Learning to Generate Scalable MILP Instances

Tianxing Yang
yangtx7@mail2.sysu.edu.cn
Tsinghua University
Beijing, China

Huigen Ye
yhg23@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Hua Xu*
xuhua@tsinghua.edu.cn
Tsinghua University
Beijing, China

ABSTRACT

Large-scale Mixed-Integer Linear Programming (MILP) problems have been efficiently addressed using Machine Learning (ML)-based frameworks, especially ML-based evolutionary optimization frameworks to obtain high-quality solutions. When addressing real-world MILP problems, ML-based evolutionary optimization frameworks often face challenges in acquiring sufficient instances that belong to the same category. This underscores the need for generators that can autonomously produce MILP problems from existing instances. This paper introduces MILPGen, a novel generative framework for autonomous MILP instance generation. Our key contribution lies in the two-stage problem generation in MILPGen: 1) Node Splitting and Merging, which splits the bipartite graph and tries to reconstruct it; 2) Scalable Problem Construction, which concatenates tree structures to get larger problems. We demonstrate that the instances generated by MILPGen are highly similar to the original problem instances and can effectively enhance the solution effect of the ML-based evolutionary optimization frameworks. Further experiments show that the scaled-up generated instances still retain the problem's structural properties, validating the proposed framework's effectiveness.

CCS CONCEPTS

• **Mathematics of computing → Mixed discrete-continuous optimization**; • **Computing methodologies** → *Unsupervised learning*.

KEYWORDS

mixed-integer linear programming, optimization, unsupervised learning, benchmarking

ACM Reference Format:
Tianxing Yang, Huigen Ye, and Hua Xu. 2024. Learning to Generate Scalable MILP Instances. In *Proceedings of The Genetic and Evolutionary Computation Conference 2024 (GECCO '24 Companion).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Hua Xu is Corresponding author.

1 INTRODUCTION

Mixed-integer linear programming (MILP) is an extension of linear programming that addresses problems where at least one variable must take a discrete integer value instead of a continuous one [9]. For large-scale MILP problems, leveraging Machine Learning (ML)-based frameworks [2, 6], especially ML-based evolutionary optimization frameworks [10], to find high-quality solutions has become increasingly popular due to its capacity to strike a balance between solution time and solution quality compared with traditional solution methods. These frameworks are usually trained on the same type of problems and can outperform traditional solvers on these types of problems. A significant challenge faced by these ML-based frameworks is the heavy reliance on a large number of problem instances that belong to the same category for training. Nevertheless, it's worth noting that numerous datasets [5] suffer from a shortage of such isomorphic instances. This underscores the requirement for a generator capable of autonomously producing MILP instances that belong to the same category from existing instances.

Current generators used to generate problem instances from existing instances can be roughly divided into two categories, mathematically constructed and ML-based approaches. Traditional methods heavily relies on manual design and often fails to accommodate a diverse range of problems. In contrast, ML-based approaches are limited to coefficient-free SAT problems and struggle to generate intricate MILP instances with specific coefficients and constraints.

To address the limitations of current generative methods and autonomously produce high-quality MILP instances, this paper introduces MILPGen (Mixed Integer Linear Programming Instance Generator), a deep generative framework designed for large-scale MILP instances. Inspired by extant generation strategies [7, 11], the key points of MILPGen lies in two components: *Node Splitting and Merging*, and *Scalable Problem Construction*. MILPGen first adopts a bipartite graph representation combined with a random-feat strategy [2] to achieve an efficient and lossless feature embedding. Subsequently, MILPGen integrates a discriminator model to split and merge nodes. In particular, it initially decomposes the original bipartite graph into tree-like structures, simultaneously collecting training data throughout the process. Subsequently, MILPGen utilizes the discriminator model to predict the graph structure of MILP instances. Furthermore, MILPGen scales problems by concatenating various tree structures of the same category to construct scalable problems.

Experimental results on three standard MILP demonstrate that MILPGen can proficiently generate MILP problems that resemble input training problems. By producing high-fidelity data imitations, it addresses the challenge of ML-based evolutionary optimization framework [10] depending on a multitude of problem instances

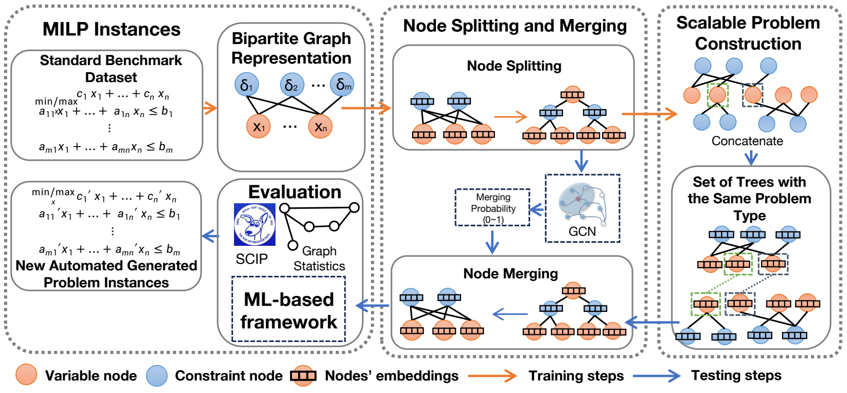
MILP Instances
Standard Benchmark Dataset
Bipartite Graph Representation
Node Splitting and Merging
Node Splitting
Scalable Problem Construction
Concatenate
Set of Trees with the Same Problem Type
Merging Probability (0~1)
GCN
Node Merging
Evaluation
SCIP
Graph Statistics
ML-based framework
New Automated Generated Problem Instances
Variable node
Constraint node
Nodes' embeddings
Training steps
Testing steps

Figure 1: An overview of MILPGen. The orange line signifies components that are active only during training, the blue line represents components that are active only during testing.

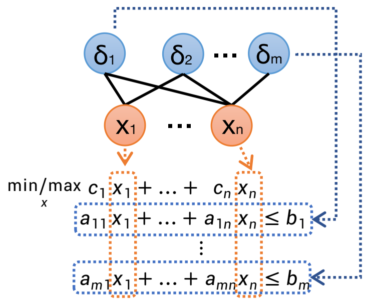

Figure 2: Bipartite graph representation of a MILP.

that belong to the same category. This paper's contributions can be summarized as follows:

(1) To the best of our knowledge, this is the first paper to propose a deep generative model designed to generate scalable MILP instances, laying the foundation for introducing large model pre-training to combinatorial optimization problems.
(2) We automate node splitting operations to attain a simplified tree-based representation of the original problem. Moreover, we leverage a discriminator model to create scalable MILP problems.
(3) We show that the instances generated by MILPGen can enhance the solution effect of the ML-based evolutionary optimization framework. The scaled-up generated instances still retain the problem's structural properties with improved solving difficulty.

2 METHODOLOGY

This section introduces the proposed MILPGen (Mixed Integer Linear Programming Instance Generator). Our generator consists of two key stages: Node Splitting and Merging (Sec. 2.1) and Scalable Problem Construction (Sec. 2.2).

2.1 Node Splitting and Merging

In this stage, MILPGen focuses on learning the distribution over the BGR of MILP problems. Given the complex structure of bipartite graphs of MILPs, generating the entire graph in one step is unfeasible. Inspired by G2SAT [11], we adopt a step-by-step learning and generation approach. Specifically, we split the original bipartite graph into tree structures and employ the discriminator model to predict which pair of nodes should be merged to reconstruct the graph structure. Let n be the number of nodes in the bipartite graph, and m the number of edges. If MILPGen performs x node splitting and merging operations to generate a new problem, the reconstruction percentage is defined as $\frac{x}{m-n}$.

2.1.1 Node Splitting. MILPGen first transforms the original bipartite graph into a tree-like structure, and then identifies a node x in graph G with the highest degree and splits it into two new nodes, p and q. In particular, if node x has a degree d in G, the new node p will inherit $d-1$ edges from x (selected randomly) and node q will inherit a single edge from x, which generates a new graph G'. For each graph G_i, let x_i be the node to be split, Node Splitting generates a new graph G_{i-1} and two new nodes p_i and q_i.Then another node r_i is selected from the same partition of G where p_i and q_i belong to but is distinct from both. This process generates training data in the form of positive examples (G_{i-1}, p_i, q_i) and negative examples (G_{i-1}, p_i, r_i). The tuples (G_{i-1}, p_i, q_i, r_i) are stored in a training dataset D for subsequent usage.

MILPGen needs to maintain the node feature information. To achieve this goal, the one-hot vectors A_i, B_j and the coefficients u_i, v_j, which represent the variables in the optimization objectives and constraint right-hand sides (RHS), remain the same as before splitting. In addition, the random feature ξ is regenerated, and the degree d is updated as $d_{p_i} = d_{x_i} - 1$ and $d_{q_i} = 1$ since p_i inherits $d_{x_i} - 1$ edges from the split node x_i, and q_i only gets one newly created edge.

2.1.2 Discriminator Model Design. The discriminator employs a half-convolution GCN optimized for bipartite graphs, augmented with a Multilayer Perceptron. This model begins with a three-layer semi-convolutional GCN that operates with identical parameters across layers. This GCN architecture is responsible for obtaining the node embeddings for each node in the bipartite graph. Then the model assesses the pair of nodes under consideration for merging. The embeddings of these two nodes are concatenated and fed into an MLP, which then outputs a value between 0 and 1—indicative of the confidence level for merging the nodes.

2.1.3 Node Merging. Node Merging in MILPGen transforms tree structures back into complex bipartite graphs. Specifically, consider an input graph H_0, through n iterations of node merging operations, the algorithm yields H_n. For each graph H_i, MILPGen randomly samples K pairs of nodes $\{(u_k, v_k)\}_{k=1}^{K}$ and employs the above discriminator model to parallelly compute the likelihood of merging each pair in the context of H_i. Then, it selects the node pair (u_o, v_o) with the highest merging probability to form the subsequent graph H_{i+1}.

To maintain the bipartite nature of the resulting graph, it is essential that the nodes in each potential pair belong to the same half of the original bipartite graph. The selection probability for merging nodes from either the variable or constraint partition is determined based on their respective splitting frequencies x and y during the Node Splitting phase. Specifically, the probability of merging a decision-variable node is $\frac{x}{x+y}$, and that for a constraint-equation node is $\frac{y}{x+y}$.

Converse to the Node Splitting stage, the feature set A_i or B_j of the new merged node is randomly selected from one of the two nodes being merged. The degree of the new node is the sum of the degrees of the two original nodes. The coefficients u_i and v_j relating to optimization targets and constraint Right-Hand Sides (RHS) are averaged, and a new random feature ξ is generated.

2.2 Scalable Problem Construction

2.2.1 Node Embeddings Computation. During the problem scaling phase, MILPGen initially utilizes the GCN component of the discriminator model for inference. This enables the computation of node embeddings for each node in the two trees marked for merging. However, due to computational constraints, it's impractical to compare the node embeddings of every node between the two trees. To address this, a hyperparameter E is introduced to select E nodes from each tree for comparison. The classical cosine similarity metric is employed to compare the embeddings. With the computational complexity being $O(E^2)$, this algorithm conducts pairwise comparisons among the embeddings of all selected nodes. Subsequently, it identifies the pair with the highest similarity score as the optimal candidates for merging.

2.2.2 Tree Merging. The key idea of tree merging is to amalgamate multiple trees into a single and larger tree, each derived from the same type of MILP problem instances and subjected to node-splitting processes. This consolidated tree is then subjected to node merging to generate a more complex problem instance than those used during training. Given a set of trees denoted as $\mathcal{T}$, the process begins by randomly sampling a base tree T_0 from $\mathcal{T}$. Subsequent operations involve iteratively merging T_{i-1} with a randomly selected tree t from $\mathcal{T}$, resulting in a new tree T_i. Finally, T_m becomes the enlarged tree after several iterations.

Merging two trees involves calculating node embeddings to identify the pair of nodes with the highest similarity for merging, similar to the node merging process. However, in addition to maintaining the node feature information of the newly merged nodes, similar to node merging, the process also keeps track of the number of node-splitting occurrences associated with the constraint equation nodes and decision variable nodes within the tree.

3 EXPERIMENTS

We evaluate the performance of MILPGen on three combinatorial optimization benchmark problems, including Maximum Independent Set (MIS) [8], Combinatorial Auction (CA) [3], and Minimum Vertex Cover (MVC) [4]. First, we study the performance of the generation of MILPGen with the same scale problem generation, assessing its role as a data augmentation tool (Sec. 3.1). Furthermore, we study the scalability of MILPGen to evaluate its remarkable imitation and generation capabilities (Sec. 3.2). The code of MILPGen is available at https://github.com/thuiar/MILPGen.

In the following experiments, we compare the generation results of MILPGen with two baselines. The first baseline, referred to as 'Bowly'[1], is a heuristic MILP instance generator. In our experiments, we set its all controllable parameter to match the corresponding features of the MILP instances in our training dataset. The second baseline, named 'Random', is derived by randomizing the output of the discriminator in the Node Merging step of MILPGen, and the reconstruction percentage is set to 100%.

3.1 Overall Performance of Generation

To study the MILPGen performance of generation, we examine whether the generated MILP instances preserve the properties of the input training MILP instances through an analysis of solver performance (Sec. 3.1.1). Subsequently, we integrate MILPGen into the ML-based evolutionary optimization framework [10] to study its potential as a data augmentation technique (Section 3.1.2). This evaluation examines the quality of the generated MILP instances in predicting feasible solutions and their capacity to improve optimization results.

3.1.1 MILP Solver Performance. We compared the solver performance of MILP problems generated using different methods with that of the original training data instances. Solver performance results are presented in Table 1. Our experimental findings indicate that the problems generated by MILPGen exhibit a level of solving complexity similar to that of the original problems, contrasting with the relatively trivial problems generated by the Random and Bowly methods.

3.1.2 Data Augmentation for ML-based Evolutionary Optimization framework. To validate MILPGen's effectiveness in tackling the challenge posed by ML-based optimization frameworks that depend on a substantial number of instances of problems for training data, we conducted experiments on the state-of-the-art GNN&GBDT-based evolutionary optimization framework [10]. We make comparisons between only using the training dataset and using both the training dataset and the 20 newly generated MILP problems (use model trained by the input dataset) as training data for the ML-based evolutionary optimization framework. The evaluation focuses on the framework's solving performance for MILP problems and is shown in Table 2. The results indicate that the MILP problems newly generated by MILPGen enhance the predictive ability and solution effectiveness of the ML-based optimization framework.

3.2 Scalability of MILPGen

To further study MILPGen's capability to generate scalable MILP instances, we evaluate the large-scale problems derived through

Table 1: Comparison of the optimality gap. The interval represents the result of 20 generated instances by specifying different random seeds. The percentage after MILPGen denoted *reconstruction percentage*. For SCIPopt solver, the time limit is set to 1200s; for Gurobi solver, the time limit is set to 600s. "Origin" represents the instance used as training data.

Optimality Gap	Origin	MILPGen-25%	MILPGen-50%	MILPGen-100%	Random	Bowly
MIS-SCIP	9.64%	7.06% ~ 9.93%	7.78% ~ 11.31%	4.64% ~ 7.33%	solved in 1.29 ~ 150.05s	solved in 0.14 ~ 0.41s
MIS-Gurobi	6.93%	5.03% ~ 6.25%	6.14% ~ 7.00%	2.64% ~ 3.87%	solved in 0.05 ~ 1.03s	solved in 0.02 ~ 0.90s
CA-SCIP	9.52%	9.30% ~ 13.12%	9.76% ~ 12.69%	6.82% ~ 9.53%	solved in 3.38 ~ 126.51s	solved in 0.09 ~ 10.49s
CA-Gurobi	7.17%	7.73% ~ 9.84%	8.22% ~ 10.45%	5.53% ~ 6.90%	solved in 0.56 ~ 9.58s	solved in 0.02 ~ 5.01s
MVC-SCIP	10.04%	7.06% ~ 9.56%	5.26% ~ 7.81%	solved in 258.78s ~ 0.49%	solved in 0.42 ~ 0.95s	solved in 0.13 ~ 2.44s
MVC-Gurobi	6.01%	4.32% ~ 5.10%	3.43% ~ 4.15%	solved in 5.14s ~ 0.05%	solved in 0.55 ~ 1.08s	solved in 0.01 ~ 0.58s

Table 2: Comparison of the final optimized solution within a fixed time trained by generated MILP instances and the training dataset. The percentage after MILPGen denoted *reconstruction percentage*. The scale-limited versions of SCIP which limit the variable proportion α is set to 30%, the time limit of the solver framework is 50s.

	MIS		CA		MVC	
	Original	Augmented	Original	Augmented	Original	Augmented
MILPGen-25%	2092.99	2122.68 ↑	1452.95	1460.78 ↑	2918.80	2870.89 ↑
MILPGen-50%		2141.37 ↑		1503.47 ↑		2837.94 ↑
MILPGen-100%		2080.6		1444.40		2911.74 ↑
Bowly		1797.2		1069.7		3008.7

Table 3: Comparison of the optimality gap between the scalable generated MILP problems and the training dataset. For SCIPopt solver, the time limit is set to 1200s; for Gurobi solver, the time limit is set to 600s.

	Var Num	Constr Num	Gap-SCIP	Gap-Gurobi
MIS-input	10000	30000	9.64%	6.93%
MIS-1x	10001	30000	5.78%	2.65%
MIS-2x	19999	60000	7.26%	4.21%
MIS-4x	39998	120000	25.13%	11.21%
MIS-8x	79994	240000	25.69%	11.42%
CA-input	10000	20000	9.52%	7.17%
CA-1x	10000	20000	7.32%	6.34%
CA-2x	19999	40000	226.65%	8.00%
CA-4x	39998	80000	224.81%	185.24%
CA-8x	79993	160000	229.21%	188.08%
MVC-input	10000	30000	10.04%	6.01%
MVC-1x	10002	30000	1.91%	solved in 82.83s
MVC-2x	20000	60000	1.16%	0.50%
MVC-4x	40000	120000	13.55%	0.77%
MVC-8x	79994	240000	14.08%	4.81%

extrapolation from the small-scale training dataset. All the results are shown in Table 3. The solving performance measured by SCIP and Gurobi, suggests that the generated scaling instances closely match those of the large-scale training dataset. This confirms the remarkable imitation and generation capabilities of MILPGen.

4 CONCLUSION

This study presents MILPGen, a pioneering deep generative model tailored for MILP problems. Leveraging advanced techniques like a random-feat policy for bipartite graph representation, and node operations in an expanded feature space, MILPGen effectively addresses the shortcomings of prior models, such as limited representation in large-scale MILP problem generation, and oversimplified problem structures. Currently focused on single-objective, linear, and static problems, future work will extend to more complex scenarios including multi-objective, nonlinear, and dynamic problems.

REFERENCES

[1] Simon Bowly, Kate Smith-Miles, Davaatseren Baatar, and Hans D. Mittelmann. 2020. Generation techniques for linear programming instances with controllable properties. *Math. Program. Comput.* 12, 3 (2020), 389–415. https://doi.org/10.1007/S12532-019-00170-6

[2] Ziang Chen, Jialin Liu, Xinshang Wang, and Wotao Yin. 2023. On Representing Mixed-Integer Linear Programs by Graph Neural Networks. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/pdf?id=4gc3MGZra1d

[3] Sven De Vries and Rakesh V Vohra. 2003. Combinatorial auctions: A survey. *INFORMS Journal on computing* 15, 3 (2003), 284–309.

[4] Irit Dinur and Samuel Safra. 2005. On the hardness of approximating minimum vertex cover. *Annals of mathematics* (2005), 439–485.

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

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

[7] Zachary Steever, Chase Murray, Junsong Yuan, Mark Karwan, and Marco Lübbecke. 2022. An image-based approach to detecting structural similarity among mixed integer programs. *INFORMS Journal on Computing* 34, 4 (2022), 1849–1870.

[8] Robert Endre Tarjan and Anthony E Trojanowski. 1977. Finding a maximum independent set. *SIAM J. Comput.* 6, 3 (1977), 537–546.

[9] Laurence A Wolsey. 2007. Mixed integer programming. *Wiley Encyclopedia of Computer Science and Engineering* (2007), 1–10.

[10] Huigen Ye, Hua Xu, Hongyan Wang, Chengming Wang, and Yu Jiang. 2023. GNN&GBDT-guided fast optimizing framework for large-scale integer programming. In *International Conference on Machine Learning*. PMLR, 39864–39878.

[11] Jiaxuan You, Haoze Wu, Clark Barrett, Raghuram Ramanujan, and Jure Leskovec. 2019. G2SAT: Learning to generate SAT formulas. *Advances in neural information processing systems* 32 (2019).