feasibility Guarantee

We prove that the proposed tree search can find a feasible solution whenever the ILP instance is feasible.

Proposition 1 (Feasibility guarantee)**.** *When allowed to run to completion, the proposed tree search is guaranteed to find a feasible solution to the binary ILP if one exists, regardless of the reference values or variable ordering.*

Proof. Since the proposed tree search is a complete depth-first search with a pruning mechanism, regardless of the reference values and variable ordering, we only need to show that the proposed tree search can terminate in finite time and that its pruning does not remove feasible solutions.

- **Finite termination:** Let n be the total number of binary variables in the ILP. The algorithm systematically assigns a binary value to one variable at each level of the tree. Therefore, the maximum depth of the search tree is strictly bounded by n, and the size of all state spaces is bounded by $\mathcal{O}(2^n)$. Because the search space is finite and by definition no state is revisited, the algorithm is guaranteed to terminate.
- **Safe Pruning:** As introduced in Sec. 4.1, a branch is pruned if and only if the current partial assignment of variables makes at least one constraint impossible to satisfy, i.e., makes the lower bound of the left-hand-side expression greater than the right-hand-side value. Since a feasible solution satisfies all constraints, any partial variable assignments that can be extended to such a feasible solution will pass the check and will not be pruned. Therefore, the tree search will not prune any feasible solutions.

Therefore, the proposed tree search guarantees the discovery of a feasible solution if the instance is feasible and execution is allowed to finish. □

B Illustrative Cases

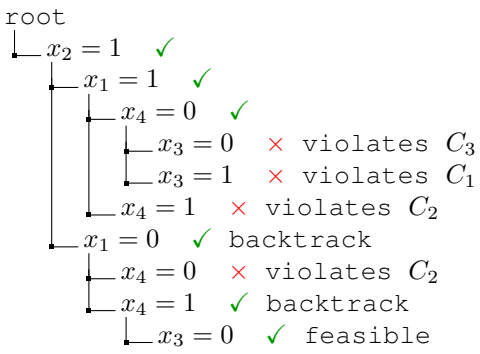
We provide illustrative examples to support the discussion in Sec. 4.1. These examples show that both components of tree search guidance, namely reference values and variable ordering, can substantially affect the quality of the feasible solution returned by tree search. The examples are based on the same toy ILP instance as in the left panel of Figure 1:

$$\begin{aligned} \min \quad & x_1 + 5x_2 + 3x_3 + 2x_4, \\ \text{s.t.} \quad & C_1 : x_2 + x_3 \leq 1, \\ & C_2 : x_1 + x_4 = 1, \\ & C_3 : x_3 + x_4 \geq 1, \\ & x_i \in \{0, 1\}, \quad i = 1, 2, 3, 4. \end{aligned}$$

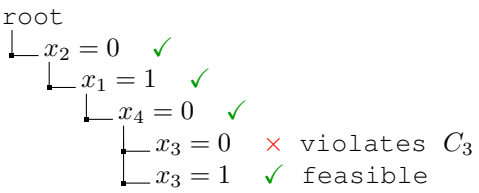
Reference values. We first examine the effect of reference values while keeping the variable ordering fixed. Consider two sets of reference values, $r = (1, 1, 0, 0)$ and $r' = (1, 0, 0, 0)$, both paired with the same variable ordering $\tau = (x_2, x_1, x_4, x_3)$ to guide the tree search. The corresponding search processes are shown in Figure 2. Compared with r, the reference values r' guide the search closer to a higher-quality feasible solution, resulting in fewer backtracking steps and a better objective value. This example demonstrates that reference values directly influence the search trajectory and the quality of the generated feasible solution. Therefore, learning effective reference values is essential for producing high-quality feasible solutions.

Variable ordering. We next examine the effect of variable ordering while keeping the reference values fixed. Using the same reference values $r = (1, 1, 0, 0)$, we compare two variable orderings, $\tau = (x_2, x_1, x_4, x_3)$ and $\tau' = (x_1, x_3, x_2, x_4)$. The corresponding search processes are shown in Figure 3. Although the reference values are identical, the two variable orderings lead to different search trajectories and yield feasible solutions with different objective values. In particular, τ' guides the tree search toward a higher-quality feasible solution. This example shows that variable ordering is also a crucial component of tree-search guidance. Hence, it should be learned together with the reference values rather than treated as a fixed or secondary design choice.

Together, these examples highlight that both reference values and variable ordering play important roles in determining the outcome of feasibility-guaranteed tree search. This motivates our objective-based preference learning scheme, which jointly learns these two components of tree-search guidance according to the objective quality of the generated feasible solutions.

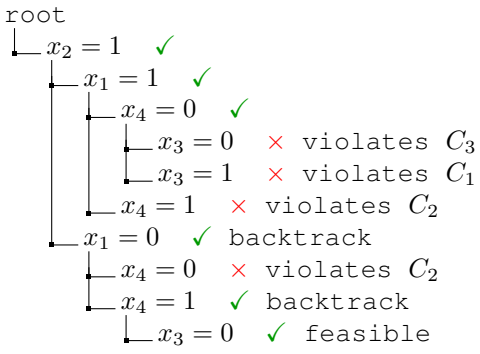


Feasible solution: $(0, 1, 0, 1)$**, obj: 7**

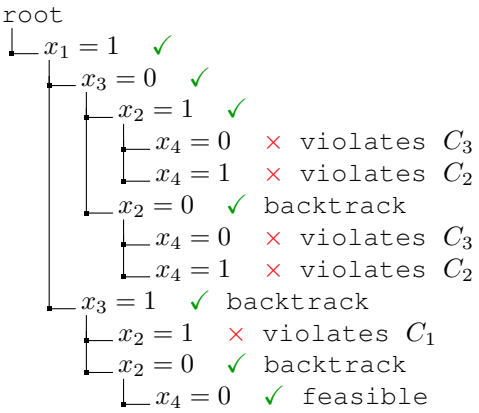


Feasible solution: $(1, 0, 1, 0)$**, obj: 4**

Figure 2: Illustration of tree search procedures guided by (r, τ) (left) and by (r', τ) (right).



Feasible solution: $(0, 1, 0, 1)$**, obj: 7**



Feasible solution: $(1, 0, 1, 0)$**, obj: 4**

Figure 3: Illustration of tree search procedures guided by (r, τ) (left) and by (r, τ') (right).

C Implementation Details

C.1 Feasibility-guaranteed Tree Search

The pseudocode of the feasibility-guaranteed tree search (introduced in Sec. 4.1) is presented in Algorithm 1. We implement the feasibility-guaranteed tree search using the probing interface in SCIP. Probing creates a temporary search state in which variable fixings can be imposed, propagation can be performed, and the state can be backtracked without modifying the original optimization model.

This interface naturally supports our feasibility-guaranteed tree search: tentative binary assignments are applied through probing, while SCIP's native propagation and infeasibility detection routines are used to prune the search. We note that this implementation choice does not restrict the proposed method to SCIP. Probing is used only as an implementation backend, and the same feasibility-guaranteed tree search can be implemented with any solver or package that provides equivalent support for temporary fixings, infeasibility detection, and backtracking, (such as HiGHS (Huangfu & Hall, 2018), OR-Tools CP-SAT (Perron & Didier, 2024), or Pyomo's APPSI interface (Hart et al., 2017)), or directly implemented from scratch.

Algorithm 1 Feasibility-Guaranteed Tree Search for ILP with Binary Variables

Require: ILP instance $\mathcal{I}$, reference values $\mathbf{r} = (r_1, \ldots, r_n)$, variable ordering $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_n)$.
Ensure: Feasible assignment π, or $\emptyset$ if no feasible solution exists.

function $\mathcal{T}(\pi, i)$
  if $i > n$ **then**
    return π
  $D_i \leftarrow (r_{\tau_i}, 1 - r_{\tau_i})$
  for $v \in D_i$ **do**
    $\hat{\pi} \leftarrow \pi \cup \{\tau_i \leftarrow v\}$
    if $\hat{\pi}$ does not make any constraint impossible to satisfy **then**
      $\pi' \leftarrow \mathcal{T}(\hat{\pi}, i + 1)$
      if $\pi' \neq \emptyset$ **then**
        return π'
  return $\emptyset$

C.2 Bipartite Graph Representation

We represent ILP instances as bipartite graphs, following DiffILO (Geng et al., 2025). The bipartite graph representation is $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$, where $\mathcal{V}$ denotes the vertices representing variables, $\mathcal{C}$ denotes the vertices for constraints, and $\mathcal{E}$ is the set of edges. An edge connects a variable vertex and a constraint vertex when the corresponding variable has a nonzero coefficient in the corresponding constraint. The graph is equipped with raw features, as described in Table 5.

Table 5: Description of variable, constraint, and edge features in our bipartite graph representation.

Tensor	Feature	Description
V	Objective	Normalized objective coefficient.
	Variable coefficient	Average variable coefficient in all constraints.
	Variable degree	Degree of the variable node in the bipartite graph representation.
	Maximum variable coefficient	Maximum variable coefficient in all constraints.
	Minimum variable coefficient	Minimum variable coefficient in all constraints.
C	Constraint coefficient	Average of all coefficients in the constraint.
	Constraint degree	Degree of constraint nodes.
	Bias	Normalized right-hand-side of the constraint.
E	Coefficient	Constraint coefficient.

C.3 Model Architecture

As introduced in Sec. 4, we use a graph neural network (GNN) with two output heads to learn tree search guidance. Specifically, given a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ equipped with the raw features $(\mathbf{V}, \mathbf{C}, \mathbf{E})$, initial embeddings are obtained by MLPs:

$$\boldsymbol{h}_{v_i}^{(0)} = \mathrm{MLP}_\theta(\boldsymbol{v}_i), \quad \boldsymbol{h}_{c_j}^{(0)} = \mathrm{MLP}_\theta(\boldsymbol{c}_j), \quad \boldsymbol{h}_{e_{ij}} = \mathrm{MLP}_\theta(\boldsymbol{e}_{ij}). \tag{6}$$

After that, K graph convolution layers are performed, with each layer in the form of two interleaved half-convolutions (Gasse et al., 2019), defined as follows:

$$\boldsymbol{h}_{c_j}^{(k+1)} \leftarrow \text{MLP}_\theta \left(\boldsymbol{h}_{c_j}^{(k)}, \sum_{i:e_{ij}\in\mathcal{E}} \text{MLP}_\phi \left(\boldsymbol{h}_{c_j}^{(k)}, \boldsymbol{h}_{e_{ij}}, \boldsymbol{h}_{v_i}^{(k)} \right) \right),$$
$$\boldsymbol{h}_{v_i}^{(k+1)} \leftarrow \text{MLP}_\phi \left(\boldsymbol{h}_{v_i}^{(k)}, \sum_{j:e_{ij}\in\mathcal{E}} \text{MLP}_\phi \left(\boldsymbol{h}_{c_j}^{(k+1)}, \boldsymbol{h}_{e_{ij}}, \boldsymbol{h}_{v_i}^{(k)} \right) \right). \tag{7}$$

Each convolution layer is followed by two GraphNorm layers, one for variables and the other for constraints. We employ a concatenation Jumping Knowledge layer to aggregate information from all K layers and obtain the final node representations:

$$\boldsymbol{h}_{v_i} = \text{MLP}_\theta \left(\underset{k=0,\dots,K}{\text{CONCAT}} \left(\boldsymbol{h}_{v_i}^{(k)} \right) \right), \quad \boldsymbol{h}_{c_j} = \text{MLP}_\theta \left(\underset{k=0,\dots,K}{\text{CONCAT}} \left(\boldsymbol{h}_{c_j}^{(k)} \right) \right). \tag{8}$$

We then use MLPs activated by ReLU on the final variable embeddings for both the output head for reference value prediction and the output head for variable ordering score:

$$z_{r_i} = \text{MLP}_{\theta_r} \left(\boldsymbol{h}_{v_i} \right), \quad z_{\tau_i} = \text{MLP}_{\theta_\tau} \left(\boldsymbol{h}_{v_i} \right). \tag{9}$$

Then the reference values r_i are sampled from parameterized Bernoulli distributions with $\text{sigmoid}(z_{r_i})$ as the parameter, and the variable ordering τ_i is sampled from sequential softmax sampling without replacement based on z_{τ_i}.

D Experimental Details

D.1 Dataset Details

Information about the CA, IS, and SC benchmarks used in our experiments is provided in Table 6. We use the testing and validation datasets provided by (Geng et al., 2025) for all methods. For our method, we generate random instances on the fly during training, using the instance generator provided by (Gasse et al., 2019). For other learning-based methods, we also use the training dataset provided by (Geng et al., 2025).

Table 6: Statistics of the benchmarks.

Dataset	Number of Constraints	Number of Variables
CA	576	1500
IS	5943	1500
SC	3000	2000

D.2 Training Details

During training, we sample 64 candidate guidance pairs for each instance and use a sampled subset of size $k = 8$ to compute the preference loss of Eq. 3. We train the model with the Adam optimizer. The learning rate is set to 10^{-5} for the graph convolution layers and the reference-value output head, and to 10^{-4} for the ordering output head. We use a smaller learning rate for the former components because they are initialized from pretrained models and only require moderate adaptation during training. A weight decay of 10^{-4} is applied to the whole model. We set the batch size to 64 and use 800 training instances per epoch. Training is terminated once the 10-hour time limit is reached. For baselines, we use the trained models in their official code if provided, otherwise we train their models using their official code by ourselves. All models (both ours and the baselines') are trained on a supercomputer node equipped with an NVIDIA H100 GPU, a 16-core AMD 4th Gen EPYC CPU, and 120 GB of memory.

D.3 Inference Details

During inference, model prediction is performed on the same GPU node used for training, while feasibility-guaranteed tree search and downstream exact solving are performed on a separate CPU node. This setup reflects the computational requirements of the two stages: neural prediction benefits from GPU acceleration, whereas tree search and exact MILP solving are CPU-based. This setup also reduces computational expenses, since GPU nodes consume substantially more billing units on our computing platform. The CPU node is equipped with the same AMD 4th Gen EPYC CPU with 16 CPU cores and 28 GB of memory. We assign one CPU core for one instance and one method, and use parallel computing to concurrently solve 16 instances with the 16 CPU cores. We also apply the same protocol to all baselines for fairness.

For long-term performance in guiding exact solvers, we set $\Delta = 200$ in Eq. 4 for our method, Diffusion, and DiffILO. For the other learning-based baselines, we follow their original downstream solving procedures. We use the hyperparameters reported in the original papers or official implementations whenever available; otherwise, we tune the hyperparameters by ourselves. The version of the backend solvers for long-term performance evaluation is 12.0.0 for Gurobi and 9.2.0 for SCIP.

E Additional Experiments

E.1 Statistics of Tree Search Behavior on Test Instances

In addition to the primal gap and inference time reported in Sec. 5.5, we report visited nodes and backtracks of the tree search during inference. As shown in Table 7, as the instance size grows, the number of visited nodes grows approximately linearly, while backtracking remains modest relative to the total number of visited nodes. We observe no exponential growth in search-tree size on these test instances. Although this does not eliminate the exponential worst-case bound or rule out adversarial instances, the observed trends and scalable training signal support PgTS as a promising approach for keeping the search efficient.

Table 7: Tree-search statistics at different scales, reported as mean $\pm$ standard deviation / maximum.

	Variables	Constraints	Visited nodes	Backtracks
CA	1500	576	3168.7 $\pm$ 18.5 / 3214	302.2 $\pm$ 13.7 / 332
CA	3000	1152	6152.7 $\pm$ 19.5 / 6213	434.2 $\pm$ 14.8 / 475
CA	6000	2304	12318.6 $\pm$ 39.1 / 12416	873.1 $\pm$ 25.9 / 932
CA	12000	4608	24600.2 $\pm$ 57.3 / 24663	1732.0 $\pm$ 36.3 / 1772
IS	1500	5943	2476.1 $\pm$ 22.6 / 2530	229.4 $\pm$ 14.8 / 259
IS	3000	11886	4947.4 $\pm$ 33.5 / 5041	450.3 $\pm$ 21.5 / 509
IS	6000	23772	9867.5 $\pm$ 37.5 / 9938	883.6 $\pm$ 27.3 / 923
IS	12000	47544	19697.6 $\pm$ 45.4 / 19784	1730.4 $\pm$ 37.4 / 1791
SC	2000	3000	2092.8 $\pm$ 3.0 / 2100	46.3 $\pm$ 1.5 / 50
SC	4000	6000	4189.5 $\pm$ 4.4 / 4202	94.7 $\pm$ 2.3 / 101
SC	8000	12000	8209.8 $\pm$ 4.7 / 8220	103.3 $\pm$ 2.9 / 109
SC	12000	18000	12227.3 $\pm$ 3.9 / 12236	113.5 $\pm$ 2.0 / 118

E.2 Ablation Studies

Jointly Learning Reference Values and Variable Ordering We first study the contribution of the two output heads used to learn the tree-search guidance. To isolate the role of the value head, we design PgTS_Order, which only learns variable ordering and uses DiffILO's reference-value predictions as reference values, since DiffILO is among the strongest solution prediction models. To isolate the role of the ordering head, we design PgTS_Value, which only learns reference values and uses MinDom (Haralick & Elliott, 1980), a simple yet effective ordering heuristic, as the variable ordering. All other components, as well as the training and testing protocols, are kept the same as in our full model. The two ablation models and our full model are evaluated on direct feasible solution

Table 8: Ablation of the two output heads on the CA benchmark. The best results are highlighted in bold.

Method	PG	Time
PgTS	**7.90**	**0.19**
PgTS_Order	15.36	0.56
PgTS_Value	11.40	0.74

generation on the CA benchmark, where we report the PG of the generated feasible solutions and the inference time. The results are reported in Table 8. Both single-head variants lead to clear performance degradation compared with our full model, indicating that jointly learning both reference values and variable ordering provides complementary benefits.

Parameter Initialization We next examine the effect of parameter initialization. As introduced in Sec. 4.3, our default model initializes the graph convolution layers and the value head from a trained DiffILO model. We compare it with two variants: PgTS_PaS, which uses the trained model from PaS for initialization, and PgTS_NoIni, which uses random initialization rather than pretrained parameters. This ablation is evaluated on the CA benchmark using two metrics: the primal gap of the direct feasible solution generation, denoted by "Inference PG", and the primal gap of guiding Gurobi as the downstream solver for 10 seconds, denoted by "PG at 10s".

Table 9 shows that replacing parameter initialization affects the proposed framework, but it remains effective without this mechanism and brings improvements to the corresponding pretrained models. These results suggest that parameters from trained models provide a useful initialization, but the objective-based preference learning procedure itself is a key source of performance. While PgTS_PaS yields the weakest performance of the three PgTS methods, it still achieves a better 10-second primal gap than PaS. Taken together with the improvements PgTS shows over DiffILO, these findings highlight our method's ability to boost existing solution prediction models. Furthermore, this suggests that integrating more advanced prediction models in the future could yield even better initialization for our approach.

Table 9: Ablation of parameter initialization on the CA benchmark. The best results are highlighted in bold. "–" is because PaS is not designed as a standalone feasible solution generation method.

Method	Inference PG	PG at 10s
PgTS	**7.90**	**0.47**
PgTS_NoIni	17.99	0.47
DiffILO	35.53	0.66
PgTS_PaS	29.54	0.48
PaS	–	0.51

Hamming Neighborhood To distinguish the contribution of PgTS' generated feasible solution from that of the Hamming-neighborhood restriction (Eq. 4), we further conduct an ablation study with Gurobi. We added PgTS-WS, which warm-starts the original unrestricted ILP with PgTS's generated solution; DiffILO-WS, the analogous DiffILO variant; and LP-HN, which applies the same Hamming neighborhood around rounded LP solutions. The results are shown in Table 10. Among the unrestricted methods, PgTS-WS has the lowest PI in all settings and no larger PG than Exact in most settings. These results show that PgTS's generated feasible solutions improve solver performance even without the Hamming-neighborhood restriction, while the restriction provides additional gains in some settings.

E.3 Sensitivity Analysis of Hyperparameter

When using our method to guide exact solvers, we introduce a hyperparameter Δ to control the size of the search region. We study the sensitivity of our method to different values of Δ. Table 11 reports the results on the SC dataset. Overall, the performance is relatively stable across different Δ values.

Table 10: Ablation of PgTS' generated feasible solution and the Hamming neighborhood restriction.

	Model	10s		100s		1000s	
		PG	PI	PG	PI	PG	PI
CA	Exact	0.48	18.47	0.11	37.67	0.00	62.70
	LP-HN	0.47	32.86	**0.08**	49.00	0.00	58.70
	DiffILO-WS	0.49	15.83	0.11	34.91	0.00	53.01
	DiffILO	0.66	23.36	0.26	56.67	0.10	178.42
	PgTS-WS	**0.46**	**13.97**	0.11	32.83	0.00	51.27
	PgTS	0.47	14.53	0.09	**31.31**	**0.00**	**46.20**
IS	Exact	0.12	22.21	0.00	23.22	**0.00**	23.29
	LP-HN	100.00	1000.00	100.00	10000.00	100.00	100000.00
	DiffILO-WS	0.15	8.47	**0.00**	9.71	**0.00**	9.71
	DiffILO	0.76	19.36	0.71	83.98	0.70	717.48
	PgTS-WS	0.13	6.06	0.00	6.95	**0.00**	7.41
	PgTS	**0.01**	**1.35**	0.00	**1.70**	0.00	**2.27**
SC	Exact	62.04	906.24	0.31	1175.54	**0.01**	1224.68
	LP-HN	81.17	944.49	0.36	1216.77	0.02	1300.44
	DiffILO-WS	13.30	565.33	0.30	654.50	0.03	722.63
	DiffILO	12.39	360.36	0.29	447.71	0.04	527.60
	PgTS-WS	**5.24**	102.03	**0.27**	**176.32**	0.03	**249.81**
	PgTS	5.30	**100.96**	0.30	179.12	0.04	262.88

We further observe that larger Δ values tend to yield worse Gap and PI within a short time budget of 10 seconds, but can achieve better performance under a longer time budget of 1000 seconds. This behavior is expected. Since the generated feasible solutions are already of high quality, a smaller search region induced by a smaller Δ allows the solver to focus on local improvement around a strong warm-start solution, which is beneficial under limited time budgets. In contrast, a larger search region induced by a larger Δ provides more flexibility for exploring alternative feasible solutions and can lead to better solutions when sufficient solving time is available. These results suggest a practical guideline for using our method: smaller Δ values are preferable under short time budgets, whereas larger Δ values can be beneficial when the solver is given enough time.

Table 11: Performance of different Δ values when using our method to guide Gurobi on the SC dataset.

Δ	10s		100s		1000s	
	PG ↓	PI ↓	PG ↓	PI ↓	PG ↓	PI ↓
200	5.30	100.96	0.30	179.12	0.04	262.88
400	7.10	110.50	0.28	194.51	0.04	266.47
600	7.09	109.70	0.31	195.08	0.02	270.74
800	7.45	109.36	0.29	194.44	0.03	269.36
1000	7.72	109.89	0.28	196.56	0.02	260.35

E.4 Feasible Solution Comparison Under Comparable Wall-Clock Time

In Sec. 5.2, we have shown that PgTS significantly outperforms the baselines in the quality of their generated solutions, although taking more time in several settings. To examine whether this quality advantage remains when the baselines receive additional solving time, we further compare PgTS with Gurobi, SCIP, Diffusion, and DiffILO under short end-to-end wall-clock budgets. The baselines are given 0.2 seconds on CA, 0.3 seconds on IS, and 1 second on SC, following the same solver-integration procedures as in the long-term comparison setting in Sec. 5.1. These budgets are slightly larger than PgTS's direct-generation running time on the corresponding benchmarks in Sec. 5.2. Table 12 shows that PgTS retains the lowest reported PG on all three benchmarks, achieving 7.90, 1.23, and 11.61 on CA, IS, and SC, respectively, compared with 11.89, 2.14, and 44.12 for the

strongest baseline, DiffILO. These results further support PgTS's effectiveness in generating high-quality feasible solutions within short running times.

Table 12: PG under a slightly larger wall-clock budget than PgTS's direct feasible solution generation. NA means that no feasible solution was found.

	CA	**IS**	**SC**
Gurobi	13.97	39.17	96.38
SCIP	17.10	3.41	95.35
Diffusion	NA	NA	93.98
DiffILO	11.89	2.14	44.12
PgTS	**7.90**	**1.23**	**11.61**

E.5 MIPLIB BENCHMARKING

To evaluate the effectiveness of our method on realistic ILP instances, we conduct two additional experiments on selected instances from the MIPLIB 2017 collection (Gleixner et al., 2021). The first evaluates application within the CVS family, while the second considers a small heterogeneous subset containing instances from different problem families. Following the prior learning-based ILP study (Geng et al., 2025), we select instances from the same CVS family to form a meaningful train/test split. The subset contains three training instances, namely cvs08r139-94, cvs16r70-62, and cvs16r89-60, and two testing instances, namely cvs16r106-72 and cvs16r128-89.

Table 13 reports the long-term performance of Gurobi when guided by each method. We compare against default Gurobi and three strong baselines selected from Table 2: PaS, which performed well within the first 100 seconds; Apollo, which achieved strong final performance; and DiffILO, the strongest solution generation baseline that gained consistently good performance across the search horizon. Strikingly, PgTS achieves lower Gap and PI across all three time limits, demonstrating that it remains effective on complex realistic instances. In particular, within 1000 seconds, Gurobi guided by PgTS achieves a negligible optimality gap and reaches optimality on most runs.

Table 13: Long-term performance with Gurobi on the CVS instances selected from the MIPLIB 2017 collection.

Method	**10s**		**100s**		**1000s**	
	PG ↓	PI ↓	PG ↓	PI ↓	PG ↓	PI ↓
Gurobi	10.20	287.42	1.65	539.37	1.13	1586.57
PaS	12.28	318.06	0.68	583.26	0.12	730.83
Apollo	9.89	228.16	1.65	452.56	1.65	1935.95
DiffILO	7.25	263.73	0.93	432.53	0.72	1101.07
PgTS	**6.68**	**131.91**	**0.47**	**293.70**	**0.00**	**476.30**

The second experiment evaluates a heterogeneous subset of instances. We follow the experimental setting of Apollo (Liu et al., 2025) and select eleven binary ILP instances, using eight for training and three for testing. Table 14 reports the long-term performance of Gurobi, DiffILO, and PgTS at 10, 100, and 1000 seconds. It shows that PgTS achieves better performance than Gurobi and outperforms DiffILO at 100s and 1000s, providing additional evidence for PgTS's generalization across heterogeneous realistic instances.

Table 14: Long-term performance with Gurobi on the selected heterogeneous instances from the MIPLIB 2017 collection.

	PG@10s	**PI@10s**	**PG@100s**	**PI@100s**	**PG@1000s**	**PI@1000s**
Gurobi	59.73	718.37	16.61	2991.26	11.32	13605.50
DiffILO	**50.07**	**692.05**	17.25	2867.64	11.27	14381.11
PgTS	50.70	707.79	**13.32**	**2603.58**	**10.46**	**12865.99**