MILP-FBGen: LP/MILP Instance Generation with Feasibility/Boundedness

Yahong Zhang [*1] **Chenchen Fan** [*1] **Donghui Chen** [1] **Congrui Li** [1] **Wenli Ouyang** [1] **Mingda Zhu** [1] **Junchi Yan** [2]

Abstract

Machine learning (ML) has been actively adopted in Linear Programming (LP) and Mixed-Integer Linear Programming (MILP), whose potential is hindered by instance scarcity. Current synthetic instance generation methods often fall short in closely mirroring the distribution of original datasets or ensuring the feasibility and boundedness of the generated data — a critical requirement for obtaining reliable supervised labels in model training. In this paper, we present a diffusion-based LP/MILP instance generative framework called MILP-FBGen. It strikes a balance between structural similarity and novelty while maintaining feasibility/boundedness via a meticulously designed structure-preserving generation module and a feasibility/boundedness-constrained sampling module. Our method shows superiority on two fronts: 1) preservation of key properties (hardness, feasibility, and boundedness) of LP/MILP instances, and 2) enhanced performance on downstream tasks. Extensive studies show two-fold superiority that our method ensures higher distributional similarity and 100% feasibility in both easy and hard datasets, surpassing current state-of-the-art techniques.

1. Introduction

LP (Linear Programming) and MILP (Mixed Integer Linear Programming) have wide applications, e.g. planning logistics path problems (Troncoso & Garrido, 2005), tackling matching problems (Wang et al., 2020), and scheduling orders in production workshop (Tang et al., 2001). Although the instances from the same applications share similar patterns and characteristics, classic heuristic methods (Whitley, 1994; Helsgaun, 2000) or branch-and-bound solvers (Gurobi; SCIP; HiGHS) solve them repeatedly without making use of those similarities. Machine learning (ML) methods (Sun & Yang, 2023; Fan et al., 2023; Han et al., 2023; Nair et al., 2020) hold significant potential in recognizing patterns, and hence they are helpful for building optimization algorithms. Nevertheless, the application of ML faces various challenges stemming from the intricacies and data scarcity inherent in real-world problems from general combinatorial problems (Bengio et al., 2021; Yehuda et al., 2020), satisfiabilty (Guo et al., 2023; Malitsky et al., 2016a), to MILP (Zhang et al., 2023). The lack of practically usable and representative instances hinders the training and generalization of ML methods. To address this issue, efforts have been dedicated to LP/MILP instance generation.

The current mainstream techniques for LP/MILP instance generation can be divided into three categories: i) building on (practically limited) domain knowledge (Vander Wiel & Sahinidis, 1995; Balas & Ho, 1980); ii) constructing general LP/MILP instances by sampling from the hyperparameter encoding space (Bowly et al., 2020; Li et al., 2024; Bowly, 2019); iii) using a deep generative framework to destroy and repair parts of the original LP/MILP instances (Geng et al., 2023; Wang et al., 2023). The first category involves a specific scheme or formulation design, and can only cover a few LP/MILP instance types with known expert knowledge. The second category is for general instance generation, but only controlling and manipulating limited statistics make it difficult to capture the fine-grained characteristics, such as keeping the topological structure of a single instance. To address this issue, the third one converts the LP/MILP instance generation into a graph generation problem. It iteratively destroys and repairs parts of the original graphs, typically with a masked variational autoencoder (VAE) model (Kipf & Welling, 2016), so that the original instance's topological structure can be preserved to some extent. G2MILP (Wang et al., 2023) generates instances that can simulate real-world scenarios and enrich the original dataset, but it does not guarantee the feasibility and boundedness for many real-world problems. If the generated instance is infeasible and unbounded, it becomes impossible to obtain corresponding label data for supervised learning, which hinders the application of most ML methods. DIG-MILP (Wang et al., 2023) claims to be able to guarantee these two points but at the

[*]Equal contribution [1]AI Lab, Lenovo Research, Beijing, China [2]School of Artificial Intelligence & Department of Computer Science and Engineering & MoE Lab of AI, Shanghai Jiao Tong University, Shanghai, China. Correspondence to: Wenli Ouyang <ouyangwl1@lenovo.com>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

cost of sacrificing the structure-preserving capacity. Due to the complete modification of all constraints, its generated instances have a large gap with the original training data concerning both the structural similarity and computational hardness, which fall out of the training distribution and can be useless in the original training task.

Therefore, ensuring these key properties of the generated instances is a fundamental step in LP/MILP sample generation and our work focuses on addressing these issues. In this paper, we resort to the probabilistic diffusion models, to build a diffusion-based LP/**MILP** instance generative framework called **MILP-FBGen** which meanwhile preserves **F**easibility and **B**oundedness. Using the standard MILP formulation $\mathbf{MILP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$ (see Eq. 1 in the method section), we first devise a structure-preserving generation module to generate new row or column vectors of the incidence matrix $\mathbf{A}$ and replace the existing ones, so that the balance between the novelty and structure's preservation can be controlled by setting the replacement ratio. Besides, we develop a feasibility and boundedness-constrained sampling module that makes right-hand side constant $\mathbf{b}$ and objective coefficient $\mathbf{c}$ fall within the appropriate domain. Thus, the feasibility and boundedness of the generated instances can be ensured and superfluous constraints from both the primal and dual forms are also avoided. Furthermore, we design different constraint/variable vector selection strategies to guide the specification of vectors to be regenerated in incidence matrix $\mathbf{A}$: random selection and task-oriented selection. **The contributions are:**

1) To our best knowledge, we have pioneered the development of the first diffusion-based generative framework for LP/MILP instance generation namely MILP-FBGen, in contrast to those based on VAE framework (Geng et al., 2023) for MILP or more broadly SAT instance generation (Malitsky et al., 2016b; Li et al., 2023a; Chen et al., 2024).

2) We develop a tailored approach to preserve both feasibility and boundedness while ensuring a high structural similarity between the generated and original instances, which has not been ensured in peer works to the best of our knowledge.

3) We design different constraint/variable vector selection strategies for LP/MILP instance generation, suitable for different scenarios.

4) Extensive experiments verify the higher similarity and 100% feasibility in diverse datasets, compared with current state-of-the-art methods. Our generated instances also significantly boost the performance of representative downstream tasks. Source code will be made publicly available.

2. Related Work

Machine learning for LP/MILP. ML techniques have emerged as a powerful approach in operations research (Bengio et al., 2021; Cappart et al., 2023). The mainstream methods can be divided into two classes. The first one directly solves specific LP/MILP problems (Han et al., 2023; Ye et al., 2023; Sun & Yang, 2023; Li et al., 2023b), e.g. by one-shot forward prediction. The second one assists the solving process by embedding ML models within the solver to replace its key components, including node selection (He et al., 2014; Labassi et al., 2022), branching (Zarpellon et al., 2021; Gupta et al., 2020), initial basis prediction (Fan et al., 2023) and so on. Although ML methods exhibit promise, they are still limited by inadequate data, especially in the supervised setting. Unfortunately, existing instance generation methods often fall short of ensuring feasibility and boundedness, resulting in instances that are practically unusable. Our generative framework can complement the original instances by generating both feasible and bounded instances, which are vital in many real-world applications.

LP/MILP instance generation. Early approaches use expert knowledge or mathematical formulas to generate instances for a specific problem type, e.g. Set Covering (SC) (Balas & Ho, 1980), Travelling Salesman Problem (TSP) (Pilcher & Rardin, 1992; Vander Wiel & Sahinidis, 1995), and Knapsack Problem (Hill et al., 2011). To generate general LP/MILP instances, (Bowly et al., 2020; Bowly, 2019) achieve it by sampling from the hyperparameter encoding space. However, it cannot control the structural characteristics while preserving the same distribution as the original data. Based on the deep generative framework, G2MILP (Geng et al., 2023) proposes a VAE-based generative model and iteratively destroys and repairs parts of the original graphs to generate new ones. Its generated instances closely resemble the original data, but they may not satisfy the feasibility and boundedness, which is quite essential for most ML tasks to obtain supervised labels. While DIG-MILP (Wang et al., 2023) asserts the assurance of these two fundamental properties, the instances it generates deviate from the original training distribution. This deviation could result in poorer performance in the downstream task when incorporating these newly generated instances. Therefore, we are the first to address the aforementioned issues existing in previous works simultaneously. Specifically, our approach allows for the direct generation of instances that are both feasible and bounded, while also ensuring their distribution closely resembles that of the original data.

3. Preliminaries

Given a triplet of the incidence matrix $\mathbf{A} \in \mathbb{Q}^{m \times n}$, the right-hand side constant $\mathbf{b} \in \mathbb{Q}^m$, and the objective coefficient $\mathbf{c} \in \mathbb{Q}^n$, a $\mathbf{MILP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$ can be defined as:

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}^\mathsf{T}\mathbf{x}, \\ &\text{s.t.} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \in \mathbb{Z}^p_{\geq 0} \times \mathbb{Q}^{n-p}_{\geq 0}, \end{aligned} \tag{1}$$

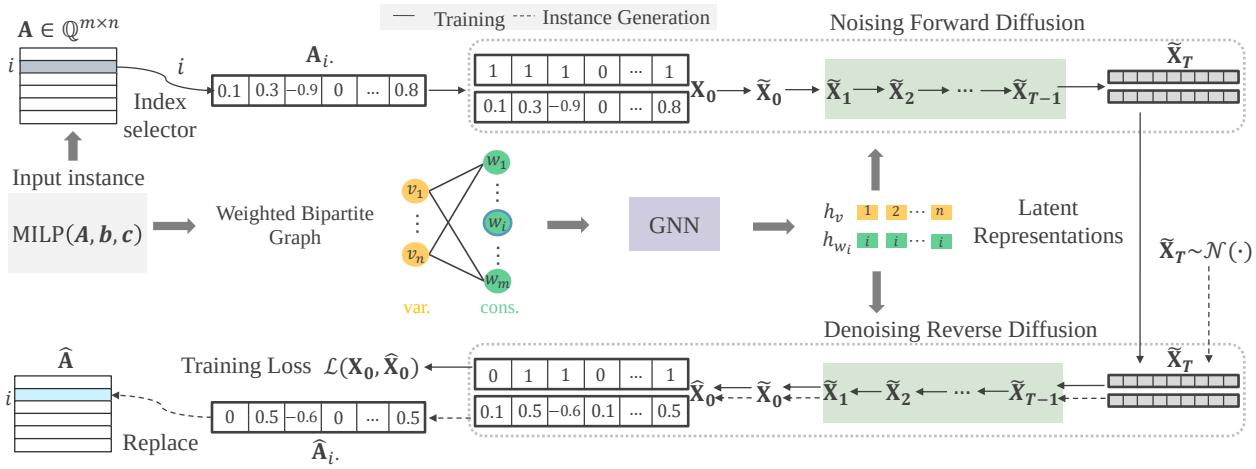


Figure 1. SPGM module overview: An LP/MILP instance is represented as a bipartite graph, where yellow and green nodes signify variables and constraints, respectively. A destroy and repair scheme is applied to a constraint vector $\mathbf{A}_{i\cdot}$ and the two rows of $\mathbf{X}_0$ indicate: i) its connections to variables; ii) connection weights. These two rows undergo the forward noising and reverse denoising processes. The multiplication of these rows yields the new constraint $\hat{\mathbf{A}}_{i\cdot}$. The diffusion model is trained by reconstructing it back to the original one.

where $\mathbf{x} = (x_1, x_2, \cdots, x_n)^\mathsf{T}$ is the vector of decision variables with p variables being integer. Each row vector of incidence matrix $\mathbf{A}$ represents the connection between each constraint and all other decision variables, while each column vector corresponds to the relationship between each decision variable and all other constraints. For clarity, we refer to the row vector and column vector of incidence matrix $\mathbf{A}$ as the constraint vector and variable vector respectively. $\mathbf{LP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$ can be seen as a special form of the MILP by setting $p = 0$, and thus all decision variables take real values. Our method applies to both MILP and LP instances.

As aforementioned, for most ML-based tasks, e.g. node selection (He et al., 2014) and initial basis prediction (Fan et al., 2023), the MILP instances need to satisfy the conditions of feasibility and boundedness and have optimal solutions. The definitions of feasibility, boundedness, and optimal solution of MILP are as follows:

Definition 3.1. (Feasibility of MILP): A $\mathbf{MILP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$ is feasible if there exists an $\mathbf{x_f}$ such that all the constraints are satisfied. Then, $\mathbf{x_f}$ is termed as a feasible solution.

Definition 3.2. (Boundedness of MILP): An instance $\mathbf{MILP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$ is bounded if there is an upper bound on $\mathbf{c}^\mathsf{T}\mathbf{x}$ across all the feasible solutions.

Definition 3.3. (Optimal solution of MILP): $\mathbf{x}^*$ is recognized as an optimal solution if it is a feasible solution and no worse than all other feasible solutions $\mathbf{x}$: $\mathbf{c}^\mathsf{T}\mathbf{x}^* \leq \mathbf{c}^\mathsf{T}\mathbf{x}$.

By narrowing the ranges of $\mathbf{A}$, $\mathbf{b}$ and $\mathbf{c}$ from the real domain $\mathbb{R}$ to the rational domain $\mathbb{Q}$, as commonly performed in MILP studies, any feasible-bounded MILP exists an optimal solution (Schrijver, 1998). Hence, our generative model only needs to ensure that the generated instances are feasible and bounded.

4. Methodology

We present MILP-FBGen, featuring three unique modules: structure-preserving generation, feasibility/boundedness-constrained sampling, and constraint/variable vector selection. We also elaborate on our model architecture in this section.

4.1. Structure-preserving Generation

Our primary objective is to preserve the intrinsic structures of the original LP/MILP instances as much as possible. The incidence matrix $\mathbf{A}$ encapsulates crucial structural information, yet generating the entire matrix directly can be computationally expensive and challenging, as noted in (Li et al., 2023a). Fortunately, leveraging prior knowledge, we adopt a destroy and repair paradigm, aiming at partial segments of the matrix to ensure minimal structural changes (Geng et al., 2023). We implement this concept by selecting one or more constraint vectors from the incidence matrix $\mathbf{A}$. Subsequently, we generate one or more new vectors to replace the existing ones based on the diffusion model. As the generation processes for one or more constraint and variable vectors are equivalent, for the sake of simplicity, we choose to exemplify the generation of one constraint vector to illustrate the structure-preserving generation module (SPGM), as depicted in Fig. 1.

Formally, given an incidence matrix $\mathbf{A}$ drawn from the dataset D, we select a constraint vector from it based on a specific selection strategy (details in Sec. 4.3), denoted by $\mathbf{A}_{i\cdot} \sim p(\mathbf{A}_{i\cdot}|\mathbf{A})$. To better represent and reconstruct $\mathbf{A}_{i\cdot} \in \mathbb{R}^{1\times n}$, here we re-formulate this selected constraint vector $\mathbf{A}_{i\cdot}$ as a matrix $\mathbf{X} = \begin{bmatrix} \mathbf{I}\{\mathbf{A}_{i\cdot} \neq 0\} \\ \mathbf{A}_{i\cdot} \end{bmatrix} \in \mathbb{R}^{2\times n}$. The first row indicates whether the i_{th} constraint is connected with each decision variable. $\mathbf{X}^0_j = 1$ $(j = 0, 1, \cdots, n-1)$

represents a connection between it and the j_{th} variable, while $\mathbf{X}_j^0 = 0$ represents no connection. The second row indicates the weights between the connected constraint and decision variable. By converting $\mathbf{A}_{i\cdot}$ to $\mathbf{X}$, we achieve task decomposition and can directly obtain the reconstructed $\mathbf{A}_{i\cdot}$ by multiplying them: $\hat{\mathbf{A}}_{i\cdot} = \mathbf{X}^0 \times \mathbf{X}^1$. Thus, we aim to build a parameterized generator $p_\theta\left(\mathbf{X}|\mathbf{A}, \mathbf{b}, \mathbf{c}, i\right)$. We train the generator by maximizing the log-likelihood $\log p_\theta\left(\mathbf{X} = \hat{\mathbf{X}}|\mathbf{A}, \mathbf{b}, \mathbf{c}, i\right)$ of reconstructing $\mathbf{X}$ given input $\mathbf{MILP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$ and constraint index i. The objective is:

$$\arg\max_\theta \mathbb{E}_{(\mathbf{A},\mathbf{b},\mathbf{c})\sim\mathbf{D}} \mathbb{E}_{\mathbf{X}\sim p(\mathbf{X}|\mathbf{A})} \log p_\theta\left(\mathbf{X}|\mathbf{A}, \mathbf{b}, \mathbf{c}, i\right). \tag{2}$$

Diffusion paradigm for incidence matrix modeling. We follow the probabilistic diffusion model (Sohl-Dickstein et al., 2015; Ho et al., 2020) to parameterize the generative distribution p_θ in Eq. 2. For brevity, we omit the conditional notations of $(\mathbf{A}, \mathbf{b}, \mathbf{c}, i)$ and denote $\mathbf{X}$ as $\mathbf{X}_0$ representing the raw sample from dataset $\mathbf{D}$. As shown in Fig. 1, we follow the processing approach in (Chen et al., 2022) that first rescales the $\{0, 1\}$-valued vector $\mathbf{X}^0$ to the $\{-1, 1\}$ domain as $\tilde{\mathbf{X}}^0$, and then treats it as real values. Thus, the matrix $\tilde{\mathbf{X}} = \begin{bmatrix}\tilde{\mathbf{X}}^0 \\ \mathbf{X}^1\end{bmatrix}$ is all consisted of real values and can be applied to a continuous-space diffusion model. $\tilde{\mathbf{X}}_1, \tilde{\mathbf{X}}_2, \cdots, \tilde{\mathbf{X}}_T$ are the latent variables with the same dimension. In the predefined forward noising process:

$$q(\tilde{\mathbf{X}}_{t+1}|\tilde{\mathbf{X}}_t) := \mathcal{N}(\tilde{\mathbf{X}}_{t+1}; \sqrt{\alpha_t}\tilde{\mathbf{X}}_t, (1-\alpha_t)\mathbf{I}), \tag{3}$$

where $1 - \alpha_t$ denotes the variance schedule. With carefully chosen and long enough iteration steps T, we want $\prod_{t=1}^T \alpha_t \approx 0$ such that $\mathbf{X}_T \sim \mathcal{N}(\cdot)$. Based on Bayes' Theorem, we can get the closed-form Gaussian posterior:

$$q\left(\tilde{\mathbf{X}}_{t-1}|\tilde{\mathbf{X}}_t, \tilde{\mathbf{X}}_0\right) = \frac{q\left(\tilde{\mathbf{X}}_t|\tilde{\mathbf{X}}_{t-1}, \tilde{\mathbf{X}}_0\right) q\left(\tilde{\mathbf{X}}_{t-1}|\tilde{\mathbf{X}}_0\right)}{q\left(\tilde{\mathbf{X}}_t|\tilde{\mathbf{X}}_0\right)}. \tag{4}$$

In continuous diffusion, the denoising network is trained to predict the unscaled Gaussian noise $\tilde{\epsilon}_t = (\tilde{\mathbf{X}}_t - \sqrt{\bar{\alpha}_t}\tilde{\mathbf{X}}_0)/\sqrt{1-\bar{\alpha}_t} = f_\theta(\tilde{\mathbf{X}}_t, t)$. The reverse process (Ho et al., 2020) uses a point estimation of $\tilde{\mathbf{X}}_0$ in the posterior:

$$p_\theta\left(\tilde{\mathbf{X}}_{t-1}|\tilde{\mathbf{X}}_t\right) = q\left(\tilde{\mathbf{X}}_{t-1}|\tilde{\mathbf{X}}_t, \frac{\tilde{\mathbf{X}}_t - \sqrt{1-\bar{\alpha}_t} f_\theta\left(\tilde{\mathbf{X}}_t, t\right)}{\sqrt{\bar{\alpha}_t}}\right). \tag{5}$$

To restore the discrete data, after generating the continuous $\tilde{\mathbf{X}}_0$, thresholding is applied to its first-row vector and convert it back to binary vector. Then, the generation model's output $\hat{\mathbf{X}}_0$ is obtained.

4.2. Feasibility/boundedness-constrained Sampling

Based on the above SPGM module, a new incidence matrix $\hat{\mathbf{A}}$ is generated. To ensure the new MILP instance is feasible and bounded, we propose a feasibility and boundedness-constrained sampling module (FBCSM) to generate $\mathbf{b}$ and $\mathbf{c}$, as depicted in Fig. 2. The primal format of a $\mathbf{MILP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$ has been provided in Eq. 1. By relaxing to the corresponding LP instance, we obtain the dual format of this relaxed instance as denoted by $\mathbf{DualLP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$:

$$\max_{\mathbf{y}} \mathbf{b}^\mathsf{T}\mathbf{y}, \quad \text{s.t.} \quad \mathbf{A}^\mathsf{T}\mathbf{y} \geq \mathbf{c}, \mathbf{y} \geq 0. \tag{6}$$

Solve for the primal instance $\mathbf{MILP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$ and the dual instance $\mathbf{DualLP}(\mathbf{A}, \mathbf{b}, \mathbf{c})$, and represent the sets of their feasible solutions as $\mathcal{S}_x$ and $\mathcal{S}_y$, which are consisted of collected feasible solutions $\mathbf{x_f}$ and $\mathbf{y_f}$ separately. It is important to emphasize that the solutions obtained need not be optimal. We randomly select from these sets to obtain $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$. Then, subsequent right-hand sampler and object coefficient sampler can be constructed based on $\hat{\mathbf{A}}$, $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$, thus the feasibility and boundedness can be guaranteed:

$$\hat{\mathbf{b}} \geq \hat{\mathbf{A}}\hat{\mathbf{x}}, \quad \hat{\mathbf{c}} \leq \hat{\mathbf{A}}^\mathsf{T}\hat{\mathbf{y}}, \tag{7}$$

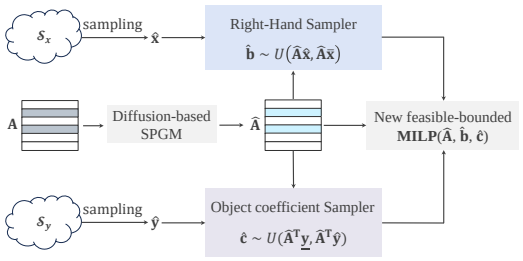


Figure 2. Our FBCSM module: $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are sampled from the sets of collected feasible solutions. Together with the newly generated $\hat{\mathbf{A}}$ from SPGM, $\hat{\mathbf{b}}$ and $\hat{\mathbf{c}}$ are constructed by sampling within the bounds on the one sides to ensure the feasibility and boundedness. Meanwhile, sampling bounds of $\hat{\mathbf{b}}$ and $\hat{\mathbf{c}}$ on the other sides are designed to maximize the tightness of generated constraints.

Proposition 4.1. ***(Boundedness and Feasibility Guarantee of MILP-FBGen):*** *MILP-FBGen ensures the new LP/MILP instance* $\mathbf{MILP}(\hat{\mathbf{A}}, \hat{\mathbf{b}}, \hat{\mathbf{c}})$ *is both feasible and bounded.*

Proof: $\hat{\mathbf{x}}$ *satisfy both* $\hat{\mathbf{x}} \in \mathbb{Z}^p_{\geq 0} \times \mathbb{Q}^{n-p}_{\geq 0}$ *and* $\hat{\mathbf{b}} \geq \hat{\mathbf{A}}\hat{\mathbf{x}}$, *therefore* $\mathbf{MILP}(\hat{\mathbf{A}}, \hat{\mathbf{b}}, \hat{\mathbf{c}})$ *at least has a feasible solution* $\hat{\mathbf{x}}$. *The feasibility of the instance is proved. Similarly,* $\hat{\mathbf{y}}$ *satisfy both* $\hat{\mathbf{y}} \geq 0$ *and* $\hat{\mathbf{c}} \leq \hat{\mathbf{A}}^\mathsf{T}\hat{\mathbf{y}}$ *and* $\hat{\mathbf{y}}$ *is the feasible solution of* $\mathbf{DualLP}(\hat{\mathbf{A}}, \hat{\mathbf{b}}, \hat{\mathbf{c}})$. *Based on the weak duality theorem,* $\hat{\mathbf{b}}^\mathsf{T}\hat{\mathbf{y}}$ *provides a lower bound on* $\mathbf{MILP}(\hat{\mathbf{A}}, \hat{\mathbf{b}}, \hat{\mathbf{c}})$. *The boundedness of the instance is proved.*

Presolving techniques play a crucial role in the resolution of LP/MILP instances. Bound and constraint tightening serves to propagate constraints and refine variable domains, eliminating redundancies. To maximize the tightness of the generated constraints, we ensure that each constraint is

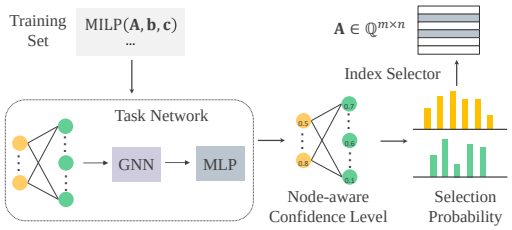


Figure 3. Task-oriented constraint/variable vector selection. For each training instance, node-aware confidence level can be extracted after the task network and used to construct the selection probability of each node. Nodes with worse performance in downstream tasks have a larger chance to be selected.

tightened to at least the extent propagated by the connected variable's range. Then, the construction of right-hand side constant and object coefficient can be further constrained by:

$$\hat{\mathbf{b}} \leq \hat{\mathbf{A}}\bar{\mathbf{x}}, \quad \hat{\mathbf{c}} \geq \hat{\mathbf{A}}^{\top}\underline{\mathbf{y}}. \tag{8}$$

Each element of $\bar{\mathbf{x}}$ and $\underline{\mathbf{y}}$ is formulated as

$$\begin{aligned} \bar{\mathbf{x}}_j &= \begin{cases} \mathbf{x}_j^l & \text{if} \quad \hat{\mathbf{A}}_{ij} \leq 0 \\ \mathbf{x}_j^u & \text{if} \quad \hat{\mathbf{A}}_{ij} > 0 \end{cases} \quad (j = 1, 2, \cdots, n), \\ \underline{\mathbf{y}}_i &= \begin{cases} \mathbf{y}_i^l & \text{if} \quad \hat{\mathbf{A}}_{ij} \geq 0 \\ \mathbf{y}_i^u & \text{if} \quad \hat{\mathbf{A}}_{ij} < 0 \end{cases} \quad (i = 1, 2, \cdots, m), \end{aligned} \tag{9}$$

where $\mathbf{x}_j^l$ and $\mathbf{x}_j^u$ are the lower bound and upper bound of $\mathbf{x}_j$ separately, $\mathbf{y}_i^l$ and $\mathbf{y}_i^u$ are corresponding bounds of $\mathbf{y}_i$. By combining Eq. 7 and Eq. 8, we derive the final sampling strategy for the right-hand side constant and object coefficient:

$$\hat{\mathbf{b}} \sim \text{Uniform}(\hat{\mathbf{A}}\hat{\mathbf{x}}, \hat{\mathbf{A}}\bar{\mathbf{x}}), \hat{\mathbf{c}} \sim \text{Uniform}(\hat{\mathbf{A}}^{\top}\underline{\mathbf{y}}, \hat{\mathbf{A}}^{\top}\hat{\mathbf{y}}). \tag{10}$$

Hence, the resulting instance can be succinctly represented as $\mathbf{MILP}(\hat{\mathbf{A}}, \hat{\mathbf{b}}, \hat{\mathbf{c}})$. As for the generation of LP instances, the processes involved are comparable, and we will refrain from repeating them here. This sampling strategy provides theoretical assurances regarding the boundedness and feasibility of the generated instances, ensuring the high quality of the produced data without superfluous constraints from both the primal and dual forms.

4.3. Constraint/variable Vector Selection

Constraint/variable vector selection is another crucial aspect, which directly influences the attributes of the generated instances. In the training phase, to ensure thorough training, constraints are randomly selected. However, during the inference phase, we offer two distinct strategies.

Random selection. The default strategy is easy to implement. It randomly selects among all the constraint vectors and can be applied to any dataset and any type of downstream tasks.

Task-oriented selection. Instance generation serves downstream tasks, so downstream tasks can in turn guide instance generation. As depicted in Fig. 3, the selection probability of constraint vectors is constructed based on the error between the outputs of downstream tasks and corresponding labels, thus the tasks currently are limited to node-level regression or classification. The worse the performance on a node, the greater the probability that the node is selected. The detailed procedure is presented in Appendix C. Based on the newly-generated instances, the original dataset is enriched and the downstream task network can also be further finetuned. For better performance, this process can be repeated.

4.4. Overall Pipeline

Bipartite graph representation. A weighted bipartite graph $\mathcal{G} = (V, W, E)$ is used to represent a MILP or LP instance, consisting of two disjoint sets of vertices V and W, and a collection E of weighted edges. Each vertex in V represents a decision variable v, and each vertex in W represents a constraint variable w. Based on the statistics information about these vertices, we can construct original vertex features, denoted as $\mathbf{e}_v$ and $\mathbf{e}_w$.

Forward noising process. Given a sample $\mathbf{X}_0$ from original dataset, we randomly select a timestamp t from the range of $[1, T]$ and then obtain the noised $\tilde{\mathbf{X}}_t$. It is embedded into the bipartite graph:

$$\mathbf{E}^v = \text{MLP}^v(\text{concat}(\mathbf{e}_v, \tilde{\mathbf{X}}_t)), \mathbf{E}^w = \text{MLP}^w(\mathbf{e}_w). \tag{11}$$

To introduce the time information, we map the timestamp t to a high-dimensional sinusoidal feature (Sun & Yang, 2023) and add it to above vertex features. We utilize the standard message-passing network (Fan et al., 2023) as the graph neural network and the learned vertex representations are denoted as $\tilde{\mathbf{E}}^v, \tilde{\mathbf{E}}^w = \text{GNN}(\mathbf{E}^v, \mathbf{E}^w)$. Next, we select the constraint feature $\tilde{\mathbf{E}}_i^w$ corresponding to the chosen constraint index i and embed it to each variable feature respectively. Subsequently, the enriched variable feature acts as the final encoded feature $\tilde{\mathbf{E}}^v$.

Reverse denoising process. Based on the encoded feature, we can output the predicted Gaussian noise $\tilde{\epsilon}_t = \text{MLP}(\tilde{\mathbf{E}}^v)$. During training process, Eq. 2 is equivalent to minimize

$$\mathcal{L} = ||\tilde{\epsilon}_t - \epsilon||^2, \epsilon \sim \mathcal{N}(0, \mathbf{I}). \tag{12}$$

With resort to the denoising formula of Eq. 5, we can gradually obtain the predicted $\hat{\mathbf{X}}_0$ and perform the multiplication operation between its row vectors to obtain $\hat{\mathbf{A}}_{i\cdot}$, which is used to replace the i_{th} constraint vector. Based on a predefined replacement ratio η, this process is performed in parallel to ensure that ηm vectors (for variable vectors, ηn) are generated and replaced.

Instance generation. Based on the newly-generated incidence matrix $\hat{\mathbf{A}}$, we use the module described in Sec. 4.2

to obtain $\hat{\mathbf{b}}$ and $\hat{\mathbf{c}}$, which can guarantee the feasibility and boundedness of the new instance $\mathbf{MILP}(\hat{\mathbf{A}}, \hat{\mathbf{b}}, \hat{\mathbf{c}})$.

5. Experiments

In this section, we first delineate the experimental settings, including datasets and baselines, Then, we provide corresponding numerical results and conduct detailed analysis to verify the superiority of our proposed method.

5.1. Datasets and Baselines

We execute MILP-FBGen across four MILP datasets and two LP datasets, covering a range of scenarios including simple and challenging instances. Considering factors such as the numbers of constraints, variables, non-zero elements in incidence matrix $\mathbf{A}$, and solving time, we categorize the instances into three groups: Easy, Medium, and Hard. A more detailed description can be found in Appendix A.

Easy: This category includes two MILP datasets, i.e., Set Covering (SC) (Balas & Ho, 1980) and Mixed-integer Knapsack (MIK) (Atamtürk, 2003). SC is synthetic and generated following previous work (Nair et al., 2020), and MIK is also a widely used dataset.

Medium: This group is consisted of two publicly available LP datasets, Maritime Inventory Routing Problems (MIRP) (Fan et al., 2023) and PDS (Mittelmann). PDS has extremely large problem sizes and can be used to test the performance of different models in the challenging scenario.

Hard: We adopt the MILP datasets utilized in ML4CO (Gasse et al., 2022), including Workload Apportionment (WA) and Anonymous. Due to the large scale of these datasets and excessive solving time, we select the first 100 instances as our datasets respectively.

Three baselines are considered, ranging from the heuristic method to the state-of-the-art (SOTA) deep generative models. Bowly (Bowly, 2019; Bowly et al., 2020) is a classic heuristic generator and can generate feasible and bounded instances. To ensure a fair comparison, we set all the controllable parameters of Bowly to match the statistics of the training set. G2MILP (Geng et al., 2023) is the first deep generative framework for MILP instance generation and preserves the training distribution well. DIG-MILP (Wang et al., 2023) can ensure the feasibility and boundedness of generated instances but at the cost of sacrificing the structural similarity.

5.2. Experimental Settings

We generate more instances w.r.t. each original one in the training set, to double, triple, or even quadruple the original dataset size. Without specification, the double one is used. For most datasets, instance generation is performed with different replacement ratios η, while for datasets with exceptionally large problem sizes, one proper ratio is kept to avoid generated instances deviating from the original training distribution. Among the two constraint/variable vector selection strategies, the Random one is the default option.

Our experiments mainly consist of two parts. For the first part, we evaluate the ability of property preservation between training and generated instances from three aspects: computational hardness, structural distributional similarity and feasibility. We utilize the commercial solver Gurobi (Gurobi) to solve instances, and report the average solving time (*Time*) and feasibility ratio (*F*). *Gap* is the relative deviation between the solving time of training data and that of generated data. As for the structural distributional similarity (*S*), we adopt the similarity score from (Geng et al., 2023), which ranges from 0 to 1.

The second part is to analyze the effect on the performance of downstream tasks using instances generated from various methods. We conduct evaluations on three tasks: initial basis prediction (Fan et al., 2023) on LP datasets, predict and search for the best feasible solution (Han et al., 2023) on MILP datasets, and hyperparameter tuning on LP datasets. We modify the open-source solver HiGHS (HiGHS) to enable it to receive an external initial basis and hyperparameters and use it to report the solving time (*TimeH*) on the unseen test set, after passing in the predicted initial basis or searched hyperparameters. For the predict and search task, we follow (Han et al., 2023) and use Gurobi to output the objective value after searching within a region around the predicted solutions. The result from the default solver without any assistance from ML models is named as *Default*, while the result only based on the original training set is denoted as *Baseline*. *Gain* is the relative improvement compared with *Baseline*, due to the usage of augmented instances. For more details, please refer to Appendix D.2.

5.3. Numerical Results

5.3.1. Property preservation

Computational hardness. Solving time can be considered as the direct reflection of computational hardness and we report the mean solving time of the training set and generated instances in Table 1. Typically, larger η leads to greater perturbation, and the corresponding gap becomes larger. Across all the datasets and ratios η, MILP-FBGen beats all the compared baselines with a much lower gap. The instances generated by Bowly perform poorly, primarily due to the significant simplicity of generated instances, and Bowly does not work for any medium and hard dataset. As for DIG-MILP, it also has a relatively large gap, because it changes all the constraints during instance generation and thus the generated instances fall out of the original distribution. G2MILP shows comparable performance in easy datasets, while for medium and hard datasets, our method shows significant superiority. Especially for the PDS dataset, both DIG-MILP and G2MILP can not handle such large instances, which greatly limits their applicability in real-

Table 1. Average solving time (Time), feasibility ratio (F), and structural distributional similarity score (S) by Gurobi.

		Easy: SC n=1000,m=500,#1000				Easy: MIK n=387,m=312,#90				Medium: MIRP n=28738,m=33085,#28			
		Time(s)	Gap(%) ↓	F(%) ↑	S ↑	Time(s)	Gap(%) ↓	F(%) ↑	S ↑	Time(s)	Gap(%) ↓	F(%) ↑	S ↑
Training Set		0.790		100		0.164		100		1.629		100	
Bowly[1]		0.014	98.23	100	0.175	0.001	99.39	100	-	-	-	-	-
$\eta = 0.01$	DIG-MILP	1.051	32.98	100	0.902	0.093	43.53	100	0.860	1.292	23.51	100	0.814
	G2MILP	0.796	0.83	100	0.812	0.129	21.49	100	**0.996**	1.024	37.13	96.4	0.895
	MILP-FBGen	**0.793**	**0.38**	**100**	**0.997**	**0.153**	**6.47**	**100**	0.946	**1.582**	**2.89**	**100**	**0.972**
$\eta = 0.05$	DIG-MILP	1.246	57.72	100	0.808	0.073	55.29	100	0.886	1.280	24.22	100	0.809
	G2MILP	0.731	7.51	100	0.861	0.103	37.17	100	**0.970**	1.136	30.26	85.7	0.898
	MILP-FBGen	**0.783**	**0.89**	**100**	**0.940**	**0.128**	**21.76**	**100**	0.884	**1.569**	**3.68**	**100**	**0.906**
$\eta = 0.1$	DIG-MILP	3.738	373.12	100	0.790	0.081	50.59	100	0.821	1.295	23.32	100	0.808
	G2MILP	0.715	9.52	100	0.819	0.057	65.21	100	**0.891**	0.722	55.68	57.1	0.872
	MILP-FBGen	**0.770**	**2.53**	**100**	**0.917**	**0.122**	**25.88**	**100**	0.804	**1.577**	**3.19**	**100**	**0.875**

		Medium:PDS,$\eta = 0.0001$ n=317105,m=188399,#9				Hard:Anonymous,$\eta = 0.01$ n=33999,m=55545,#98				Hard:WA,$\eta = 0.001$ n=61000,m=64305,#100			
		Time(s)	Gap(%) ↓	F(%) ↑	S ↑	Time(s)	Gap(%) ↓	F(%) ↑	S ↑	Time(s)	Gap(%) ↓	F(%) ↑	S ↑
Training Set		11.236		100		344.962		100		621.088		100	
DIG-MILP[2]		-	-	-	-	-	-	-	-	0.046	99.99	100	0.654
G2MILP[2]		-	-	-	-	189.410	45.09	81.6	0.879	70.841	88.59	16.0	**0.885**
MILP-FBGen		**12.273**	**9.22**	**100**	**0.937**	**295.718**	**14.28**	**100**	**0.947**	**468.621**	**24.55**	**100**	0.853

[1] Bowly cannot generate instances for hard, even medium datasets, including MIRP, PDS, WA and Anonymous. The similarity of Bowly on MIK is not reported because Ecole (Prouvost et al., 2020) fails to read its generated instances during computation due to large numerical values.

[2] Based on the official implementation, DIG-MILP and G2MILP cannot handle extremely large instances of PDS dataset. For Anonymous dataset, DIG-MILP cannot converge and incurs an infinite loss.

Table 2. Average solving time (TimeH) by HiGHS with predicted initial basis from different models.

Dataset	MIRP						PDS	
#Aug	MILP-FBGen		G2MILP		DIG-MILP		MILP-FBGen	
	TimeH(s)	Gain(%) ↑	TimeH(s)	Gain(%) ↑	TimeH(s)	Gain(%) ↑	TimeH(s)	Gain(%) ↑
Default	130.51		130.51		130.51		88.23	
Baseline	125.57		125.57		125.57		78.21	
1:1	**86.07**	**31.46**	98.32	21.70	112.03	10.78	75.70	3.21
1:2	88.91	29.19	94.43	24.80	109.55	12.76	76.14	2.65
1:3	92.07	26.68	101.81	18.92	125.90	-0.26	**63.24**	**19.14**

world scenarios. Besides solving time, we also provide the average branching nodes for MILP instances (iterations for LP) and the results can be found in Appendix D.1 with a similar conclusion.

Feasibility and structural distributional similarity. Our method can ensure both feasibility and boundedness, while boundedness is hard to violate, and most cases encounter infeasibility first. In our experiments, all generated instances are bounded, hence there is no need for specific reporting regarding their boundedness. From Table 1, the feasibility preservation advantage of MILP-FBGen is demonstrated by generating 100% feasible samples under different replacement ratios and different datasets. Due to the simplicity of SC and MIK datasets, all the methods can generate feasible samples. For medium and hard datasets, most of the instances generated by G2MILP are infeasible, especially with a larger replacement ratio. For the Anonymous dataset, the feasibility ratio of G2MILP is 81.6%, and for the WA dataset, it decreases to 16%, which means most of the generated samples are infeasible for this dataset and cannot be applied to downstream tasks. Although Bowly and DIG-MILP can also guarantee feasibility, their generated instances have low similarity compared with the training set, making them less usable in original tasks. In contrast to them, our method always has higher similarity, which means the statistical characteristics are well maintained.

5.3.2. Evaluation on downstream tasks

Initial basis prediction. The solving process in solver can be greatly boosted by starting with a basis much closer to an optimal one. This is a 3-classification task in essence and the direct classification accuracy can be found in Appendix D.2.1. To show the effect of the predicted basis on the solving process, we also provide the end-to-end solving time on the test set. We conduct experiments on MIRP and PDS datasets, which construct augmented datasets with mixed ratio and 0.0001 separately. From the results in Table 2, we can observe that our method owns the shortest time and the largest gain compared with *Baseline*, especially

Table 3. Average objective value (OBJ) given by different methods. Please note DIG-MILP cannot generate instances for Anonymous.

Dataset	BKS	Default	Baseline		MILP-FBGen			DIG-MILP			G2MILP		
			OBJ	$\text{gap}_{\text{abs}} \downarrow$	OBJ	$\text{gap}_{\text{abs}} \downarrow$	Gain(%) $\uparrow$	OBJ	$\text{gap}_{\text{abs}} \downarrow$	Gain(%) $\uparrow$	OBJ	$\text{gap}_{\text{abs}} \downarrow$	Gain(%) $\uparrow$
SC	228.36	228.36	228.36	0	228.36	0	-	228.36	0	-	228.36	0	-
WA	701.52	701.64	701.68	0.16	**701.60**	**0.08**	**50.00**	701.64	0.12	25.00	701.67	0.15	6.25
Anonymous	11545.78	11904.86	11904.09	358.31	**11900.94**	**355.16**	**0.879**	-	-	-	11906.63	360.85	-0.709

when doubling the original dataset (#Aug=1:1). When excessive augmented instances are used, solving time increases across all the methods, which may be due to the accumulation of noise in the enhanced data and then the original training distribution is disturbed to some extent. In the case of DIG-MILP with #Aug=1:3, the solving time is even a little larger than *Baseline*. However, the increase from our method is slight, which demonstrates the stability and high quality of our generated instances. For the PDS dataset, it is quite small and the problem size is very large, so data augmentation is extremely important for it. However, neither G2MILP and DIG-MILP can generate an instance for it. With the increase in the total number of augmented instances from our method, the solving time is significantly reduced.

Predict and search. This task aims at efficiently identifying high-quality feasible solutions. Specifically, the marginal probability of each binary variable is predicted, and then the solver searches for the best feasible solution within a properly defined ball around the predicted solution. By tightening the search area to a compact one closer to the optimal solution, the optimal solution can be found faster and better, especially in a limited time. Following the evaluation method in (Han et al., 2023), we report the average objective value of incumbent solutions across test instances at 1000 seconds as OBJ. A single-thread solver is run for 3600 seconds and its results are denoted as BKS, the objective of the incumbent solutions. The absolute primal gap, defined as $\text{gap}_{\text{abs}} = |\text{OBJ} - \text{BKS}|$, acts as the performance metrics and a smaller primal gap indicates a stronger performance. *Gain* is computed between the gap_{abs} of each model and that of *Baseline*. We conduct experiments on three MILP datasets and the results by Gurobi (Gurobi) are shown in Table 3. SC is easily solved by default solver, thus there is no difference in objective values for this dataset. For WA, our method results in 50% improvement compared with *Baseline* and surpasses DIG-MILP and G2MILP significantly. The Anonymous dataset is hard to solve within 1000 seconds and thus the gap between OBJ and BKS is relatively large. In that case, we decrease this gap, while G2MILP performs even worse than *Baseline*.

Hyperparameter tuning. While the above two tasks focus on node-level classification tasks, it is noteworthy that MILP-FBGen possesses the capability to handle instance-level and dataset-level optimization as well. Hyperparameter tuning on Linear Programming Solver is a critical instance-level task that ensures the solver is finely tuned to maximize its efficiency and effectiveness. We choose 11 key parameters of the primal simplex method in HiGHS (HiGHS) and the details of parameters are described in Appendix D.2.2. SMAC (Hutter et al., 2011) is used to search for the best hyperparameter configurations on the training or augmented datasets separately and we report the average solving time on the test set with the tuned hyperparameters. We conduct the experiments on MIRP and PDS datasets, which are both LP datasets, and the results are reported in Table 4. In MIRP, the configuration searched only with training data performs worse than *Default*. By using the augmented dataset of our method, the searched configuration is much better, thus the solving time decreases by 4.14% compared with *Baseline* in MIRP and 19.41% in PDS, demonstrating the effectiveness of our method in the hyperparameter tuning task. However, G2MILP shows little improvement and DIG-MILP performs extremely worse, mainly due to its complete modification of all the constraints, thus its generated instances fall out of the original training distribution. Both of them cannot generate instances in PDS, due to its extremely large problem size.

Table 4. Average solving time (TimeH) by HiGHS with searched hyperparameters of different methods.

Dataset	MIRP		PDS	
	TimeH(s)	Gain(%) $\uparrow$	TimeH(s)	Gain(%) $\uparrow$
Default	28.89		276.58	
Baseline	29.25		36.68	
G2MILP	29.16	0.31	-	-
DIG-MILP	101.27	-246.22	-	-
MILP-FBGen	**28.04**	**4.14**	**29.56**	**19.41**

Table 5. Performance comparison between constraint vector selection strategies on MIRP and PDS.

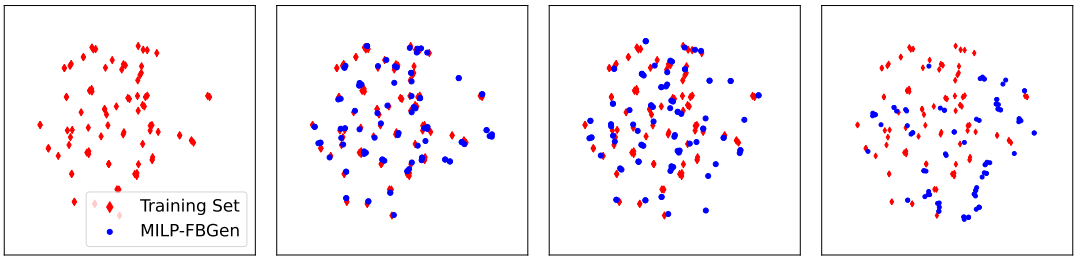
	MIRP, $\eta = 0.01$		PDS, $\eta = 0.0001$	
	TimeH(s)	Gain(%) $\uparrow$	TimeH(s)	Gain(%) $\uparrow$
Baseline	125.57		78.21	
FBGen-Random	86.07	31.46	75.70	3.21
FBGen-Task	**83.41**	**33.57**	**73.55**	**5.96**

5.4. Ablations and Further Study

Which selection strategy of constraints is better? We compare the results of two different selection strategies and denote them as FBGen-Random and FBGen-Task respectively. As shown in Table 5, we use them to double the original dataset size separately and compare their perfor-

Table 6. Source of gain evaluated by Gurobi on SC, MIK and MIRP ($\eta = 0.01$).

	SPGM		FBCSM		SC		MIK		MIRP	
	Diffusion	VAE	Eq. 7	Eq. 8	Time(s)	Gap(%) ↓	Time(s)	Gap(%) ↓	Time(s)	Gap(%) ↓
Training set	-	-	-	-	0.790		0.164		1.629	-
MILP-FBGen	✓		✓	✓	**0.793**	**0.38**	**0.153**	**6.47**	**1.58**	**2.90**
MILP-FBGen-V1	✓		✓		0.800	1.29	0.197	20.12	1.44	11.80
MILP-FBGen-V2		✓	✓	✓	0.798	0.97	0.144	12.35	1.51	**7**.40



(a) Training set (b) η=0.01 (c) η=0.05 (d) η=0.1

Figure 4. The t-SNE visualization of instance representations for MIK.

mance with the *Baseline*. We report the solving time from HiGHS after feeding the predicted initial basis from these two methods to the solver and task-oriented selection strategy leads to the best performance in both two datasets. It is quite reasonable because constraints are selected based on their performance in downstream tasks. Constraints with worse performance are much easier to be chosen, thus more instances are generated around these hard cases and subsequent training can be more adequate. We also compare their preservation capacities of computational hardness and the results are shown in Table 12 of Appendix. It can be observed that these two selection strategies both perform well in this aspect and task-oriented selection shows more superiority in the downstream tasks.

Is maximizing the tightness of constraints important? To demonstrate the necessity of maximizing the tightness when generating $\hat{\mathbf{b}}$ and $\hat{\mathbf{c}}$ for the constraints of primal and dual instances, we define a variant MILP-FBGen-V1, which generates them without the sampling bounds of Eq. 8. Table 6 presents the results of MILP-FBGen with replacement ratio $\eta = 0.01$ and the results with more ratios can be found in Appendix D.2.3. We can observe that the performance of MILP-FBGen is better than MILP-FBGen-V1, which indicates that simply setting the constraints by Eq. 7 is insufficient and the generated constraints are more likely to be removed during presolving, leading to a shift compared with the solving time of training set.

Is the diffusion model a better choice to preserve the structures? To validate the superiority of the diffusion model in instance generation, we replace it with another generative model variational autoencoder (VAE) in our structure-preserving generation module, while maintaining the subsequent feasibility/boundedness-constrained sampling module. We denote this version as MILP-FBGen-V2 and the comparative results can be seen in Table 6. The improved performance of MILP-FBGen over MILP-FBGen-V2 implies that the diffusion model has better generative capacity.

t-SNE visualization. To visually verify the claim that the generated instances by MILP-FBGen can well extend the original problem space, we provide the instance representations for MIK, utilizing the visualization technique in (Van der Maaten & Hinton, 2008). The results are shown in Fig. 4, where each point represents an instance. Red points are from the training set and blue points are instances generated by MILP-FBGen. The generated instances, while closely resembling the training set, contribute to a broader and more continuous exploration of the problem space, thereby enhancing model robustness and generalization. Additionally, by increasing η, it can explore a wider problem space beyond the confines of the training sets.

6. Conclusions

In this paper, we have emphasized three pivotal properties - computational hardness, feasibility, and boundedness (especially the first two) for LP/MILP instances. The MILP-FBGen is proposed to generate new instances endowed with these essential attributes. Integrating MILP-FBGen into downstream tasks through the task-oriented selection strategy facilitates the creation of more refined instances, thereby enhancing overall performance. Through extensive experiments, we substantiate the effectiveness and superiority of MILP-FBGen compared to other LP/MILP instance generation methods. Concurrently, it is identified that the reverse diffusion process remains time-consuming. We anticipate and advocate for further endeavors to expedite the diffusion-based LP/MILP instance generation process.

Impact Statement

This study in general belongs to the realm of generative AI by extending it from classic image, and text data into the (more complicated) constrained combinatorial optimization problem instances, specifically Linear Programming (LP) and Mixed-Integer Linear Programming (MILP). Like other generative AI models, our techniques may also have some potential to impact the world in many aspects, yet we believe our results are less sensitive than those generating human faces, etc.

Acknowledgements

The work was in part supported by NSFC (92370201, 62222607).

References

Atamtürk, A. On the facets of the mixed–integer knapsack polyhedron. *Mathematical Programming*, 98(1-3):145–175, 2003.

Balas, E. and Ho, A. *Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study*. Springer, 1980.

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

Bowly, S., Smith-Miles, K., Baatar, D., and Mittelmann, H. Generation techniques for linear programming instances with controllable properties. *Mathematical Programming Computation*, 12(3):389–415, 2020.

Bowly, S. A. *Stress testing mixed integer programming solvers through new test instance generation methods*. PhD thesis, School of Mathematical Sciences, Monash University, 2019.

Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Veličković, P. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24(130):1–61, 2023.

Chen, T., Zhang, R., and Hinton, G. Analog bits: Generating discrete data using diffusion models with self-conditioning. *arXiv preprint arXiv:2208.04202*, 2022.

Chen, X., Li, Y., Wang, R., and Yan, J. Mixsatgen: Learning graph mixing for sat instance generation. In *The Twelfth International Conference on Learning Representations*, 2024.

Fan, Z., Wang, X., Yakovenko, O., Sivas, A. A., Ren, O., Zhang, Y., and Zhou, Z. Smart initial basis selection for linear programs. In *International Conference on Machine Learning*, pp. 9650–9664. PMLR, 2023.

Gasse, M., Chetelat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32, 2019.

Gasse, M., Bowly, S., Cappart, Q., Charfreitag, J., Charlin, L., Chételat, D., Chmiela, A., Dumouchelle, J., Gleixner, A., Kazachkov, A. M., et al. The machine learning for combinatorial optimization competition (ml4co): Results and insights. In *NeurIPS 2021 competitions and demonstrations track*, pp. 220–231. PMLR, 2022.

Geng, Z., Li, X., Wang, J., Li, X., Zhang, Y., and Wu, F. A deep instance generative framework for milp solvers under limited data availability. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Guo, W., Zhen, H., Li, X., Luo, W., Yuan, M., Jin, Y., and Yan, J. Machine learning methods in solving the boolean satisfiability problem. *Machine Intelligence Research*, 20: 640—-655, 2023.

Gupta, P., Gasse, M., Khalil, E., Mudigonda, P., Lodi, A., and Bengio, Y. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33: 18087–18097, 2020.

Gurobi. Gurobi optimizer reference manual. http://www.gurobi.com. 2020.

Han, Q., Yang, L., Chen, Q., Zhou, X., Zhang, D., Wang, A., Sun, R., and Luo, X. A gnn-guided predict-and-search framework for mixed-integer linear programming. *arXiv preprint arXiv:2302.05636*, 2023.

He, H., Daume III, H., and Eisner, J. M. Learning to search in branch and bound algorithms. *Advances in neural information processing systems*, 27, 2014.

Helsgaun, K. An effective implementation of the lin–kernighan traveling salesman heuristic. *European journal of operational research*, 126(1):106–130, 2000.

HiGHS. Highs- high performance software for linear optimization documentation. https://highs.dev/top. 2020.

Hill, R., Moore, J., Hiremath, C., and Cho, Y. Test problem generation of binary knapsack problem variants and the implications of their use. *Int. J. Oper. Quant. Manag*, 18 (2):105–128, 2011.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Advances in neural information processing systems*, volume 33, pp. 6840–6851, 2020.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pp. 507–523. Springer, 2011.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Labassi, A. G., Chételat, D., and Lodi, A. Learning to compare nodes in branch and bound with graph neural networks. *Advances in Neural Information Processing Systems*, 35:32000–32010, 2022.

Li, ., X, Chen, Guo, W., Li, X., Luo, W., Huang, J., Zeng, H., Yuan, M., and Yan, J. Hardsatgen: Understanding the difficulty of hard sat formula generation and a strong structure-hardness-aware baseline. In *Advances in neural information processing systems*, 2023a.

Li, A., Han, C., Guo, T., and Li, B. Generating linear programming instances with controllable rank and condition number. *Computers & Operations Research*, 162:106471, 2024.

Li, Y., Guo, J., Wang, R., and Yan, J. T2t: From distribution learning in training to gradient search in testing for combinatorial optimization. In *Advances in neural information processing systems*, 2023b.

Malitsky, Y., Merschformann, M., O'Sullivan, B., and Tierney, K. Structure-preserving instance generation. In *Learning and Intelligent Optimization: 10th International Conference, LION 10, Ischia, Italy, May 29–June 1, 2016, Revised Selected Papers 10*, pp. 123–140. Springer, 2016a.

Malitsky, Y., Merschformann, M., O'Sullivan, B., and Tierney, K. Structure-preserving instance generation. In *Learning and Intelligent Optimization: 10th International Conference, LION 10, Ischia, Italy, May 29–June 1, 2016, Revised Selected Papers 10*, pp. 123–140. Springer, 2016b.

Mittelmann, H. Benchmark of simplex lp solvers. URL https://plato.asu.edu/ftp/lptestset/.

Nair, V., Bartunov, S., Gimeno, F., Von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Papageorgiou, D. J., Nemhauser, G. L., Sokol, J., Cheon, M.-S., and Keha, A. B. Mirplib–a library of maritime inventory routing problem instances: Survey, core model, and benchmark results. *European Journal of Operational Research*, 235(2):350–366, 2014.

Pilcher, M. G. and Rardin, R. L. Partial polyhedral description and generation of discrete optimization problems with known optima. *Naval Research Logistics (NRL)*, 39 (6):839–858, 1992.

Prouvost, A., Dumouchelle, J., Scavuzzo, L., Gasse, M., Chételat, D., and Lodi, A. Ecole: A gym-like library for machine learning in combinatorial optimization solvers. *arXiv preprint arXiv:2011.06069*, 2020.

Schrijver, A. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

SCIP. The scip optimization suite 8.0. https://www.scipopt.org/. 2021.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. PMLR, 2015.

Sun, Z. and Yang, Y. Difusco: Graph-based diffusion solvers for combinatorial optimization. *arXiv preprint arXiv:2302.08224*, 2023.

Tang, L., Liu, J., Rong, A., and Yang, Z. A review of planning and scheduling systems and methods for integrated steel production. *European Journal of operational research*, 133(1):1–20, 2001.

Troncoso, J. J. and Garrido, R. A. Forestry production and logistics planning: an analysis using mixed-integer programming. *Forest Policy and Economics*, 7(4):625–633, 2005.

Van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

Vander Wiel, R. J. and Sahinidis, N. V. Heuristic bounds and test problem generation for the time-dependent traveling salesman problem. *Transportation Science*, 29(2):167–183, 1995.

Wang, H., Liu, J., Chen, X., Wang, X., Li, P., and Yin, W. Dig-milp: a deep instance generator for mixed-integer linear programming with feasibility guarantee. *arXiv preprint arXiv:2310.13261*, 2023.

Wang, R., Yan, J., and Yang, X. Combinatorial learning of robust deep graph matching: an embedding based approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

Whitley, D. A genetic algorithm tutorial. *Statistics and computing*, 4:65–85, 1994.

Ye, H., Xu, H., Wang, H., Wang, C., and Jiang, Y. Gnn&gbdt-guided fast optimizing framework for large-scale integer programming. In *International Conference on Machine Learning*, pp. 39864–39878. PMLR, 2023.

Yehuda, G., Gabel, M., and Schuster, A. It's not what machines can learn, it's what we cannot teach. In *International conference on machine learning*, pp. 10831–10841. PMLR, 2020.

Zarpellon, G., Jo, J., Lodi, A., and Bengio, Y. Parameterizing branch-and-bound search trees to learn branching policies. In *Proceedings of the aaai conference on artificial intelligence*, volume 35, pp. 3931–3939, 2021.

Zhang, J., Liu, C., Li, X., Zhen, H., Yuan, M., Li, Y., and Yan, J. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519(28):205–217, 2023.

A. Dataset Description

To fully demonstrate the superiority of our method, we conduct comprehensive experiments on diverse datasets. Mainly based on their solving time, we classify all the used datasets into three categories, i.e., easy, medium, and hard. Please note the #Non-Zero represents the number of non-zero elements in the incidence matrix $\mathbf{A}$, which is also a typical indicator and directly reflects the problem scale. Please note that the MIRP dataset we use is actually an LP subset of the complete dataset (Papageorgiou et al., 2014), which includes both LP and MILP instances, and this follows the way (Fan et al., 2023) does.

Table 7. Statistics of datasets used in experiments.

Dataset	SC	MIK	MIRP	PDS	WA	Anonymous
# Category	easy	easy	medium	medium	hard	hard
# Data	1000	90	28	9	100	98
# Variable	1000	387	28738	317105	61000	33999
# Constraint	500	312	33085	188399	64305	55545
# Non-Zero	25000	16950	226081	1668084	361774	1149037

B. Bipartite Graph Representation

To represent an LP or MILP instance, the weighted bipartite graph (Gasse et al., 2019) is used. This representation method also enables both LP and MILP instances to serve as the input for deep learning neural networks and has been widely used in recent research. A weighted bipartite graph $\mathcal{G} = (V, W, E)$ consists of two disjoint sets of vertices V and W, and a collection E of weighted edges. Each edge connects exactly one vertex from V and one vertex from W. Each vertex in V represents a variable, and each vertex in W represents a constraint. More specifically,

- Set $V = \{v_1, v_2, \cdots, v_n\}$ contains n nodes, each representing a decision variable.
- Set $W = \{w_1, w_2, \cdots, w_m\}$ is composed of m nodes and each of them represents a constraint.
- Set $E = \{e_{ji}\}$ and $e_{ji} = A_{ij}$, for all $(i, j) \in [m] \times [n]$.

We follow the method in (Fan et al., 2023) and construct 7-dim features for constraint node i and variable node j, based on extracting the information in $\mathbf{c}$, $\mathbf{A}$, l^x, u^x, l^s and u^s. Denote $\mathbf{A}_{i\cdot}$ and $\mathbf{A}_{\cdot j}$ respectively the i-th row and j-th column in the incidence matrix $\mathbf{A}$. Denote by $\langle \cdot, \cdot \rangle$ the cosine similarity between two vectors. The specific description for each feature dimension is shown in Table 8. To ensure numerical stability, tag dimensions (feature index=5,7) are added. If the lower/upper bound does not exist, the tag dimensions will be marked as -1/+1.

Table 8. Feature for each constraint node i and variable node j.

Feature index	Constraint node i	Variable node j
1	$\langle \mathbf{A}_{i\cdot}, \mathbf{c} \rangle$	c_j
2	$\langle \mathbf{A}_{i\cdot}, l^x \rangle$	$\langle l^s, \mathbf{A}_{\cdot j} \rangle$
3	$\langle \mathbf{A}_{i\cdot}, u^x \rangle$	$\langle u^s, \mathbf{A}_{\cdot j} \rangle$
4	$\begin{cases} l_i^s & \text{if } l_i^s \neq -\infty \\ 0 & \text{else} \end{cases}$	$\begin{cases} l_j^x & \text{if } l_j^x \neq -\infty \\ 0 & \text{else} \end{cases}$
5	$\begin{cases} 0 & \text{if } l_i^s \neq -\infty \\ -1 & \text{else} \end{cases}$	$\begin{cases} 0 & \text{if } l_j^x \neq -\infty \\ -1 & \text{else} \end{cases}$
6	$\begin{cases} u_i^s & \text{if } u_i^s \neq \infty \\ 0 & \text{else} \end{cases}$	$\begin{cases} u_j^x & \text{if } u_j^x \neq \infty \\ 0 & \text{else} \end{cases}$
7	$\begin{cases} 0 & \text{if } u_i^s \neq \infty \\ 1 & \text{else} \end{cases}$	$\begin{cases} 0 & \text{if } u_j^x \neq \infty \\ 1 & \text{else} \end{cases}$

C. Training and Inference Process

The detailed descriptions of training and inference (instance generation) processes are provided in Algorithm 1. With regard to the two different constraint/variable vector selection strategies, we take the task-oriented selection as an example and choose the node-level $k-$ classification as the downstream task.

Algorithm 1 MILP-FBGen with task-oriented selection strategy.

Input: Original dataset D and node-level k-classification as downstream task.
/* Training Phase
Randomly select constraint vectors to form batches.
Structure-preserving generator training (Eq. 12).
/* Inference Phase
Pretrain the task network using the original dataset D.
The softmax output of node i is $o \in \mathbb{R}^{1\times k}$ with label index c.
The confidence level for correct classification is o^c and selection probability is $p_i = 1 - o^c$.
Take the softmax for $\{p_i, i \in [1, m]\}$ and denote as P.
Select constraint vectors based on P to form batches.
for instance $\leftarrow 1$ to N **do**
  Select constraint index $i \sim P$ and generate $\hat{\mathbf{A}}_i$ in parallel until ηm vectors are processed (SPGM).
  Replace with $\hat{\mathbf{A}}_{i\in\{1,\dots,\eta m\}}$ to obtain $\hat{\mathbf{A}}$.
  Sample $\hat{\mathbf{b}}$ and $\hat{\mathbf{c}}$ (FBCSM).
  Generate new instance $\mathbf{MILP}(\hat{\mathbf{A}}, \hat{\mathbf{b}}, \hat{\mathbf{c}})$.
end for

D. Supplementary Experimental Results

D.1. Computational Hardness

We report the average numbers of branching nodes or iterations in Table 9, which can also reflect the computational hardness besides the average solving time. The conclusion is the same as that of average solving time that we have the better ability to preserve the computational hardness and thus the generated instances can closely resemble the original training data. Please note that for this part of the experiments, we set the running time limit of Gurobi as 900 seconds and set the hyperparameter *Params.Method* as 2. We double the original dataset and use all the generated instances of each method to calculate the computational hardness indicators.

D.2. Downstream Tasks

One direct approach to verify the effectiveness of generated data is through downstream task experiments, which is also the motivation for instance generation. Therefore, we choose three representative application scenarios, i.e., initial basis prediction task, predict and search task, and hyperparameter tuning task. For each task, 60% of the original dataset is used as the training set, 15% as the validation set, and the left 25% acts as the test set. Our reported results are all based on the unseen test set. When some methods generate unfeasible instances for specific datasets, we repeat them multiple times until a specified number of generated instances are collected to ensure the fairness of the comparison experiments. Without specification, we randomly select from the generated instances with three replacement ratios (0.01, 0.05, and 0.01), and use them to construct an augmented dataset with a mixed ratio for downstream tasks. As for some datasets with extremely large instances, we always keep one proper replacement ratio.

D.2.1. Initial Basis Prediction

As for the initial basis prediction task, it is performed on the MIRP and PDS datasets. Initial basis prediction is quite an effective task in the simplex method. Hence, starting with a basis that is much closer to an optimal one can significantly boost the solving process. We follow the model structure from (Fan et al., 2023). To efficiently handle the large-scale sparse constraint-variable bipartite graph, one graph convolutional layer is implemented as two sparse matrix multiplication respectively for message passing from constraint to variable node and back. By default, five-layer GNN is used, and the size of the hidden layers is 128 and the dropout ratio is 0.1. We train for 800 epochs and select the best checkpoint based on the

Table 9. Average numbers of branching nodes for MILP and iterations for LP. η is the replacement ratio.

		Easy: SC n=1000,m=500,#1000		Easy: MIK n=387,m=312,#90		Medium: MIRP n=28738,m=33085,#28	
		Nums of Node	Gap(%) ↓	Num of Node	Gap(%) ↓	Nums of iter	Gap(%) ↓
Training Set		730.945		131.570		4422.178	
$\eta = 0.01$	DIG-MILP	13714.000	1776.20	17.089	87.01	5910.036	33.65
	G2MILP	760.475	4.04	108.000	17.91	3910.176	11.58
	MILP-FBGen	**731.715**	**0.11**	**141.200**	**7.32**	**4873.930**	**10.22**
$\eta = 0.05$	DIG-MILP	5634.900	671.25	21.244	83.85	5692.429	28.72
	G2MILP	**761.640**	**4.20**	372.000	182.74	3555.250	19.60
	MILP-FBGen	762.485	4.31	**152.130**	**15.63**	**5162.007**	**16.73**
$\eta = 0.1$	DIG-MILP	3271.000	347.51	**65.111**	**50.51**	5831.750	31.88
	G2MILP	762.010	4.25	331.400	151.88	1969.110	55.47
	MILP-FBGen	**708.710**	**3.04**	218.930	66.40	**5471.257**	**23.72**

	Medium:PDS,$\eta = 0.0001$ n=317105,m=188399,#9		Hard:Anonymous,$\eta = 0.01$ n=33999,m=55545,#98		Hard:WA,$\eta = 0.001$ n=61000,m=64305,#100	
	Nums of Iter	Gap(%) ↓	Nums of Node	Gap(%) ↓	Nums of Node	Gap(%) ↓
Training Set	38185.556		355232.898		60288.22	
DIG-MILP	-	-	-	-	0	100
G2MILP	-	-	187037.849	47.35	7089.93	88.24
MILP-FBGen	**43474.889**	**13.85**	**202429.398**	**43.02**	**44453.71**	**26.26**

validation accuracy.

We report the end-to-end solving time by HiGHS after feeding the predicted basis to the solver. During this process, the maximum running time is set to 3600 seconds. Besides that, we also provide the direct classification accuracy, because initial basis prediction is actually a 3-classification task. As shown in Table 10, the classification accuracy of our method is approximately inversely proportional to the end-to-end solving time, because accurately predicted the initial basis boost solving process more significantly.

Table 10. Classification accuracy(%) of initial basis prediction models.

	MIRP, mixed ratio			PDS, $\eta = 0.0001$		
	1:1	1:2	1:3	1:1	1:2	1:3
Baseline	81.73	81.73	81.73	73.68	73.68	73.68
MILP-FBGen	83.60	82.45	82.33	73.69	74.30	74.71

D.2.2. Hyperparameter Tuning

Table 11 is the detailed description of used hyperparameters. Please note that part of the hyperparameters is not built-in to HiGHS, but we have extracted and renamed them ourselves. During this experiment, we set the maximum running time of HiGHS as 3600 seconds.

D.2.3. Ablation experiments

Constraint/variable vector selection. We compare the effect of different selection strategies on downstream tasks and verify the effectiveness of the task-oriented selection strategy. Meanwhile, we also provide the comparison on preservation capacities of computational hardness, and the results are shown in Table 12. We can observe that both of them have good preservation capacities compared with current methods and their results only have slight differences.

Source of gain. Besides the results with $\eta = 0.01$, we also provide the results with more replacement ratios for full demonstration. The conclusion is similar that for most datasets, MILP-FBGen performs better than those two versions.

Table 11. Detailed description of hyperparameters in HiGHS.

Hyperparameter	Value Type	Selecting Range	Description
crashStrategy	category	$\{0, 1, 2\}$	strategy for simplex startup basis
simplexStrategy	category	$\{0, 1, 2\}$	to choose primal or dual simplex method
dualizeStrategy	category	$\{0, 1, 2\}$	whether to solve the dualized problem
tightenStrategy	category	$\{0, 1, 2\}$	whether to tighten primal bound
matrixStrategy	category	$\{0, 1\}$	try to use sparse matrix with only ± 1
PrimalPricingStrategy	category	$\{0, 1, 2, 3\}$	pricing strategy in primal
DualPricingStrategy	category	$\{0, 1, 2, 3\}$	pricing strategy in dual
PricingWiseThv	float	$[0, 1]$	density threshold for column-wise or row-wise
PricingDenseThv	float	$[0, 1]$	density threshold for sparse or dense method
dualizePresolveStrategy	category	$\{0, 1, 2\}$	strategy for cowork between dualize and presolve
crashPresolveStrategy	category	$\{0, 1, 2\}$	strategy for cowork between crash and presolve

Table 12. Preservation capacities of computational hardness with different constraint vector selection strategies on MIRP and PDS.

	MIRP, $\eta = 0.01$		PDS, $\eta = 0.0001$	
	Time(s)	Gap(%) ↓	Time(s)	Gap(%) ↓
Baseline	1.629		11.236	
FBGen-Random	1.582	2.89	12.273	9.22
FBGen-Task	1.589	2.45	12.210	8.63

Table 13. Source of gain evaluated by Gurobi on SC, MIK and MIRP with $\eta = 0.05$

Method	SPGM		FBCSM		SC		MIK		MIRP	
	Duffision	VAE	Eq. 7	Eq. 8	Time(s)	Gap(%) ↓	Time(s)	Gap(%) ↓	Time(s)	Gap(%) ↓
Training set	-	-	-	-	0.790		0.164		1.629	-
MILP-FBGen	✓		✓	✓	0.780	0.90	**0.128**	**21.76**	**1.57**	**3.60**
MILP-FBGen-V1	✓		✓		**0.794**	**0.54**	0.224	36.60	1.43	12.20
MILP-FBGen-V2		✓	✓	✓	0.760	3.77	0.126	22.94	1.70	4.40

Table 14. Source of gain evaluated by Gurobi on SC, MIK, and MIRP with $\eta = 0.1$

Method	SPGM		FBCSM		SC		MIK		MIRP	
	Duffision	VAE	Eq. 7	Eq. 8	Time(s)	Gap(%) ↓	Time(s)	Gap(%) ↓	Time(s)	Gap(%) ↓
Training set	-	-	-	-	0.790		0.164		1.629	-
MILP-FBGen	✓		✓	✓	**0.770**	**2.50**	0.122	25.88	**1.580**	**3.20**
MILP-FBGen-V1	✓		✓		0.768	2.80	0.219	33.60	1.410	13.50
MILP-FBGen-V2		✓	✓	✓	0.770	2.58	**0.141**	**14.13**	1.530	6.30