MILPBench: A Large-scale Benchmark Test Suite for Mixed Integer Linear Programming Problems

Huigen Ye
Tsinghua University
Beijing, China

Yaoyang Cheng
Hong Kong Baptist University
Hongkong, China

Hua Xu*
Tsinghua University
Beijing, China

Zhiguang Cao
Singapore Management University
Singapore, Singapore

Hanzhang Qin
National University of Singapore
Singapore, Singapore

Abstract

Mixed-integer linear programming (MILP) is a cornerstone of optimization with applications across numerous domains. However, the development and evaluation of MILP-solving algorithms are hindered by existing benchmark datasets, which are often limited in scale, lack diversity, and are poorly structured, making them inadequate for systematic testing across different solving approaches, especially for machine learning (ML)-based methods. To address these issues, we introduce MILPBench, a large-scale benchmark suite comprising 100,000 MILP instances organized into 60 well-categorized classes. Using structural properties and embedding similarity metrics, we developed a novel classification framework to ensure both intra-class homogeneity and inter-class diversity. In addition to the dataset, MILPBench includes a comprehensive baseline library featuring 15 mainstream solving methods, spanning traditional solvers, heuristic algorithms, and ML-based approaches. This design enables rigorous and standardized evaluation of MILP-solving algorithms under diverse conditions. Extensive benchmarking demonstrates the utility of MILPBench as a scalable and versatile testbed for advancing MILP research, fostering innovation in solver development, and bridging the gap between optimization and machine learning.

CCS Concepts

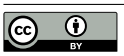
• **Theory of computation → Mixed discrete-continuous optimization**; *Integer programming*.

Keywords

Benchmark Dataset, Mixed Integer Linear Programming, Machine Learning, Branch-and-bound, Heuristic

ACM Reference Format:
Huigen Ye, Yaoyang Cheng, Hua Xu, Zhiguang Cao, and Hanzhang Qin. 2025. MILPBench: A Large-scale Benchmark Test Suite for Mixed Integer Linear Programming Problems. In *Genetic and Evolutionary Computation Conference (GECCO '25), July 14–18, 2025, Malaga, Spain.* ACM, New York, NY, USA, 27 pages. https://doi.org/10.1145/3712256.3726324

*Hua Xu is the corresponding author (xuhua@tsinghua.edu.cn).



GECCO '25, July 14–18, 2025, Malaga, Spain

ACM ISBN 979-8-4007-1465-8/2025/07
https://doi.org/10.1145/3712256.3726324

1 Introduction

Mixed-integer linear programming (MILP) is a cornerstone of optimization, widely applied in fields such as routing, scheduling, network design, and resource allocation [7, 33, 36, 40, 46, 48, 51, 54, 56, 59, 67, 83]. MILP involves optimizing a linear objective function under constraints with both integer and continuous variables [75, 76], providing solutions to complex combinatorial problems. Given its ubiquity and importance, advancing algorithmic efficiency for solving MILPs is critical for addressing real-world challenges.

Traditional MILP solvers can broadly be categorized into **exact algorithms** and **heuristic approaches** [82]. Exact methods, such as branch-and-bound [5, 45, 81], include techniques like pseudo-cost branching [16, 55], strong branching [4, 29], and hybrid branching [3, 73]. Heuristic methods, on the other hand, focus on approximate solutions, leveraging techniques such as feasibility pumps [17, 32], evolutionary algorithms [60, 68], and large neighborhood search [47, 63]. While these methods are highly effective, they often struggle in scenarios requiring the repeated solution of homogeneous MILP instances (i.e., problems with similar combinatorial structures). These scenarios expose traditional solvers to the challenge of **cold starts**, as they are unable to exploit accumulated solving knowledge to accelerate subsequent computations [62, 82].

In recent years, researchers have explored **machine learning (ML)** to improve MILP-solving methods. Inspired by the bipartite graph representation of MILP problems [35], neural networks— particularly graph neural networks (GNNs) [69]—have been employed to enhance branch-and-bound decisions [21, 41, 42], neighborhood selection [70, 71, 77], and heuristic initialization [30, 79, 80]. These methods promise significant speedups and improved scalability. However, despite their potential, progress has been hindered by the lack of **large-scale, systematically categorized benchmark datasets**. Existing datasets, such as MIPLIB [37], are limited in size, diversity, and structural organization. Moreover, none of these datasets provide a **baseline library** for standardized performance comparison across traditional solvers, heuristic algorithms, and ML-based approaches. This gap has slowed MILP solver development and impeded rigorous comparisons under diverse conditions.

To address these challenges, we introduce **MILPBench**, the largest MILP benchmark dataset to date, comprising 100,000 instances across 60 systematically categorized classes. MILPBench provides a diverse and comprehensive testbed for evaluating MILP-solving algorithms. By leveraging structural similarity metrics and

GNN-based embeddings, we ensure that MILPBench achieves intra-class homogeneity and inter-class diversity, enabling robust evaluations. In addition, MILPBench includes a baseline library featuring 15 mainstream solving methods, spanning traditional solvers, heuristic algorithms, and ML-based approaches, making it the first dataset to offer such a complete evaluation framework. Experimental results demonstrate that MILPBench enables detailed performance comparisons and reveals inconsistencies in some algorithm claims, further underscoring its value as a standardized and scalable testbed for MILP research.

In summary, MILPBench offers the following contributions:

(1) **Largest-Scale Dataset**: MILPBench is the largest MILP benchmark dataset to date, comprising 100,000 instances across 60 heterogeneous classes.
(2) **Comprehensive Baseline Library**: MILPBench provides a standardized evaluation framework with 15 mainstream solving methods, supporting traditional solvers, heuristic algorithms, and ML-based approaches alike.
(3) **Scalability and Adaptability**: MILPBench offers a diverse test suite with plans for regular updates, ensuring its relevance for future research and applications.

By addressing the limitations of existing datasets, MILPBench establishes a new standard for MILP benchmarking, facilitating algorithm development and fostering innovation across both traditional and ML-based approaches. Our MILPBench is open-source and accessible at: https://github.com/thuiar/MILPBench.

2 Related Work

2.1 Mixed Integer Linear Programming Problem

Mixed Integer Linear Programming (MILP) is a foundational class of combinatorial optimization problems with extensive applications in diverse fields, including logistics, robotics, and operations research. Formally, a MILP problem can be defined as follows [75, 76]:

$$\min_x \ c^T x, \text{subject to } Ax \le b, l \le x \le u, x_i \in \mathbb{Z}, i \in \mathbb{I}, \tag{1}$$

where x represents the decision variables, with dimension denoted by $n \in \mathbb{Z}$, and $l, u, c \in \mathbb{R}^n$ correspond to the lower bounds, upper bounds, and coefficient values of the variables, respectively. The matrix $A \in \mathbb{R}^{m \times n}$ and the vector $b \in \mathbb{R}^m$ define the linear constraints of the problem. The set $\mathbb{I} \subseteq \{1, 2, \ldots, n\}$ denotes the indices of variables that are constrained to be integers.

2.2 Existing Methods for Solving MILP

Solving MILP problems effectively has been a long-standing challenge, leading to the development of various algorithmic approaches. Traditional methods, including branch-and-bound, branch-and-cut, and cutting-plane algorithms, guarantee optimal solutions but often suffer from scalability issues when dealing with large problem instances. In contrast, heuristic approaches, such as genetic algorithms, simulated annealing, and large neighborhood search, offer faster solutions by sacrificing optimality. The availability of well-organized test datasets is crucial for understanding the performance of these methods across diverse scenarios, enabling researchers to identify their strengths and weaknesses and make informed choices for specific problem settings.

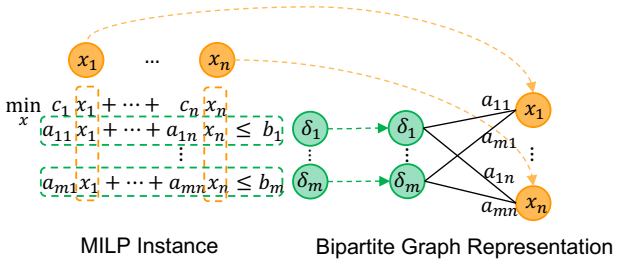


Figure 1: Convert MILP into bipartite graph.

In recent years, machine learning (ML) has emerged as a promising tool for solving MILP problems, with most frameworks relying on Graph Neural Networks (GNNs) to exploit the structural properties of MILP. As shown in Figure 1, Gasse et al. proposed a bipartite graph representation of MILP problems [35], which achieves a lossless transformation of MILP instances into graph structures. Such graph representations are well-suited for neural embedding networks [62], enabling ML-based approaches to learn optimization strategies directly from problem instances. These methods have demonstrated potential in bridging the gap between traditional solvers and modern data-driven techniques, offering new opportunities for improving solver efficiency and adaptability.

However, the lack of well-structured and carefully categorized datasets poses a significant challenge for both traditional and ML-based approaches. Homogeneous datasets are particularly important for ML-based methods, as heterogeneous problems can lead to poor model generalization. Previous studies [23, 64] have shown that GNNs often suffer from over-smoothing and limited expressiveness, which hinder their ability to handle diverse problem distributions. Furthermore, the generalization of large models with strong out-of-distribution capabilities also benefits from fine-grained category labels, which can help capture the structural differences between problem types during pretraining. These findings highlight the necessity of datasets with intra-class homogeneity and inter-class diversity to support both current ML techniques and the future development of large-scale optimization models.

2.3 Related Benchmark Datasets

The development and evaluation of mixed-integer linear programming (MILP)-solving methods, particularly ML-based approaches, are significantly hindered by the limitations of existing datasets. Widely used datasets such as MIPLIB [53] and Coral [26] provide general MILP instances but lack the structure and diversity needed for systematic evaluation. These datasets often fail to cover a wide range of problem scales and structural variations, making them inadequate for testing solvers designed for large-scale or domain-specific applications. Moreover, they are not tailored for ML research, as they lack fine-grained categorizations critical for training and evaluating ML models. Distributional MIPLIB [49] has made some attempts to address these issues by introducing MILP distributions from 13 domains, classified into different hardness levels. However, similar to other datasets, it is limited by its small number of problem categories (13 classes), small-scale problem instances (variables and constraints at most in the tens of thousands), reliance

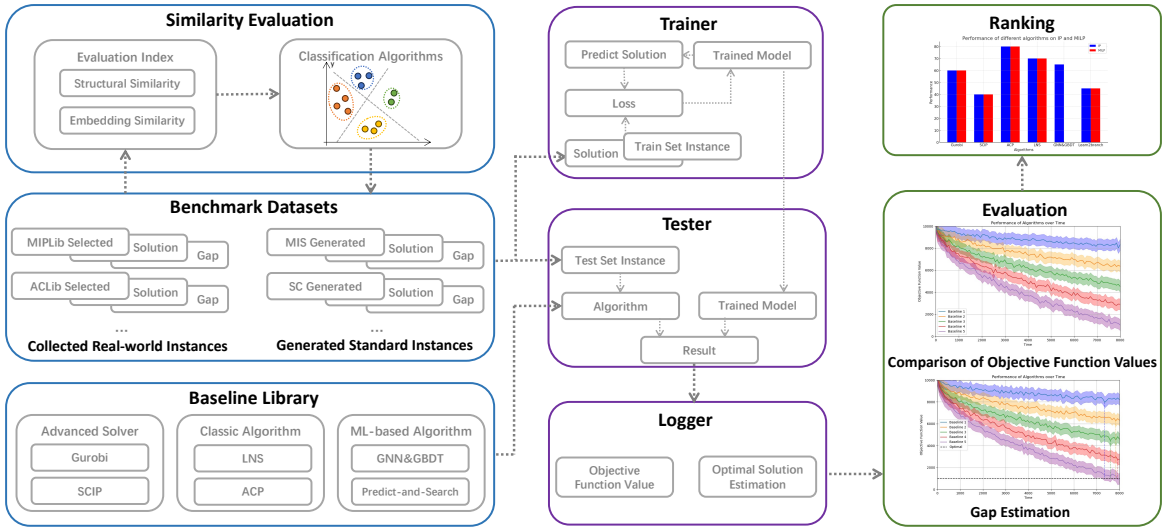


Figure 2: An overview of *MILPBench*. The *Benchmark Datasets* component organizes collected MILP instances into 60 well-categorized classes, guided by the *Similarity Evaluation* component, which uses structural and embedding-based metrics to ensure dataset quality. The *Baseline Library* component enables standardized training, testing, and logging of solving algorithms, with performance evaluated based on objective function values, gap metrics, and runtime efficiency. Results are aggregated into a leaderboard to facilitate systematic comparisons across algorithms.

on synthetic problem generation, and lack of real-world problem curation or an accompanying baseline library.

To address these limitations, we introduce MILPBench, a large-scale benchmark tailored to the needs of both traditional and ML-based MILP-solving methods. Unlike existing datasets, MILPBench offers extensive structural diversity, fine-grained categorizations, and realistic problem scales, paving the way for more rigorous and generalizable evaluation.

3 Proposed MILPBench

We propose MILPBench, a large-scale benchmark suite designed to address the limitations of existing datasets and enable systematic evaluation of MILP-solving algorithms across diverse conditions. MILPBench provides a scalable and versatile platform for evaluating a wide range of solving techniques, including traditional solvers, heuristic algorithms, and ML-based approaches. By offering a comprehensive dataset and baseline library, MILPBench bridges the gap between academic benchmarks and real-world applications, supporting the advancement of MILP research and fostering innovation in solver development.

As illustrated in Figure 2, MILPBench comprises three core components: Similarity Evaluation, Benchmark Datasets, and Baseline Library. The Benchmark Datasets include 100,000 MILP instances organized into 60 well-categorized classes, ensuring both intra-class homogeneity and inter-class diversity. This is achieved through a novel classification framework that integrates structural properties and embedding similarity metrics to analyze and categorize problem instances. The Similarity Evaluation component ensures that instances with low similarity scores are reclassified, maintaining the quality and consistency of the dataset. To enable rigorous and standardized testing, the Baseline Library provides 15 mainstream solving methods spanning traditional solvers, heuristic algorithms, and ML-based approaches. These methods are evaluated using a standardized "*Trainer-Tester-Logger*" pipeline, which measures solver performance based on objective function values, gap estimates, and runtime efficiency under fixed wall-clock times. Results are aggregated and ranked to create a comprehensive leaderboard, offering insights into the strengths and weaknesses of different solving techniques. This robust evaluation framework ensures that MILPBench serves as a scalable and practical testbed for advancing MILP research.

3.1 Benchmark Datasets

To enable the systematic evaluation of MILP-solving algorithms, we constructed a comprehensive benchmark dataset that consists of two main components: MILP Instances and their corresponding Reference Solutions and Gap Estimates. The dataset is designed to meet the diverse needs of solving techniques, ranging from traditional solvers to heuristic algorithms and machine learning (ML)-based approaches. By combining real-world problems and standardized mathematical formulations, the dataset ensures broad coverage across different problem types, scales, and complexities.

The MILP instances in our dataset originate from two primary sources. The first source includes real-world problems carefully curated from a wide range of open-source datasets and academic studies. These include mainstream datasets such as MIPLIB [53], AClib [50], Regions200 [57], MIRPLib [65], and COR@L [26]. Additionally, we incorporated problems from domain-specific studies, such as robustness verification of neural networks [62], lot-sizing polytopes [13], network design [11], combinatorial auctions [28],

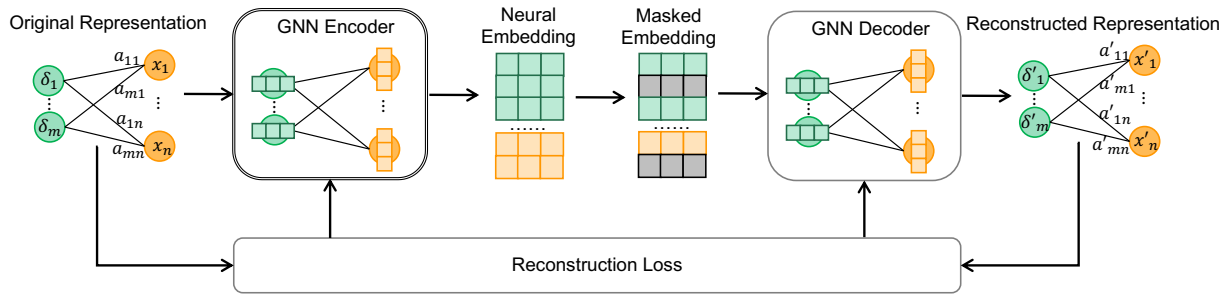


Figure 3: The overview of graph self-supervised learning. For the bipartite graph representation of MILP, we use the encoder-decoder structure for reconstruction, and the loss is computed by comparing the output with the features of the nodes in the original representation.

and 0-1 knapsack problems [14]. Competitions such as the ML4CO challenge at NeurIPS 2021 [34] and the Reoptimization Competition 2023 [19] were also valuable sources for real-world MILP problems. These curated instances reflect a wide variety of application domains, but many of them are small in scale and fail to represent the complexity of large-scale scenarios. Specific details about these datasets can be found in Appendix A.1.

To address the limitations of small-scale real-world problems, the second source of our MILP instances consists of problems generated from standardized mathematical formulations. Specifically, we constructed instances based on nine canonical MILP problem classes, including Maximum Independent Set [72], Minimum Vertex Covering [31], Set Covering [20], Mixed Integer Knapsack Set [12], and Capacitated Facility Location [8]. For each problem type, we generated instances at three levels of difficulty—easy, medium, and hard—corresponding to tens of thousands, hundreds of thousands, and millions of decision variables, respectively. This approach ensures that the dataset includes ultra-large-scale problems, bridging the gap between academic benchmarks and real-world applications. Further details of the process are provided in Appendix A.3.

Once the MILP instances were collected, we applied the Similarity Evaluation Metrics described in Section 3.2.1 to assess the homogeneity of the dataset. Instances with significant internal variability were re-screened and re-classified using the Classification Algorithm outlined in Section 3.2.2. This process ensures that the final dataset achieves a high degree of intra-class homogeneity and inter-class diversity, facilitating robust and systematic evaluation of solving techniques.

For each MILP instance, we also provided high-quality reference solutions and gap estimates to support benchmarking. To obtain these, we first employed the state-of-the-art solver Gurobi (version 11.0.1) to solve the problems for a fixed duration (e.g., 8 hours), during which most problems achieved optimal solutions. These solutions, along with the problem instances, were saved in pickle files for seamless integration into experiments. However, for larger-scale problems where Gurobi could not compute optimal solutions within the allocated time, we utilized an Adaptive Constraints Partition (ACP)-based strategy [78] to iteratively improve the solutions. Starting with the solution $\overline{x}$ and gap estimate $\overline{g}$ obtained from Gurobi, we computed the improved solution x^* and its corresponding gap g^* using the following formula:

$$g^* = \frac{x^* - (1 - \overline{g})\overline{x}}{x^*}. \tag{2}$$

This approach ensures that high-quality reference solutions are available even for large-scale problems. Detailed descriptions of the ACP-based strategy are presented in Appendix B. Finally, the dataset was partitioned into training and testing subsets to support algorithm development and evaluation. Specific details of the partitioning scheme are provided in Appendix A.4.

3.2 Similarity Evaluation

To ensure the quality and utility of MILPBench, we introduce a novel Similarity Evaluation framework that addresses the limitations of existing datasets. By integrating structural properties and embedding similarity metrics, this framework ensures that the 100,000 MILP instances in MILPBench are organized into 60 classes with both intra-class homogeneity and inter-class diversity. The framework consists of two key components: **Similarity Evaluation Metrics**, which assess the structural and embedding-based characteristics of MILP instances to quantify their similarity, and a **Classification Algorithm**, which leverages these metrics to systematically categorize and refine the dataset. This categorization process mitigates challenges such as over-smoothing and poor generalization in GNN-based methods [64], while also enhancing the applicability of traditional and heuristic solvers through a more systematic evaluation setup.

3.2.1 Similarity Evaluation Metrics. To quantify the similarity between MILP instances, we employ a two-pronged approach that combines structural embedding and neural embedding. These methods capture complementary aspects of MILP problem structures, ensuring a comprehensive evaluation of similarity.

Structural Embedding: Each MILP instance is represented as a 10-dimensional feature vector, summarizing key structural properties derived from its mathematical formulation and bipartite graph representation. Table 1 lists the metrics used, including features such as the fraction of non-zero entries in the coefficient matrix (`coef_dens`), the mean and standard deviation of variable and constraint vertex degrees (`var_degree_mean`, `cons_degree_std`), and the modularity of the graph. These metrics are selected to provide

Table 1: Structural Metrics for Similarity Evaluation.

Metric Name	Description
coef_dens	Fraction of non-zero entries in the coefficient matrix.
cons_degree_mean	Mean degree of constraint vertices in the bipartite graph.
cons_degree_std	Standard deviation of degrees of constraint vertices.
var_degree_mean	Mean degree of variable vertices in the bipartite graph.
var_degree_std	Standard deviation of degrees of variable vertices.
lhs_mean	Mean of non-zero entries in the coefficient matrix (LHS).
lhs_std	Standard deviation of non-zero entries in the coefficient matrix (LHS).
rhs_mean	Mean of right-hand-side (RHS) values in constraints.
rhs_std	Standard deviation of right-hand-side (RHS) values in constraints.
modularity	Modularity of the bipartite graph, measuring community structure.

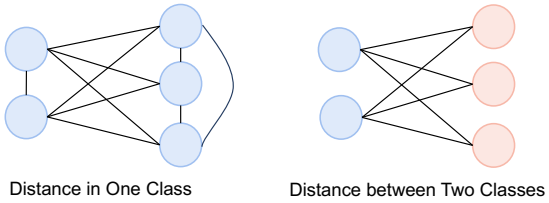


Figure 4: The Calculation of Similarity Score.

a holistic view of the MILP instance's structural characteristics. To evaluate the similarity of MILP instances based on these features, we compute the Jensen-Shannon (JS) divergence for each metric. Let JS_i denote the JS divergence for the i^{th} metric. The similarity score for the i^{th} metric is defined as:

$$\text{score}_i = \frac{\max(JS) - JS_i}{\max(JS) - \min(JS)}. \tag{3}$$

The overall structural similarity score is calculated as the average of all individual metric scores:

$$\text{score} = \frac{1}{10} \sum_{i=1}^{10} \text{score}_i. \tag{4}$$

This scoring system ensures that instances with similar structural properties are grouped together, facilitating systematic evaluation and classification.

Neural Embedding: To model intricate relationships such as local connectivity and coefficient interactions that are not fully captured by structural embedding, we adopt a graph self-supervised learning paradigm inspired by Graph Autoencoders [52]. MILP instances are represented as bipartite graphs and encoded using a Graph Convolutional Neural Network (GNN) encoder. As illustrated in Figure 3, the encoder maps the graph into an intermediate neural embedding, while the decoder reconstructs the original graph. The reconstruction loss, computed as the discrepancy between the reconstructed graph and the original input, ensures that the embeddings capture meaningful structural patterns. To further enhance the embeddings, we mask parts of the graph (both node features and edges) during training, forcing the model to predict the masked information. Formally, let $\mathcal{G} = (\mathcal{V}, X, E)$ represent a bipartite graph, where $\mathcal{V}$, X, and E are the nodes, node features, and edges, respectively. During masking, a subset of nodes $\mathcal{V}_{\text{mask}}$ and edges $\mathcal{E}_{\text{mask}}$ are selected with probabilities p_v and p_e, and their features are replaced with a special [mask] token. This process ensures that the learned embeddings encode both global and local structural features, making them robust for downstream similarity evaluations.

Similarity Measure: To evaluate the similarity of a set of MILP instances $\mathcal{I}$, we calculate the average pairwise Euclidean distance between their embeddings:

$$\text{Embedding Distance} = \frac{\sum_{\mathcal{I}_i, \mathcal{I}_j \in \mathcal{I}, i \neq j} \text{Distance}(\mathcal{I}_i, \mathcal{I}_j)}{|\mathcal{I}|(|\mathcal{I}| - 1)}, \tag{5}$$

where $\text{Distance}(\mathcal{I}_i, \mathcal{I}_j)$ denotes the Euclidean distance between the embeddings of instances $\mathcal{I}_i$ and $\mathcal{I}_j$. A lower embedding distance indicates higher similarity among instances, as illustrated in Figure 3. By integrating structural and neural embeddings, this hybrid approach provides a robust and scalable metric for evaluating MILP instance similarity.

3.2.2 Classification Algorithm. For datasets with low internal similarity, we propose a novel strategy to re-screen and re-classify the data, ensuring greater homogeneity within each subset. This process begins by applying the neural embedding method described in Section 3.2.1 to generate embeddings for all problem instances. These embeddings, which capture both global and local structural features, are then used as inputs to a spectral clustering algorithm. Spectral clustering constructs a similarity graph, where nodes represent problem instances and edge weights encode pairwise similarities. By analyzing the graph's structure through the eigenvectors of its Laplacian matrix, spectral clustering effectively identifies groups of instances with high internal similarity, even in cases where clusters are non-convex or non-linearly separable.

After the initial clustering, we evaluate the similarity metrics within each subset to assess their internal homogeneity. Subsets with sufficiently high similarity scores are retained, while those with lower scores are further subdivided using the same spectral clustering approach. This iterative refinement process, as illustrated in the right panel of Figure 4, ensures that the final dataset is well-structured, with clearly defined categories that exhibit both intra-class homogeneity and inter-class diversity. By leveraging this classification algorithm, we provide a robust foundation for systematic evaluation and training of solving algorithms, facilitating both research advancements and practical applications.

3.3 Baseline Library

To validate the effectiveness of MILPBench, we developed a comprehensive Baseline Library consisting of 15 mainstream solving methods. These methods span traditional solvers, heuristic algorithms, and ML-based approaches, providing a broad spectrum of techniques for rigorous benchmarking and comparison. This library ensures that MILPBench can evaluate solving techniques across diverse methodologies and application scenarios.

The first category in the Baseline Library includes state-of-the-art traditional solvers, such as SCIP (version 4.3.0) [2], Gurobi (version 11.0.1) [43] and CPLEX(version 22.1.1.0) [24]. These solvers represent the leading academic and commercial optimization tools and are widely used in both research and industrial applications. We implemented these solvers using their official APIs to solve specific MILP problems. The second category consists of heuristic algorithms, including Large Neighborhood Search (LNS) [47], General Large Neighborhood Search (GLNS) [70], Least Integeal

Table 2: The distance of graph structure and neural embedding in each heterogeneous classes. The blue background colour indicates classical MILP datasets.

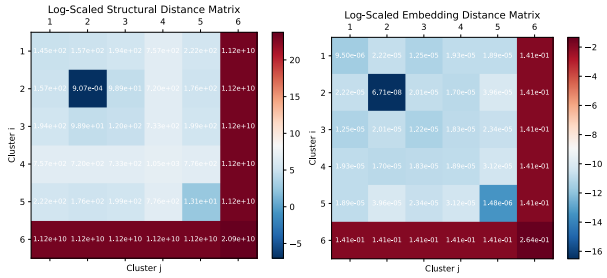
	Structure Distance	Embedding Distance
MIS_easy	0.011	1.50e-08
MVC_easy	0.010	1.24e-08
SC_easy	0.009	3.69e-08
Aclib	25.402	2.52e-06
Cut	7.122	1.47e-05
fc.data	5.867	9.68e-07
Hem_knapsack	0.566	8.75e-07
Hem_mis	0.356	4.29e-07
Hem_setcover	0.286	5.33e-07
Hem_corlat	5.956	3.34e-07
Hem_mik	226.520	9.54e-06
item_placement	0.037	7.03e-12
load_balance	0.436	6.97e-07
anonymous	222.061	8.99e-06
nn_verification	18.460	3.71e-06
vary_bounds_s1	0.001	7.08e-08
vary_bounds_s2	0.009	3.26e-08
vary_bounds_s3	0.009	1.90e-08
vary_matrix_s1	0.011	0.0
vary_matrix_rhs_bounds_s1	0.008	1.74e-08
vary_obj_s1	3.18e-06	8.39e-07
vary_obj_s2	1.22e-05	2.64e-07
vary_rhs_s1	0.007	9.83e-08
vary_rhs_s2	0.550	1.11e-05
vary_rhs_s4	0.601	1.47e-05
Transportation	148.368	1.10e-05
Coral	1056768.799	6.03e-03
ECOGCNN	468055.052	5.41e-02
MIPlib	7002615758.102	8.82e-02
Nexp	5511575859.716	2.48e-01

Heuristic (LIH) [61], Most Integeal Heuristic (MIH) [18], Relaxation Induced Neighborhood Search (RINS) [27] and Adaptive Constraint Partitioning (ACP) Optimization Framework [78]. These algorithms were reproduced based on their published pseudocode and adapted to work seamlessly with the MILP instances in our dataset. This category offers robust heuristic approaches that balance solution quality and computational efficiency. The third category comprises ML-based solving algorithms, which leverage advanced learning techniques to improve MILP-solving performance. These include Learn2Branch [35], Hybrid-Learn2Branch [41], GNN &GBDT-guided Optimization Framework [80], Neural Diving [62], Predict&Search Optimization Framework[44], and GNN-MILP [22]. These methods were retrained and tested using benchmark datasets, ensuring compatibility and fair evaluation.

Additionally, we experimented with other solving algorithms, such as Feasible Pump [32] and Simulated Annealing [1]. However, due to their inability to produce feasible solutions within reasonable computational limits for most problems, these methods were ultimately excluded from the final Baseline Library. Overall, the Baseline Library, combined with the Benchmark Datasets, provides a robust and standardized platform for evaluating and comparing the performance of diverse MILP-solving techniques.

4 Experiments

To validate the effectiveness of MILPBench, we conducted a series of experiments designed to highlight its advantages over traditional MILP datasets, such as MIPLib, Coral, and Nexp. These experiments systematically demonstrate MILPBench's ability to provide more homogeneous problem subsets, its robust classification algorithm, and its utility for benchmarking optimization algorithms. Specifically, we first compare the internal similarity of problem instances



(a) Structure distance matrix. (b) Embedding distance matrix.

Figure 5: Structure and embedding distance within classes.

in MILPBench against traditional datasets (Section 4.1), showcasing its superior ability to group structurally and semantically similar instances. Next, we analyze the distance matrix between problem categories (Section 4.2), validating the effectiveness of the classification method. Finally, we evaluate the performance of various benchmark algorithms on categorized problem subsets (Section 4.3), exposing potential weaknesses in state-of-the-art methods and demonstrating MILPBench's value in advancing the field.

4.1 Dataset Analysis

To validate the benefits of MILPBench over traditional MILP datasets such as MIPLib, Coral, and Nexp in evaluating ML-based optimization algorithms, we conducted a comparative analysis focusing on problem similarity. The experimental configurations are provided in Appendix C.2 for reproducibility.

The results, detailed in Table 2, reveal a striking contrast in problem similarity. For graph structural embedding, traditional datasets like MIPLib exhibited remarkably high variability, with pairwise distances between problem instances often exceeding hundreds of millions of times those observed in MILPBench. Similarly, the neural embedding distances in traditional datasets were found to be tens of thousands of times greater than those in MILPBench. These results demonstrate that MILPBench achieves significantly lower variability between instances, both in terms of structural and neural embeddings. This reduced variability directly translates to enhanced homogeneity within MILPBench. By providing problem instances that are more structurally and semantically similar, MILPBench enables more systematic evaluation of ML-based optimization algorithms. This is in stark contrast to traditional datasets, where the wide variability among instances often leads to inconsistent and less interpretable evaluation outcomes. MILPBench addresses this challenge by offering a well-curated dataset designed to promote consistency and comparability, ultimately advancing the development and benchmarking of optimization algorithms.

4.2 Dataset Reclassification

In some datasets within the collection, particularly those derived from open-source sources, problem instances generated from diverse scenarios are often intermingled. This results in highly complex distributions, making these datasets unsuitable for direct use as training data for ML-based optimization algorithms. To address

Table 3: Gap estimation of baselines. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. - represents MILP problems that GNN&GBDT cannot solve.

	Gurobi	SCIP	CPLEX	GLNS	ACP	RINS	Learn2branch	GNN&GBDT	Predict&Search	Time
MIS_hard	0.1714	53.4844	1.1265	0.1714	0.1184	0.2998	+	**0.1169**	+	4000s
MVC_hard	0.1310	91.8785	0.7098	0.1018	0.1077	0.1961	+	**0.0951**	+	4000s
SC_hard	0.9920	425323.8	+	0.9852	**0.9850**	0.9902	+	0.9887	+	4000s
MIPlib	**0.0000**	0.0587	**0.0000**	0.2157	0.0004	0.2331	0.3363	-	**0.0000**	150s
Coral	2.85e+04	2.857e+19	4.96e+05	3.05e+04	3.05e+04	3.52e+04	+	-	**2.33e+04**	4000s
Cut	0.1490	0.5387	**0.1046**	0.2744	0.1651	0.3781	0.5782	-	0.1568	4000s
ECOGCNN	**0.2512**	4.6056	0.2899	0.2730	0.2516	0.2814	+	-	**0.2512**	4000s
HEM_knapsack	**0.0000**	0.0000	0.0000	**0.0000**	**0.0000**	0.0117	**0.0000**	**0.0000**	**0.0000**	100s
HEM_corlat	**0.0000**	0.0000	0.0000	0.0129	**0.0000**	0.0141	!	-	**0.0000**	100s
HEM_mik	**0.0000**	0.0000	0.0000	0.0034	0.0146	0.0467	!	-	3.00e-15	100s
item_placement	**0.6481**	2.332e+07	0.9204	0.8431	0.8076	0.9971	4.17e+07	-	0.6595	4000s
load_balancing	0.0028	0.0273	**0.0003**	0.0227	0.0035	0.0616	+	-	0.0028	1000s
anonymous	0.3088	4.2244	0.3386	0.9256	0.5449	0.8876	+	-	**0.2909**	4000s
Nexp	0.0787	0.1514	0.0813	0.1095	0.0754	0.0989	0.1629	-	**0.0759**	4000s
Transportation	**0.1512**	0.2575	0.1646	0.2490	0.1767	0.1962	0.2725	-	0.1568	4000s
vary_bounds_s1	**0.0000**	0.0485	0.0001	0.3956	**0.0000**	0.4536	0.1518	-	0.0003	400s
vary_matrix_s1	**0.0000**	0.3796	0.0001	0.0008	0.0008	0.2443	0.4002	-	**0.0000**	100s
vary_obj_s1	**0.0000**	0.0031	0.0000	0.0019	**0.0000**	0.0012	0.0054	**0.0000**	**0.0000**	100s
vary_rhs_s1	0.0003	0.0364	0.0000	5.5134	0.2037	1.7870	+	-	**0.0000**	100s
Aclib	**0.0000**	0.0000	0.0001	0.0006	0.0028	0.0044	!	-	**0.0000**	100s
fc.data	**0.0000**	0.0000	0.0000	0.1729	**0.0000**	0.8141	!	-	**0.0000**	100s
nn_verification	0.0001	0.0756	0.0001	0.1493	0.1493	0.3734	!	-	**0.0000**	100s

this, we applied the spectral clustering algorithm to reclassify the complete datasets, including MIPLib, Coral, ECOGCNN, and Nexp.

Using MIPLib as an example, we employed spectral clustering to partition the dataset into six distinct classes. To evaluate the similarity between these classes, we calculated the average Euclidean distance between instance embeddings from different classes. As shown in the right panel of Figure 4, these distances provide a quantitative measure of class similarity. Furthermore, the intra-class distances, measured using both graph structural and neural encoding metrics, are visualized in Figure 5. The results reveal that the first five classes exhibit relatively small intra-class and inter-class distances, indicating a high degree of similarity among instances. In contrast, the sixth class stands out with significantly larger distances, suggesting the presence of outlier instances.

Based on these findings, we excluded the sixth class from the MIPLib dataset to ensure a more homogeneous classification, thereby improving its suitability for subsequent baseline tests. Detailed classification results for other datasets, including Coral, ECOGCNN, and Nexp, are provided in Appendix C.3.

4.3 Benchmarking Study

To validate the effectiveness of MILPBench and analyze the performance of representative algorithms across different categories, we conducted comparative experiments focusing on objective function values and gap estimation under a consistent wall-clock time limit. This section highlights the performance of three major algorithm categories: classical solvers (Gurobi, SCIP, and CPLEX), heuristic approaches (GLNS, ACP, and RINS), and machine learning (ML)-based methods (Learn2Branch, GNN&GBDT, and Predict&Search). Detailed comparisons within each category are provided in Appendix C.5. Here, we focus on cross-category comparisons and the unique value of MILPBench in evaluating algorithm performance beyond standard benchmarks.

As shown in Table 3, classical solvers exhibit the most robust and reliable performance across diverse problem types. Among them, Gurobi consistently achieves the lowest gap values in real-world problems such as `load_balancing` and `Transportation`, as well as in smaller-scale precision-oriented problems like `MIPlib` and `HEM_knapsack`. CPLEX also demonstrates competitive performance on many instances, particularly in structured problems such as `Cut`. However, the results highlight the limitations of SCIP under time constraints, with significantly higher gap values in complex problems like `SC_hard` and `MVC_hard`. These findings reaffirm the dominance of classical solvers, especially in scenarios requiring optimality and feasibility guarantees.

Heuristic approaches, such as GLNS, ACP, and RINS, demonstrate effectiveness in large-scale problems like `MIS_hard` and `SC_hard`, where computational efficiency is critical. GLNS performs comparably to Gurobi on `MIS_hard`, while RINS achieves reasonable gap values on `MVC_hard` and `Cut`, albeit at the expense of slightly higher computational time. However, heuristic methods struggle in more constrained scenarios, such as `HEM_corlat` and `HEM_mik`, where their inability to handle tight constraints becomes evident.

ML-based methods, including Learn2Branch, GNN&GBDT, and Predict&Search, demonstrate promising results in specific instances but face challenges in broader contexts. Notably, GNN&GBDT achieves strong performance across all IP problems, outperforming all other baselines in these instances. However, as an IP-specific framework, it struggles to generalize to MILP problems, leaving some instances unsolved. Similarly, Predict&Search shows competitive performance on problems like `Cut` and `vary_rhs_s1`, occasionally matching or outperforming heuristic methods. Nevertheless, it also fails to solve certain constrained problems, limiting its overall robustness. Meanwhile, Learn2Branch struggles with large-scale problems like `Coral` and `Cut`, where the exponential growth of the search space significantly impacts its effectiveness.

To visually compare the performance of representative algorithms across diverse problem types, we employed a scoring mechanism based on their gap estimates. Scores were normalized to a 0–100 scale using the Gaussian transformation $\text{Score}(x) = 100 \cdot$

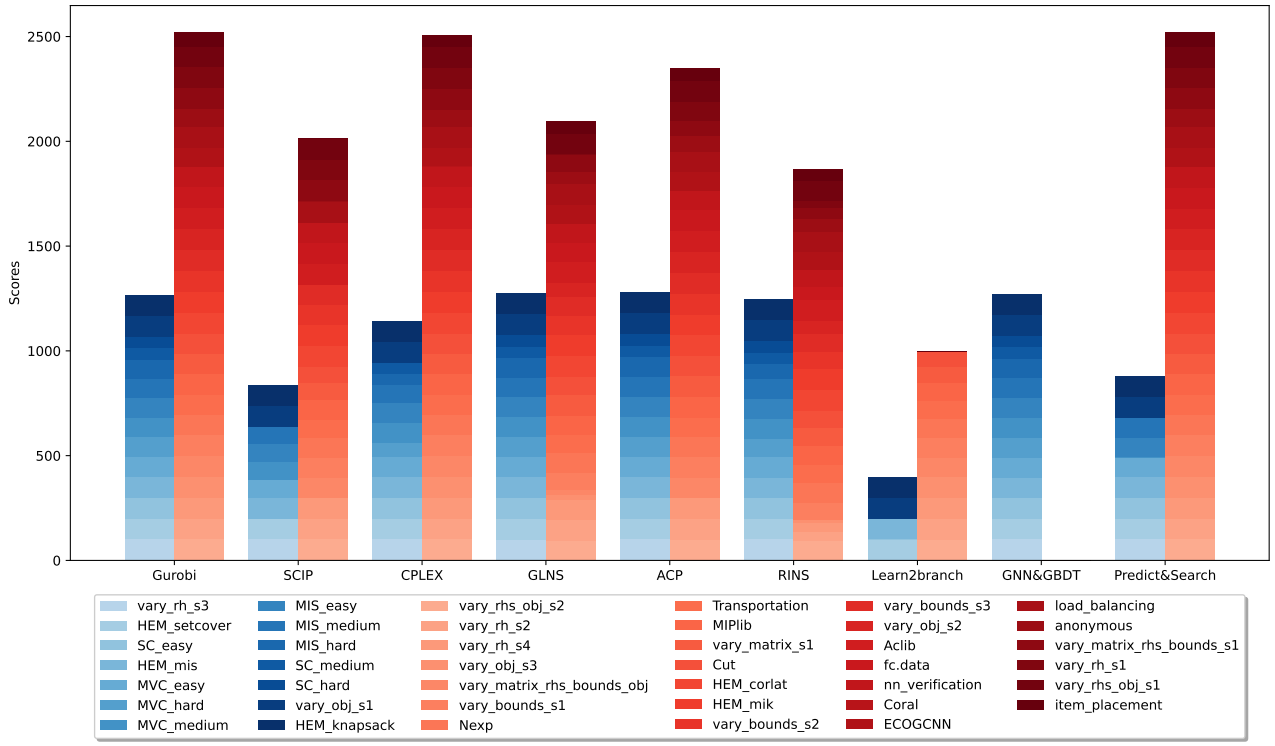


Figure 6: Scores for IP (blue) and MILP (red) problems in MILPBench for different baselines.

$e^{-\frac{x^2}{2}}$, where x is the gap estimate. This approach ensures that algorithms with smaller gap values receive higher scores, providing an objective and standardized metric for cross-category evaluation.

Figure 6 illustrates the normalized scores for integer programming (IP) and mixed-integer linear programming (MILP) problems across all baselines. Classical solvers, particularly Gurobi and CPLEX, consistently achieve higher scores, reflecting their robust performance across both IP and MILP problems. Gurobi excels in precision-oriented and large-scale problems, while CPLEX demonstrates strength in structured instances like `Cut`. In contrast, SCIP achieves relatively lower scores due to its struggles in handling complex and time-constrained problems. Heuristic approaches, such as ACP and RINS, achieve competitive scores on large-scale problems, with ACP consistently outperforming other heuristics. However, their scores decline in constrained scenarios, highlighting the trade-off between computational efficiency and solution robustness. ML-based methods, while showing potential in specific benchmarks, exhibit greater variability. For instance, GNN&GBDT achieves high scores in IP problems but struggles to generalize to MILP instances. Similarly, Predict&Search demonstrates strong performance in structured problems but lacks consistency across other scenarios.

In summary, classical solvers remain the most reliable and versatile algorithms, while heuristics offer computational advantages in large-scale problems. ML-based methods, despite their potential, require further refinement to improve generalizability and scalability. These findings reinforce the importance of MILPBench as a comprehensive benchmarking framework that facilitates detailed performance evaluations and inspires the development of hybrid strategies integrating the strengths of all three categories.

5 Conclusion and Future Work

This paper presented MILPBench, a comprehensive and open-source benchmark dataset for evaluating optimization algorithms in mixed-integer linear programming (MILP). By systematically categorizing problem instances, MILPBench enables structured analyses of algorithm performance, revealing the strengths and limitations of both classical solvers and emerging approaches.

Future work will focus on expanding the dataset to include more diverse MILP problems and enhancing compatibility with evolving optimization methodologies. MILPBench aims to foster rigorous evaluations and drive the development of robust and efficient algorithms, benefiting both research and practical applications.

6 Acknowledgement

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2022-031).

References

[1] David Abramson and Marcus Randall. 1999. A simulated annealing code for general integer linear programs. *Annals of Operations Research* 86, 0 (1999), 3–21.
[2] Tobias Achterberg. 2009. SCIP: solving constraint integer programs. *Mathematical Programming Computation* 1 (2009), 1–41.

[3] Tobias Achterberg and Timo Berthold. 2009. Hybrid branching. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 6th International Conference, CPAIOR 2009 Pittsburgh, PA, USA, May 27-31, 2009 Proceedings 6*. Springer, 309–311.
[4] Tobias Achterberg, Thorsten Koch, and Alexander Martin. 2005. Branching rules revisited. *Operations Research Letters* 33, 1 (2005), 42–54.
[5] Nathan Adelgren and Akshay Gupte. 2022. Branch-and-bound for biobjective mixed-integer linear programming. *INFORMS Journal on Computing* 34, 2 (2022), 909–933.
[6] Kiyan Ahmadizadeh, Bistra Dilkina, Carla P Gomes, and Ashish Sabharwal. 2010. An empirical study of optimization for maximizing diffusion in networks. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 514–521.
[7] Attahiru S Alfa, Bodhaswar T Maharaj, Shruti Lall, and Sougata Pal. 2016. Mixed-integer programming based techniques for resource allocation in underlay cognitive radio networks: A survey. *Journal of Communications and Networks* 18, 5 (2016), 744–761.
[8] Hyung-Chan An, Mohit Singh, and Ola Svensson. 2017. LP-based algorithms for capacitated facility location. *SIAM J. Comput.* 46, 1 (2017), 272–306.
[9] A. Atamtürk, , G. L. Nemhauser, and M. W. P. Savelsbergh. 2001. Valid Inequalities for Problems with Additive Variable Upper Bounds. *Mathematical Programming* 91 (2001), 145–162.
[10] Alper Atamtürk. 2001. Flow pack facets of the single node fixed-charge flow polytope. *Operations Research Letters* 29, 3 (2001), 107–114.
[11] Alper Atamtürk. 2002. On capacitated network design cut–set polyhedra. *Mathematical Programming* 92 (2002), 425–437.
[12] Alper Atamtürk. 2003. On the Facets of the Mixed-Integer Knapsack Polyhedron. *Mathematical Programming* 98, 1-3 (2003), 145–175.
[13] Alper Atamtürk and Juan Carlos Munoz. 2004. A study of the lot-sizing polytope. *Mathematical Programming* 99, 3 (2004), 443–465.
[14] A. Atamtürk and V. Narayanan. 2009. The Submodular 0-1 Knapsack Polytope. 6 (2009), 333–344.
[15] A. Atamtürk and V. Narayanan. 2010. Conic Mixed-Integer Rounding Cuts. *Mathematical Programming* 122 (2010), 1–20.
[16] Michel Bénichou, Jean-Michel Gauthier, Paul Girodet, Gerard Hentges, Gerard Ribière, and Olivier Vincent. 1971. Experiments in mixed-integer linear programming. *Mathematical programming* 1 (1971), 76–94.
[17] Livio Bertacco, Matteo Fischetti, and Andrea Lodi. 2007. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization* 4, 1 (2007), 63–76.
[18] Timo Berthold. 2006. *Primal heuristics for mixed integer programs*. Ph. D. Dissertation. Zuse Institute Berlin (ZIB).
[19] Suresh Bolusani, Mathieu Besançon, Ambros Gleixner, Timo Berthold, Claudia d'Ambrosio, Gonzalo Muñoz, Joseph Paat, and Dimitri Thomopulos. 2023. The MIP Workshop 2023 Computational Competition on Reoptimization. *arXiv preprint arXiv:2311.14834* (2023).
[20] Alberto Caprara, Paolo Toth, and Matteo Fischetti. 2000. Algorithms for The Set Covering Problem. *Annals of Operations Research* 98 (2000), 353–371.
[21] Li Chen, Hua Xu, Ziteng Wang, Chengming Wang, and Yu Jiang. 2023. Self-paced learning based graph convolutional neural network for mixed integer programming (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 16188–16189.
[22] Ziang Chen, Jialin Liu, Xinshang Wang, Jianfeng Lu, and Wotao Yin. 2022. On representing linear programs by graph neural networks. *arXiv preprint arXiv:2209.12288* (2022).
[23] Ziang Chen, Jialin Liu, Xinshang Wang, Jianfeng Lu, and Wotao Yin. 2023. On Representing Mixed-Integer Linear Programs by Graph Neural Networks. arXiv:2210.10759 [cs.LG] https://arxiv.org/abs/2210.10759
[24] CPLEX User's Manual. 1987. IBM ILOG CPLEX Optimization Studio. *Version* 12, 1987-2018 (1987), 1.
[25] A. Şen, A. Atamtürk, and P. Kaminsky. 2015. *A Conic Integer Programming Approach to Constrained Assortment Optimization under the Mixed Multinomial Logit Model*. Research Report BCOL.15.06. IEOR, University of California–Berkeley.
[26] Frank E. Curtis. [n. d.]. COR@L MIP Dataset. https://coral.ise.lehigh.edu/data-sets/mixed-integer-instances/.
[27] Emilie Danna, Edward Rothberg, and Claude Le Pape. 2005. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming* 102 (2005), 71–90.
[28] Sven De Vries and Rakesh V Vohra. 2003. Combinatorial auctions: A survey. *INFORMS Journal on computing* 15, 3 (2003), 284–309.
[29] Santanu S Dey, Yatharth Dubey, Marco Molinaro, and Prachi Shah. 2024. A theoretical and computational analysis of full strong-branching. *Mathematical Programming* 205, 1 (2024), 303–336.
[30] Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. 2020. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 34. 1452–1459.
[31] Irit Dinur and Samuel Safra. 2005. On The Hardness of Approximating Minimum Vertex Cover. *Annals of mathematics* (2005), 439–485.
[32] Matteo Fischetti, Fred Glover, and Andrea Lodi. 2005. The feasibility pump. *Mathematical Programming* 104 (2005), 91–104.
[33] Christodoulos A Floudas and Xiaoxia Lin. 2005. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research* 139 (2005), 131–162.
[34] Maxime Gasse, Simon Bowly, Quentin Cappart, Jonas Charfreitag, Laurent Charlin, Didier Chételat, Antonia Chmiela, Justin Dumouchelle, Ambros Gleixner, Aleksandr M Kazachkov, et al. 2022. The machine learning for combinatorial optimization competition (ml4co): Results and insights. In *NeurIPS 2021 competitions and demonstrations track*. PMLR, 220–231.
[35] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. 2019. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems* 32 (2019).
[36] Sethavidh Gertphol, Yang Yu, Shriram B Gundala, Viktor K Prasanna, Shoukat Ali, Jong-Kook Kim, Anthony A Maciejewski, and Howard Jay Siegel. 2002. A metric and mixed-integer-programming-based approach for resource allocation in dynamic real-time systems. In *Proceedings 16th International Parallel and Distributed Processing Symposium*. IEEE, 10–pp.
[37] 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.
[38] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. 2021. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation* (2021). https://doi.org/10.1007/s12532-020-00194-3
[39] Jens Gottlieb and Lutz Paulmann. 1998. Genetic algorithms for the fixed charge transportation problem. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*. IEEE, 330–335.
[40] Chrysanthos E Gounaris, Karthikeyan Rajendran, Ioannis G Kevrekidis, and Christodoulos A Floudas. 2016. Designing networks: A mixed-integer linear optimization approach. *Networks* 68, 4 (2016), 283–301.
[41] Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. 2020. Hybrid models for learning to branch. *Advances in neural information processing systems* 33 (2020), 18087–18097.
[42] Prateek Gupta, Elias B Khalil, Didier Chetélat, Maxime Gasse, Yoshua Bengio, Andrea Lodi, and M Pawan Kumar. 2022. Lookback for learning to branch. *arXiv preprint arXiv:2206.14987* (2022).
[43] LLC Gurobi Optimization. 2021. Gurobi optimizer reference manual. (2021).
[44] Qingyu Han, Linxin Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. 2023. A gnn-guided predict-and-search framework for mixed-integer linear programming. *arXiv preprint arXiv:2302.05636* (2023).
[45] He He, Hal Daume III, and Jason M Eisner. 2014. Learning to search in branch and bound algorithms. *Advances in neural information processing systems* 27 (2014).
[46] Jörg Heisterman and Thomas Lengauer. 1991. The efficient solution of integer programs for hierarchical global routing. *IEEE transactions on computer-aided design of integrated circuits and systems* 10, 6 (1991), 748–753.
[47] Gregor Hendel. 2022. Adaptive large neighborhood search for mixed integer programming. *Mathematical Programming Computation* 14, 2 (2022), 185–221.
[48] Mojtaba Heydar, Jie Yu, Yue Liu, and Matthew EH Petering. 2016. Strategic evacuation planning with pedestrian guidance and bus routing: a mixed integer programming model and heuristic solution. *Journal of Advanced Transportation* 50, 7 (2016), 1314–1335.
[49] Weimin Huang, Taoan Huang, Aaron M Ferber, and Bistra Dilkina. 2024. Distributional MIPLIB: a Multi-Domain Library for Advancing ML-Guided MILP Methods. *arXiv preprint arXiv:2406.06954* (2024).
[50] Frank Hutter, Manuel López-Ibánez, Chris Fawcett, Marius Lindauer, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2014. AClib: A benchmark library for algorithm configuration. In *Learning and Intelligent Optimization: 8th International Conference, Lion 8, Gainesville, FL, USA, February 16-21, 2014. Revised Selected Papers 8*. Springer, 36–40.
[51] David E Kaufman, Jason Nonis, and Robert L Smith. 1998. A mixed integer linear programming model for dynamic route guidance. *Transportation Research Part B: Methodological* 32, 6 (1998), 431–440.
[52] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).
[53] Thorsten Koch, Tobias Achterberg, Erling Andersen, Oliver Bastert, Timo Berthold, Robert E Bixby, Emilie Danna, Gerald Gamrath, Ambros M Gleixner, Stefan Heinz, et al. 2011. MIPLIB 2010: mixed integer programming library version 5. *Mathematical Programming Computation* 3 (2011), 103–163.

[54] Wen-Yang Ku and J Christopher Beck. 2016. Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research* 73 (2016), 165–173.

[55] A Land and S Powell. 1979. Computer codes for problems of integer programming. In *Annals of Discrete Mathematics*. Vol. 5. Elsevier, 221–269.

[56] Markus Leitner and Markus Leitner. 2010. *Solving two network design problems by mixed integer programming and hybrid optimization methods*. na.

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

[58] Jeffrey T Linderoth and Ted K Ralphs. 2005. Noncommercial software for mixed-integer linear programming. In *Integer programming*. CRC Press, 269–320.

[59] Paramet Luathep, Agachai Sumalee, William HK Lam, Zhi-Chun Li, and Hong K Lo. 2011. Global optimization method for mixed transportation network design problem: a mixed-integer linear programming approach. *Transportation Research Part B: Methodological* 45, 5 (2011), 808–827.

[60] Jiaxiang Luo, Jiyin Liu, and Yueming Hu. 2017. An MILP model and a hybrid evolutionary algorithm for integrated operation optimisation of multi-head surface mounting machines in PCB assembly. *International Journal of Production Research* 55, 1 (2017), 145–160.

[61] Vinod Nair, Mohammad Alizadeh, et al. 2020. Neural large neighborhood search. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*.

[62] 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).

[63] Napoleão Nepomuceno, Ricardo Saboia, and André Coelho. 2023. A MILP-Based Very Large-Scale Neighborhood Search for the Heterogeneous Vehicle Routing Problem with Simultaneous Pickup and Delivery. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 330–338.

[64] Kenta Oono and Taiji Suzuki. 2019. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947* (2019).

[65] Dimitri J Papageorgiou, George L Nemhauser, Joel Sokol, Myun-Seok Cheon, and Ahmet B Keha. 2014. MIRPLib–A library of maritime inventory routing problem instances: Survey, core model, and benchmark results. *European Journal of Operational Research* 235, 2 (2014), 350–366.

[66] Qingyu Qu, Xijun Li, Yunfan Zhou, Jia Zeng, Mingxuan Yuan, Jie Wang, Jinhu Lv, Kexin Liu, and Kun Mao. 2022. An improved reinforcement learning algorithm for learning to branch. *arXiv preprint arXiv:2201.06213* (2022).

[67] RN Ramlogan and IC Goulter. 1989. Mixed integer model for resource allocation in project management. *Engineering optimization* 15, 2 (1989), 97–111.

[68] Edward Rothberg. 2007. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing* 19, 4 (2007), 534–541.

[69] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.

[70] Jialin Song, Yisong Yue, Bistra Dilkina, et al. 2020. A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems* 33 (2020), 20012–20023.

[71] Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. 2021. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201* (2021).

[72] Robert Endre Tarjan and Anthony E Trojanowski. 1977. Finding A Maximum Independent Set. *SIAM J. Comput.* 6, 3 (1977), 537–546.

[73] Mark Turner, Timo Berthold, Mathieu Besançon, and Thorsten Koch. 2023. Branching via cutting plane selection: Improving hybrid branching. *arXiv preprint arXiv:2306.06050* (2023).

[74] Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. 2023. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. *arXiv preprint arXiv:2302.00244* (2023).

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

[76] Laurence A Wolsey. 2020. *Integer programming*. John Wiley & Sons.

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

[78] Huigen Ye, Hongyan Wang, Hua Xu, Chengming Wang, and Yu Jiang. 2023. Adaptive constraint partition based optimization framework for large-scale integer linear programming (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 16376–16377.

[79] Huigen Ye, Hua Xu, and Hongyan Wang. 2023. Light-MILPopt: Solving Large-scale Mixed Integer Linear Programs with Small-scale Optimizer and Small Training Dataset. In *The Twelfth International Conference on Learning Representations*.

[80] 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.

[81] Ryohei Yokoyama, Yuji Shinano, Syusuke Taniguchi, Masashi Ohkura, and Tetsuya Wakui. 2015. Optimization of energy supply systems by MILP branch and bound method in consideration of hierarchical relationship between design and operation. *Energy conversion and management* 92 (2015), 92–104.

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

[83] Ziyan Zhao, Shixin Liu, MengChu Zhou, and Abdullah Abusorrah. 2020. Dual-objective mixed integer linear program and memetic algorithm for an industrial group scheduling problem. *IEEE/CAA Journal of Automatica Sinica* 8, 6 (2020), 1199–1209.

APPENDIX

This Appendix is divided into four sections. Appendix A provides details of the benchmark dataset. Appendix B outlines the specifics of the algorithms. Appendix C presents additional experimental results to further demonstrate the effectiveness and efficiency of MILPBench.

A Details of Benchmark Dataset

A.1 Open-source Datasets

We have carefully curated a substantial collection of mixed-integer linear programming (MILP) instances from a wide range of sources to assemble the MILPBench dataset. These sources include well-established open-source datasets such as MIPLib [53], AClib [50], Regions200 [57], COR@L [26], and MIRPLIB [65]. Additionally, we incorporated problem instances from domain-specific academic studies, including research on robustness verification for neural networks [62], cut selection [74], lot-sizing polytope [13], network diffusion maximization [6], network design [11], fixed-charge flow polytope [10], valid inequalities [9], conic cuts [15, 25], and 0-1 knapsack problems [14]. Furthermore, the dataset includes challenging instances from competitions such as the ML4CO competition at NeurIPS 2021 [34] and the Reoptimization Challenge 2023 [19].

Details regarding the number of instances per problem, including the sizes of training and testing datasets, the average number of decision variables, and the average number of constraints, are provided in Table 4. Due to the double-blind review policy, source URLs for the datasets have been omitted but will be made available after the review process.

Our analysis shows that MILPBench is a robust and diverse dataset, encompassing a wide range of problem characteristics, including varying numbers of decision variables, constraints, and coefficient matrix densities. This diversity ensures that MILPBench is not only suitable for evaluating classical solvers but also provides a valuable resource for understanding and analyzing the strengths and limitations of modern optimization algorithms. By including instances from real-world applications, MILPBench facilitates a comprehensive assessment of algorithm performance across different problem domains and scales, ultimately contributing to the advancement of MILP research and practical applications.

Table 4: Detailed parameter information for each open-source dataset.

Name (Path)	Num. (Train)	Num. (Test)	Avg. Vars	Avg. Cons
nn_verification	3613	9	7144.02	6533.58
item_placement	9990	10	1083	195
load_balancing	9990	10	61000	64307.19
anonymous	134	4	34674.03	44498.19
HEM_knapsack	9995	5	720	72
HEM_mis	9979	5	500	1953.48
HEM_setcover	9995	5	1000	500
HEM_corlat	1979	5	466	486.17
HEM_mik	85	5	386.67	311.67
vary_bounds_s1	45	5	3117	1293
vary_bounds_s2	45	5	1758	351
vary_bounds_s3	45	5	1758	351
vary_matrix_s1	45	5	802	531
vary_matrix_rhs_bounds_s1	45	5	27710	16288
vary_matrix_rhs_bounds_obj	45	5	7973	3558
vary_obj_s1	45	5	360	55
vary_obj_s2	45	5	745	26159
vary_obj_s3	45	5	9599	27940
vary_rhs_s1	45	5	12760	1501
vary_rhs_s2	45	5	1000	1250
vary_rhs_s3	45	5	63009	507
vary_rhs_s4	45	5	1000	1250
vary_rhs_obj_s1	45	5	90983	33438
vary_rhs_obj_s2	45	5	4626	8274
Aclib	89	10	181	180
Coral	272	7	18420.92	11831.01
Cut	11	3	4113	1608.57
ECOGCNN	41	3	36808.25	58768.84
fc.data	15	5	571	330.5
MIPlib	46	4	7719.98	6866.04
Nexp	72	5	9207.09	7977.14
Transportation	27	5	4871.5	2521.467
MIPLIB_collection_easy	639	10	119747.4	123628.3
MIPLIB_collection_hard	102	5	96181.4	101135.8
MIPLIB_collection_open	199	5	438355.9	258599.5
MIRPLIB_Original	67	5	36312.2	11485.8
MIRPLIB_Maritime_Group1	35	5	13919.5	19329.25
MIRPLIB_Maritime_Group2	35	5	24639.8	34053.25
MIRPLIB_Maritime_Group3	35	5	24639.8	34057.75
MIRPLIB_Maritime_Group4	15	5	4343.0	6336.0
MIRPLIB_Maritime_Group5	15	5	48330.0	66812.0
MIRPLIB_Maritime_Group6	15	5	48330.0	66815.0

A.2 Used Assets

MILPBench is an open-source tool, and it can be accessed at: . Table 5 provides a detailed list of the resources and assets utilized in MILPBench, along with their respective licenses. We strictly adhere to these licenses during the development and distribution of MILPBench.

For assets marked as "Need to cite," this indicates that there is no explicit license provided, but the repository explicitly requires that the corresponding publication be cited when using the dataset. For example, datasets prefixed with "vary" in the MILPBench collection originate from the MIP Workshop 2023 Computational Competition (MIPcc23) [19]. Proper citation of the corresponding literature is essential when employing these datasets in any analysis or publication. Similarly, datasets such as Coral [58], MIPlib [38], and Transportation [39] also require appropriate citations. This ensures that all contributors are acknowledged for their work and that the academic integrity of the research community is maintained.

For assets labeled as "Readme," the absence of an explicit license is clarified by a Readme file that specifies the terms of use. For example, the datasets Aclib, Cut, fc.data, and Nexp include detailed Readme files hosted at the following location: Link. These Readme files provide important usage guidelines and permissions, indicating that the datasets can be used without restriction, provided proper acknowledgments are made.

Additionally, the ECOGCNN dataset's usage guidelines are outlined in a Readme file hosted at Link. This link provides comprehensive information about public domain software and datasets, emphasizing adherence to license conditions and proper attribution. By following these practices, MILPBench upholds academic integrity and supports the continued openness and collaboration in the optimization research community.

A.3 Standard Problem Instances

We generated a substantial number of standard problem instances based on seven canonical MILP problems: **Maximum Independent Set (MIS)** [72], **Minimum Vertex Covering (MVC)** [31], **Set Covering (SC)** [20], **Mixed Integer Knapsack Set (MIKS)** [12], **Balanced Item Placement (BIP)** [66], **Combinatorial Auctions (CA)** [28], and **Capacitated Facility Location (CFL)** [8].

Table 5: License of each open source dataset. "Need to cite" means there is no explicit license, but the repository states that the corresponding article needs to be cited to use the dataset and it can be used without restriction. "Readme" means there is no explicit license, but the repository's Readme file explains that it can be used without restriction.

Name(Path)	License	Source
nn_verification	CC BY 4.0	Link
item_placement	BSD 3-Clause License	Link
load_balancing	BSD 3-Clause License	Link
anonymous	BSD 3-Clause License	Link
HEM_knapsack	MIT	Link
HEM_mis	MIT	Link
HEM_setcover	MIT	Link
HEM_corlat	MIT	Link
HEM_mik	MIT	Link
vary_bounds_s1	Need to cite	Link
vary_bounds_s2	Need to cite	Link
vary_bounds_s3	Need to cite	Link
vary_matrix_s1	Need to cite	Link
vary_matrix_rhs_bounds_s1	Need to cite	Link
vary_matrix_rhs_bounds_obj	Need to cite	Link
vary_obj_s1	Need to cite	Link
vary_obj_s2	Need to cite	Link
vary_obj_s3	Need to cite	Link
vary_rhs_s1	Need to cite	Link
vary_rhs_s2	Need to cite	Link
vary_rhs_s3	Need to cite	Link
vary_rhs_s4	Need to cite	Link
vary_rhs_obj_s1	Need to cite	Link
vary_rhs_obj_s2	Need to cite	Link
Aclib	Readme	Link
Coral	Need to cite	Link
Cut	Readme	Link
ECOGCNN	Readme	Link
fc.data	Readme	Link
MIPlib_collection_easy	Need to cite	Link
MIPlib_collection_hard	Need to cite	Link
MIPlib_collection_open	Need to cite	Link
MIRPLIB_Original	BSD 3-Clause License	Link
MIRPLIB_Maritime_Group1	BSD 3-Clause License	Link
MIRPLIB_Maritime_Group2	BSD 3-Clause License	Link
MIRPLIB_Maritime_Group3	BSD 3-Clause License	Link
MIRPLIB_Maritime_Group4	BSD 3-Clause License	Link
MIRPLIB_Maritime_Group5	BSD 3-Clause License	Link
MIRPLIB_Maritime_Group6	BSD 3-Clause License	Link
Nexp	Readme	Link
Transportation	Need to cite	Link

For each type of problem, we generated instances at three levels of difficulty—easy, medium, and hard—corresponding to problem scenarios with tens of thousands, hundreds of thousands, and millions of decision variables, respectively. Below, we detail each problem type and provide their mathematical formulations.

Maximum Independent Set Problem: Given an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a subset of nodes $\mathcal{S} \subseteq \mathcal{V}$ is an independent set if no two nodes in $\mathcal{S}$ are connected by an edge $e \in \mathcal{E}$. The goal of the MIS problem is to find the independent set of maximum cardinality. Representing the decision variables as a binary vector x, where $x_v = 1$ indicates that node v is included in the independent set and $x_v = 0$ otherwise, the problem can be formulated as:

$$\begin{aligned} \max \quad & \sum_{v \in \mathcal{V}} x_v \\ \text{s.t.} \quad & x_u + x_v \leq 1, \quad \forall (u, v) \in \mathcal{E}, \\ & x_v \in \{0, 1\}, \quad \forall v \in \mathcal{V}. \end{aligned} \tag{6}$$

Minimum Vertex Covering Problem: For an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a subset of nodes $\mathcal{S} \subseteq \mathcal{V}$ is a vertex cover if every edge $e \in \mathcal{E}$ has at least one endpoint in $\mathcal{S}$. The MVC problem seeks the vertex cover of minimum cardinality. Using a binary variable x_v where $x_v = 1$ indicates that node v is included in the vertex cover, the problem is formulated as:

$$\begin{aligned} \min \quad & \sum_{v \in \mathcal{V}} x_v \\ \text{s.t.} \quad & x_u + x_v \geq 1, \quad \forall (u, v) \in \mathcal{E}, \\ & x_v \in \{0, 1\}, \quad \forall v \in \mathcal{V}. \end{aligned} \tag{7}$$

Set Covering Problem: Given a finite universal set $\mathcal{U} = \{1, 2, \ldots, n\}$ and a collection of subsets $S_1, S_2, \ldots, S_m \subseteq \mathcal{U}$, each associated with a cost c_i, the SC problem seeks to select the combination of subsets that covers all elements of $\mathcal{U}$ at minimum cost. Using binary decision variables x_i where $x_i = 1$ indicates subset S_i is selected, the problem is formulated as:

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} c_i x_i \\ \text{s.t.} \quad & \sum_{i: j \in S_i} x_i \geq 1, \quad \forall j \in \mathcal{U}, \\ & x_i \in \{0, 1\}, \quad \forall i = 1, \ldots, m. \end{aligned} \tag{8}$$

Mixed Integer Knapsack Set Problem: The MIKS problem is a variant of resource allocation problems involving both binary and continuous decisions. Given N sets and M items, the goal is to minimize the total cost of selecting sets while ensuring that each item is covered by at least one set. Let x_i denote the decision variable for set i, where $x_i = 1$ if set i is fully selected and $0 \leq x_i \leq 1$ if set i is partially selected. The problem is formulated as:

$$\begin{aligned} \min \quad & \sum_{i=1}^{N} c_i x_i \\ \text{s.t.} \quad & \sum_{i: j \in S_i} x_i \geq 1, \quad \forall j = 1, \ldots, M, \\ & 0 \leq x_i \leq 1, \quad \forall i = 1, \ldots, N. \end{aligned} \tag{9}$$

Balanced Item Placement Problem: The BIP problem involves distributing N items across B buckets to minimize the imbalance across resource dimensions. Let $x_{i,j}$ be a binary variable where $x_{i,j} = 1$ if item i is placed in bucket j. The problem is formulated as:

$$\begin{aligned} \min \quad & \sum_{r=1}^{R} \text{max\_deficit}_r \\ \text{s.t.} \quad & \sum_{j=1}^{B} x_{i,j} = 1, \quad \forall i = 1, \ldots, N, \\ & \sum_{i=1}^{N} w_{i,j,r} x_{i,j} \leq C_{j,r}, \quad \forall j, r, \\ & x_{i,j} \in \{0, 1\}, \quad \forall i, j. \end{aligned} \tag{10}$$

Combinatorial Auction Problem: In CA problems, bidders place bids on combinations of items. The goal is to select bids that maximize revenue while ensuring no item is allocated to more than one bidder. Using binary decision variables x_i where $x_i = 1$ if bid i is selected, the problem is formulated as:

Table 6: Summary of Generated Instances for Seven Problem Types

Name (Path)	Number of Instances	Avg. Vars	Avg. Cons
MIS_easy	50	20000	60000
MIS_medium	50	100000	300000
MIS_hard	50	1000000	3000000
MVC_easy	50	20000	60000
MVC_medium	50	100000	300000
MVC_hard	50	1000000	3000000
SC_easy	50	40000	40000
SC_medium	50	200000	200000
SC_hard	50	2000000	2000000
BIP_easy	50	4081	290
BIP_medium	50	14182	690
BIP_hard	50	54584	2090
CAT_easy	50	2000	2000
CAT_medium	50	22000	22000
CAT_hard	50	2000000	2000000
CFL_easy	50	16040	80
CFL_medium	50	144200	320
CFL_hard	50	656520	800
MIKS_easy	50	5000	5000
MIKS_medium	50	55000	55000
MIKS_hard	50	1000000	1000000

$$\begin{aligned} \max \quad & \sum_{i=1}^{N} c_i x_i \\ \text{s.t.} \quad & \sum_{i:j\in S_i} x_i \le 1, \quad \forall j = 1, \ldots, M, \\ & x_i \in \{0,1\}, \quad \forall i = 1, \ldots, N. \end{aligned} \tag{11}$$

Capacitated Facility Location Problem: The CFL problem involves determining optimal facility locations and customer assignments while minimizing total costs. Let $x_{i,j}$ be a binary variable indicating whether customer i is assigned to facility j, and y_j indicate whether facility j is opened. The problem is formulated as:

$$\begin{aligned} \min \quad & \sum_{i=1}^{N}\sum_{j=1}^{M} c_{i,j}x_{i,j} + \sum_{j=1}^{M} f_j y_j \\ \text{s.t.} \quad & \sum_{j=1}^{M} x_{i,j} = 1, \quad \forall i = 1, \ldots, N, \\ & \sum_{i=1}^{N} d_i x_{i,j} \le C_j y_j, \quad \forall j = 1, \ldots, M, \\ & x_{i,j}, y_j \in \{0,1\}, \quad \forall i, j. \end{aligned} \tag{12}$$

A.4 Training and Testing Dataset Partition

For training and testing, we have randomly partitioned the problem data into training and testing datasets, the result of testing dataset is shown in https://anonymous.4open.science/r/MILPBench.

B Details of ACP

The **Adaptive Constraint Partition Based Optimization Framework (ACP)** is an iterative approach designed to improve the solution quality of integer programming (IP) problems. It starts with an initial feasible solution $\overline{x}$, which can be obtained from a subroutine solver such as Gurobi or SCIP. At each iteration, the set of constraints C is randomly partitioned into k disjoint blocks, $C = C_1 \cup C_2 \cup \cdots \cup C_k$, where $C_i \cap C_j = \emptyset$ for $i \neq j$. A single block C_i is then selected, and the associated decision variables X_{sub} are optimized using a subroutine solver, while the remaining variables $X \setminus X_{sub}$ are fixed at their current values. This localized optimization reduces computational complexity by focusing on a smaller subproblem rather than solving the entire problem.

To avoid stagnation in local optima, ACP adaptively adjusts the number of blocks k based on the improvement of the objective value $f(\overline{x})$. If the improvement rate between consecutive iterations is smaller than a predefined threshold ϵ, k is reduced to expand the neighborhood, allowing the algorithm to explore a broader search space. The process continues until a stopping criterion, such as a time limit or convergence, is met. The final solution $\overline{x}^*$ is returned as the improved result. The pseudocode for ACP is provided in Algorithm 1.

Algorithm 1 The Framework of ACP

Input: An IP P with constraints C, an initial feasible solution $\overline{x}$, a subroutine solver F
Output: An improved solution $\overline{x}^*$
while Stopping criteria not met **do**
  Randomly partition C into $C_1, C_2, \ldots, C_k$
  Select variables X_{sub} associated with a random block C_i
  Fix $X \setminus X_{sub}$ at their current values
  Optimize X_{sub} using solver F to obtain updated solution $\overline{x}$
  if $f(\overline{x}) - f(\text{Last\_x}) < \epsilon \cdot f(\text{Last\_x})$ **then**
    $k \leftarrow k - 1$ {Adaptive Adjustment}
  end if
end while
Return: $\overline{x}$

C Details of Experiments

C.1 Experimental Environments

All experiments were conducted on a machine equipped with an Intel Xeon Platinum 8375C @ 2.90GHz CPU and four NVIDIA TESLA V100 (32G) GPUs. Each scale of Benchmark MILP problems was tested across multiple instances, and the results presented are the average values from five runs.

C.2 Dataset Analysis

The dataset analysis involved measuring both graph structural embedding distances and neural embedding distances, comparing these metrics with those derived from established MILP datasets. To ensure computational feasibility, comparisons were restricted to representative problems containing fewer than 50,000 decision variables.

C.3 Dataset Reclassification

For certain datasets, particularly those sourced from open collections, problem instances generated from diverse scenarios are often combined, resulting in a highly complex distribution. To address this, we utilized the spectral clustering algorithm to reclassify the dataset into distinct clusters. The reclassified datasets and their respective clusters are presented in https://anonymous.4open.science/r/MILPBench.

Table 7: Objective function value of solver baselines. The bold numbers indicate the best results achieved by the solvers.

Problem	Gurobi	SCIP	CPLEX	Time
MIS_easy	**4598.067**	3723.410	4554.145	600s
MIS_medium	**21754.976**	18627.825	21246.690	2000s
MIS_hard	**216540.975**	9078.864	209950.985	4000s
MVC_easy	**5383.041**	6290.954	5407.354	600s
MVC_medium	**28198.226**	31262.780	28308.290	4000s
MVC_hard	**283348.706**	490890.570	327134.826	4000s
SC_easy	3301.348	5047.490	**3209.813**	600s
SC_medium	**18026.891**	25228.193	16507.670	2000s
SC_hard	**320046.569**	919489.182	-	4000s
MIPlib	**18436.048**	18436.534	18436.048	150s
Coral	**3805.704**	8.484e+07	5741.045	4000s
Cut	28944.667	36987.667	**25092.500**	4000s
ECOGCNN	**755952.240**	755954.907	755954.573	4000s
HEM_knapsack	**422.600**	**422.600**	**422.600**	100s
HEM_mis	**228.800**	**228.800**	**228.800**	100s
HEM_setcover	**231.600**	**231.600**	**231.600**	100s
HEM_corlat	**251.000**	**251.000**	**251.000**	100s
HEM_mik	-62758.400	**-62758.400**	-62758.400	100s
item_placement	**5.331**	10.803	6.530	4000s
load_balancing	**708.800**	712.000	708.800	1000s
anonymous	**249645.728**	1.068e+06	274700.122	4000s
Nexp	**1.163e+08**	1.167e+08	1.163e+08	4000s
Transportation	**1.240e+06**	1.300e+06	1.255e+06	4000s
vary_bounds_s1	**12381.800**	12542.200	12381.800	400s
vary_bounds_s2	**351.000**	351.000	**351.000**	1000s
vary_bounds_s3	**351.000**	351.000	**351.000**	1000s
vary_matrix_s1	**61.594**	62.618	61.594	100s
vary_matrix_rhs_bounds_s1	2.002e+09	**2.002e+09**	2.002e+09	100s
vary_matrix_rhs_bounds_obj_s1	**-51638.297**	-45611.603	-51637.689	100s
vary_obj_s1	**8625.400**	8630.000	**8625.400**	100s
vary_obj_s2	**1169.489**	1171.025	1169.489	150s
vary_obj_s3	**-2180.098**	30.251	**-349.464**	100s
vary_rhs_s1	-349.464	-338.944	**-349.464**	100s
vary_rhs_s2	**-17168.351**	-17167.794	-17167.985	100s
vary_rhs_s3	57259.200	**57257.400**	57258.400	100s
vary_rhs_s4	-17166.446	**-17166.460**	-17166.275	100s
vary_rhs_obj_s1	**-179337.380**	-177955.024	-179338.088	600s
vary_rhs_obj_s2	-807958.650	**-807929.003**	-807960.279	100s
Aclib	82427.000	**82427.000**	82427.000	100s
fc.data	**378.600**	**378.600**	**378.600**	100s
nn_verification	**-8.251**	-8.390	-8.251	100s

C.4 Settings of Benchmarking Study

In our benchmarking study, hyperparameter selection played a critical role in optimizing the performance of the baseline algorithms. Different strategies were employed for hyperparameter tuning depending on the method. While some algorithms performed well with default settings, others required careful manual tuning of specific hyperparameters to achieve better performance on complex problem instances.

The study evaluated nine baseline algorithms discussed in the main text: **Gurobi**, **SCIP**, **CPLEX**, **GLNS**, **ACP**, **Learn2Branch**, **RINS**, **GNN&GBDT**, and **Predict&Search**. Additionally, we included six supplementary machine learning-based methods: **LIH**, **MIH**, **LNS**, **Neural Diving**, **Hybrid_Learn2Branch**, and **GNN-MILP**. These methods are categorized into three groups: classical solvers, heuristic approaches, and machine learning-based methods.

For the classical solvers, including **Gurobi**, **SCIP**, and **CPLEX**, we used the default solver settings. These configurations are generally well-optimized for a wide range of problems and provide strong performance without the need for additional parameter tuning.

For heuristic methods, we manually explored different hyperparameter configurations to ensure the algorithms performed as expected. The details for each method are as follows:

- For **LIH**, **MIH**, and **RINS**, which are less sensitive to hyperparameters, we used consistent settings across all problems: `choose=0.5` and `turn_limit=100`.
- For **LNS**, we set `choose=0.5` and `turn_limit=50` for all problems.
- For **ACP**, which features an adaptive neighborhood size adjustment mechanism, we used `Initial blocking num=2` for all problem instances.
- For **GLNS**, which is more sensitive to initial hyperparameters, we applied the following configurations:
  - For most problems: `blocking num=2`.
  - For `MVC_easy`, `item_placement`, `load_balancing`, and `anonymous`: `blocking num=3`.
  - For larger-scale problems like `MIS_medium`, `MVC_medium`, and `SC_medium`: `blocking num=4`.
  - For the largest problems, such as `MVC_hard`: `blocking num=5`; and for `MIS_hard` and `SC_hard`: `blocking num=6`.

For machine learning-based methods, most training parameters were consistent across different problem instances. The detailed configurations are as follows:

- **Learn2Branch**: For most problems, the following settings were used: `sample-rate=10, learning-rate=0.001, max-epoch=100, num-bad-epoch=20, batch-size= pretrain-batch-size= valid-batch-size=64`. For `vary_bounds_s2` and `vary_bounds_s3`, we reduced `sample-rate=1` to account for extended sampling times.
- **GNN&GBDT**: The following consistent settings were applied across all problems: `learning-rate=0.0001,`

Table 8: Gap estimation of solver baselines. The bold numbers indicate the best results achieved by the solvers.

Problem	Gurobi	SCIP	CPLEX	Time
MIS_easy	**0.091**	0.355	0.099	600s
MIS_medium	**0.163**	0.361	0.192	2000s
MIS_hard	**0.171**	53.484	1.127	4000s
MVC_easy	**0.075**	0.271	0.077	600s
MVC_medium	**0.125**	0.270	0.129	4000s
MVC_hard	**0.131**	93.587	0.710	4000s
SC_easy	0.041	1.000e+20	**0.016**	600s
SC_medium	**0.986**	2.000e+19	1.000	2000s
SC_hard	**0.992**	425323.823	-	4000s
MIPlib	**2.471e-05**	0.059	4.964e-05	150s
Coral	**28470.925**	2.857e+19	5.714e+09	4000s
Cut	0.149	0.539	**0.105**	4000s
ECOGCNN	**0.251**	4.606	0.290	4000s
HEM_knapsack	**0.000**	**0.000**	**0.000**	100s
HEM_mis	**0.000**	**0.000**	**0.000**	100s
HEM_setcover	**0.000**	**0.000**	**0.000**	100s
HEM_corlat	**0.000**	**0.000**	**0.000**	100s
HEM_mik	3.082e-15	**0.000**	9.668e-05	100s
item_placement	**0.648**	2.332e+07	0.920	4000s
load_balancing	**0.003**	0.027	**0.003**	1000s
anonymous	**0.309**	4.224	0.339	4000s
Nexp	**0.079**	0.151	0.081	4000s
Transportation	**0.151**	0.258	0.165	4000s
vary_bounds_s1	**1.729e-05**	0.049	5.663e-05	400s
vary_bounds_s2	**0.000**	0.086	**0.000**	1000s
vary_bounds_s3	**0.000**	0.089	**0.000**	1000s
vary_matrix_s1	**0.000**	0.380	6.828e-05	100s
vary_matrix_rhs_bounds_s1	3.488e-05	**6.789e-06**	3.036e-05	100s
vary_matrix_rhs_bounds_obj_s1	**3.810e-05**	0.113	9.782e-05	100s
vary_obj_s1	**0.000**	0.003	**0.000**	100s
vary_obj_s2	**2.029e-07**	6.825	0.010	150s
vary_obj_s3	**0.000**	6.000e+19	**0.000**	100s
vary_rhs_s1	**0.000**	0.036	**0.000**	100s
vary_rhs_s2	**1.299e-05**	0.001	9.565e-05	100s
vary_rhs_s3	6.879e-05	**5.491e-05**	6.308e-05	100s
vary_rhs_s4	2.492e-05	**0.000**	9.927e-05	100s
vary_rhs_obj_s1	**8.469e-05**	0.009	9.313e-05	600s
vary_rhs_obj_s2	7.356e-05	**0.000**	9.818e-05	100s
Aclib	2.243e-06	**0.000**	9.505e-05	100s
fc.data	**0.000**	**0.000**	**0.000**	100s
nn_verification	**5.108e-05**	0.076	7.463e-05	100s

`max-patient-epoch=10, n_estimators=30, max-depth=5, rate=0.4, fix=0.6`.

- **Predict&Search**: Uniform settings were used: `learning-rate=0.001, NB-epochs=9999, batch-size=4, weight-norm=100`.
- **GNN-MILP**: `learning-rate=0.03, max-epoch=200`.
- **Neural Diving**: `batch-size=1, learning-rate=0.0001, num-epochs=30`.
- **Hybrid_Learn2Branch**: `sample-rate=10, learning-rate=0.001, max-epoch=1000, patience=10, early-stopping=20, epoch-size=312, batch-size=32, pretrain-batch-size=128, valid-batch-size=128`.

These configurations provide a detailed overview of the hyperparameters and their respective values, along with the rationale behind any modifications. By carefully selecting and tuning these parameters, we ensured that each algorithm performed at its best for the given problem instances. This information is essential for understanding the choices made during the benchmarking process and for facilitating the reproducibility of our results.

C.5 Benchmarking Study

In this section, we present an extensive benchmarking study to evaluate the performance of various algorithms across different problem instances. The study is designed to provide insights into the strengths and limitations of classical solvers, heuristic approaches, and machine learning-based methods. By systematically comparing their performance, we aim to identify specific scenarios where each method excels and where it struggles, highlighting the practical implications for solving mixed-integer linear programming (MILP) problems.

The study includes the following groups of algorithms:

(1) **Classical Solvers**: These are general-purpose optimization tools that rely on branch-and-bound, cutting planes, and other advanced techniques. In this study, we benchmarked three prominent solvers:
   - **Gurobi (version 11.0.1)** [43]
   - **SCIP (version 4.3.0)** [2]
   - **CPLEX (version 22.1.1.0)** [24]

(2) **Heuristic Approaches**: These methods are designed to find approximate solutions in a computationally efficient manner, often using problem-specific heuristics or metaheuristics. We evaluated the following algorithms:
   - **General Large Neighborhood Search (GLNS)** [70]
   - **Adaptive Constraint Partitioning Optimization Framework (ACP)** [78]
   - **Relaxation Induced Neighborhood Search (RINS)** [27]
   - **Least Integeal Heuristic (LIH)** [61]
   - **Most Integeal Heuristic (MIH)** [18]
   - **Large Neighborhood Search (LNS)** [47]

(3) **Machine Learning-Based Methods**: These approaches leverage machine learning models to improve or replace

Table 9: Objective function value of heuristic baselines. The bold numbers indicate the best results achieved by the solvers.

Problem	GLNS	ACP	LIH	MIH	LNS	RINS	Time
MIS_easy	4587.126	**4610.652**	3694.908	3477.540	4592.077	4590.956	600s
MIS_medium	22848.493	**23185.858**	18577.235	17484.158	23069.472	23064.101	2000s
MIS_hard	216526.551	226803.013	195782.155	157685.378	**229976.132**	162217.674	4000s
MVC_easy	5395.675	**5368.408**	6299.467	6524.834	6127.823	5386.112	600s
MVC_medium	27084.455	**26767.794**	31394.246	32493.615	30555.503	26873.974	4000s
MVC_hard	274125.746	275924.443	306567.072	327742.416	**270187.795**	337703.358	4000s
SC_easy	3252.528	**3190.472**	5126.109	5189.812	5717.808	3288.794	600s
SC_medium	16250.072	**15892.545**	32038.169	21908.331	28816.370	32038.169	2000s
SC_hard	172678.041	170261.609	519715.800	**22825.301**	161922.188	321093.060	4000s
MIPlib	19767.810	**18436.257**	5.978e+06	8.481e+06	8.481e+06	36234.829	150s
Coral	4.674e+08	**1.401e+08**	3.380e+09	3.380e+09	3.380e+09	3.380e+09	4000s
Cut	33503.182	30678.000	34765.333	38248.667	**25406.333**	37315.333	4000s
ECOGCNN	758358.302	**757015.035**	757061.333	765457.928	758659.000	765456.595	4000s
HEM_knapsack	**422.600**	**422.600**	420.400	417.800	420.000	417.800	100s
HEM_mis	227.600	**228.800**	160.000	145.200	227.600	226.600	100s
HEM_setcover	233.000	**231.600**	**231.600**	2597.600	2085.600	231.800	100s
HEM_corlat	248.800	**251.000**	249.800	245.000	249.800	248.600	100s
HEM_mik	-62523.200	-61759.600	-59673.200	-59673.200	**-62758.400**	-59673.200	100s
item_placement	12.755	**10.675**	663.912	663.912	666.718	663.912	4000s
load_balancing	723.200	**709.300**	756.500	756.500	756.500	756.500	1000s
anonymous	2.041e+06	**528655.978**	737536.042	4.045e+06	4.044e+06	1.878e+06	4000s
Nexp	1.178e+08	**1.163e+08**	1.171e+08	1.857e+08	1.472e+08	1.189e+08	4000s
Transportation	1.402e+06	**1.278e+06**	1.309e+06	2.981e+06	2.981e+06	1.309e+06	4000s
vary_bounds_s1	20661.400	**12381.800**	18497.200	31423.800	16778.600	22871.400	400s
vary_bounds_s2	413.600	**351.000**	525.000	550.000	358.200	536.000	1000s
vary_bounds_s3	417.200	**351.000**	525.000	550.000	361.800	536.000	1000s
vary_matrix_s1	**61.642**	61.646	81.547	81.547	63.077	81.547	100s
vary_matrix_rhs_bounds_s1	**2.875e+09**	5.887e+09	3.673e+10	3.673e+10	3.673e+10	3.673e+10	100s
vary_matrix_rhs_bounds_obj_s1	-21294.907	**-48212.645**	-4103.848	-4103.848	-4103.848	-4103.848	100s
vary_obj_s1	8642.000	**8625.400**	8635.400	11754.600	9563.800	8635.800	100s
vary_obj_s2	4045.943	**1169.488**	3322.903	3322.903	3322.903	3322.903	150s
vary_obj_s3	**1127.278**	638.801	1097.307	1313.842	1122.516	1272.107	100s
vary_rhs_s1	-54.416	**-291.528**	-231.064	276139.584	-97.656	-130.672	100s
vary_rhs_s2	-16711.302	**-17153.912**	-7409.521	-7409.523	-7409.523	-7409.523	100s
vary_rhs_s3	57257.400	**57257.400**	57258.400	68805.000	66770.800	57258.800	100s
vary_rhs_s4	-16792.771	**-17139.156**	-7414.353	-7414.353	-7414.353	-7414.353	100s
vary_rhs_obj_s1	-175657.357	**-178610.603**	-175985.671	-175985.671	-175985.671	-175985.671	600s
vary_rhs_obj_s2	-707409.607	**-764259.878**	-694023.828	-694023.828	-694023.828	-694023.828	100s
Aclib	**82450.799**	82827.399	82833.900	122098.800	146566.600	82791.600	100s
fc.data	490.400	**378.600**	1732.000	3915.000	3793.400	2013.000	100s
nn_verification	**-9.718**	**-9.718**	-9.913	-9.913	-9.913	-13.166	100s

traditional optimization components, such as branching decisions or heuristic search. In this category, we benchmarked:

- **Learn2Branch** [35]
- **GNN&GBDT-guided Optimization Framework (GNN&GBDT)** [80]
- **Predict&Search Optimization Framework (Predict&Search)** [44]
- **GNN-MILP** [22]
- **Neural Diving** [62]
- **Hybrid_Learn2Branch** [41]

For each group, we analyze the performance of the algorithms across a diverse set of problem instances, including `MIS`, `MVC`, `SC`, `item_placement`, `load_balancing`, and other tasks. Each problem instance varies in complexity, scale, and structure, allowing for a comprehensive comparison.

After presenting the detailed analysis for each group, we provide a summary that synthesizes the findings, identifies key trends, and underscores the contributions of our MILPBench framework. MILPBench serves as a standardized and reproducible benchmarking platform, offering a unique opportunity to evaluate and improve MILP-solving strategies in both academic and industrial contexts.

C.5.1 Classical Solvers. Classical solvers are widely recognized as the gold standard for solving MILP problems due to their reliance on advanced optimization techniques, such as branch-and-bound, branch-and-cut, and cutting planes. In this benchmarking study, we evaluate the performance of three prominent solvers: **Gurobi**, **SCIP**, and **CPLEX**, across a variety of problem instances. The results, presented in Table 7 (objective function values) and Table 8 (GAP estimations), highlight the strengths and limitations of these solvers under different scenarios.

From Table 7, Gurobi demonstrates superior performance in solving large-scale and complex problems, such as `MIS_hard` and `SC_hard`, achieving the best objective function values in these instances. Its advanced optimization techniques and parallelization capabilities allow it to effectively handle challenging problems, making it the most robust solver overall. In contrast, SCIP struggles with these harder instances, producing significantly worse objective function values compared to both Gurobi and CPLEX. However, SCIP performs competitively in smaller-scale problems, such as `MIS_easy` and `MIS_medium`, where its results are closer to those of the other solvers.

CPLEX stands out for its consistent performance in problems that benefit from structured cutting planes and preprocessing, such as `SC_easy` and `Cut`. In these cases, it achieves the best objective function values and demonstrates its effectiveness in solving problems with specific structural characteristics. Furthermore, CPLEX achieves smaller GAP values in many instances, as shown in Table 8, indicating its ability to find near-optimal solutions within the specified time limits. For example, in instances like `SC_easy` and `vary_bounds_s3`, CPLEX outperforms its competitors in terms of both objective value and GAP.

Table 10: Gap estimation of heuristic baselines. The bold numbers indicate the best results achieved by the solvers.

Problem	GLNS	ACP	LIH	MIH	LNS	RINS	Time
MIS_easy	0.093	**0.088**	0.357	0.442	0.092	0.092	600s
MIS_medium	0.108	**0.092**	0.362	0.448	0.097	0.097	2000s
MIS_hard	0.171	0.118	0.299	0.622	**0.103**	0.564	4000s
MVC_easy	0.077	**0.073**	0.210	0.237	0.188	0.076	600s
MVC_medium	0.089	**0.079**	0.214	0.241	0.193	0.082	4000s
MVC_hard	0.102	0.108	0.197	0.248	**0.089**	0.271	4000s
SC_easy	0.029	**0.010**	0.309	0.371	0.448	0.039	600s
SC_medium	0.985	**0.984**	0.992	0.989	0.991	0.992	2000s
SC_hard	0.985	0.985	0.995	**0.887**	0.984	0.992	4000s
MIPlib	0.216	**0.000**	0.319	0.764	0.763	0.233	150s
Coral	30468.591	**30468.410**	35715.405	35247.380	35247.362	35247.380	4000s
Cut	0.274	0.165	0.345	0.392	**0.126**	0.378	4000s
ECOGCNN	0.273	**0.252**	0.275	0.287	0.280	0.281	4000s
HEM_knapsack	**0.000**	**0.000**	0.005	0.012	0.006	0.012	100s
HEM_mis	0.005	**0.000**	0.434	0.583	0.005	0.010	100s
HEM_setcover	0.007	**0.000**	**0.000**	0.911	0.889	0.001	100s
HEM_corlat	0.013	**0.000**	0.007	0.038	0.007	0.014	100s
HEM_mik	0.003	0.015	0.047	0.047	**3.082e-15**	0.047	100s
item_placement	0.843	**0.808**	0.997	0.997	0.997	0.997	4000s
load_balancing	0.023	**0.004**	0.062	0.062	0.062	0.062	1000s
anonymous	0.926	**0.545**	0.788	0.963	0.963	0.888	4000s
Nexp	0.109	**0.075**	0.123	0.317	0.175	0.099	4000s
Transportation	0.249	**0.177**	0.196	0.647	0.647	0.196	4000s
vary_bounds_s1	0.396	**1.729e-05**	0.326	0.601	0.260	0.454	400s
vary_bounds_s2	0.151	**0.000**	0.331	0.362	0.020	0.345	1000s
vary_bounds_s3	0.159	**7.443e-10**	0.331	0.362	0.030	0.345	1000s
vary_matrix_s1	**0.001**	**0.001**	0.244	0.244	0.023	0.244	100s
vary_matrix_rhs_bounds_s1	**0.286**	0.557	0.945	0.945	0.945	0.945	100s
vary_matrix_rhs_bounds_obj_s1	1.920	**0.067**	12.143	12.143	12.143	12.143	100s
vary_obj_s1	0.002	**0.000**	0.001	0.266	0.098	0.001	100s
vary_obj_s2	0.723	**6.323e-07**	0.701	0.701	0.701	0.701	150s
vary_obj_s3	**2.188**	10.084	2.397	2.346	2.458	2.386	100s
vary_rhs_s1	5.513	**0.204**	0.552	1.001	4.025	1.787	100s
vary_rhs_s2	0.027	**0.001**	1.317	1.317	1.317	1.317	100s
vary_rhs_s3	4.274e-05	**3.762e-05**	5.302e-05	0.174	0.151	6.526e-05	100s
vary_rhs_s4	0.022	**0.002**	1.315	1.315	1.315	1.315	100s
vary_rhs_obj_s1	0.019	**0.004**	0.017	0.017	0.017	0.017	600s
vary_rhs_obj_s2	0.103	**0.037**	0.122	0.122	0.122	0.122	100s
Aclib	**0.001**	0.003	0.004	0.347	0.431	0.004	100s
fc.data	0.173	**6.521e-09**	0.790	0.898	0.896	0.814	100s
nn_verification	**0.149**	**0.149**	0.151	0.151	0.151	0.373	100s

When analyzing performance by problem type, Gurobi and CPLEX dominate in most scenarios, with each solver excelling in different areas. For `MIS` problems, Gurobi consistently achieves the best results, particularly in harder instances like `MIS_hard`, where its objective function value surpasses those of both SCIP and CPLEX. For `MVC` problems, Gurobi and CPLEX show comparable performance in easier cases like `MVC_easy`, but Gurobi outperforms the other solvers in harder instances such as `MVC_hard`. In contrast, SCIP faces significant challenges in these harder problems, often producing suboptimal solutions or larger GAP values.

For `SC` problems, CPLEX demonstrates its strength in structured and easier instances like `SC_easy`, where it achieves the best results. However, as the problem scale increases, Gurobi takes the lead in instances like `SC_hard`, showcasing its scalability and robustness. For `HEM` problems, including `HEM_knapsack`, `HEM_mis`, and `HEM_setcover`, all solvers achieve identical objective function values and a GAP of zero, indicating that these instances are relatively simple and well-suited for classical solvers.

Finally, in specialized instances like `Cut`, CPLEX excels by leveraging its advanced cutting strategies, achieving the best performance among the three solvers. Similarly, for problems like `vary_bounds_s3` and `vary_rhs_obj_s1`, CPLEX achieves the smallest GAP, reaffirming its ability to find high-quality solutions in a short amount of time. However, Gurobi remains the top choice for large-scale and complex instances, where its computational efficiency and optimization strategies consistently outperform SCIP and, in some cases, CPLEX.

In summary, Gurobi and CPLEX emerge as the most competitive solvers among the three, with each excelling in different scenarios. Gurobi's scalability and robustness make it particularly effective for large-scale and complex problems, while CPLEX demonstrates superior performance in structured and easier instances, often achieving smaller GAP values. SCIP, while competitive in smaller and simpler problems, struggles with harder and larger instances, indicating limitations in its current implementation. These results establish a solid baseline for comparing heuristic and machine learning-based methods, providing valuable insights into the relative strengths and weaknesses of classical solvers.

C.5.2 Heuristic Approaches. Heuristic approaches aim to provide approximate solutions to MILP problems with a focus on computational efficiency. Unlike classical solvers, heuristics often exploit problem-specific structures or employ metaheuristic strategies, making them particularly suited for large-scale and complex instances where exact solvers may struggle. In this study, we evaluate six heuristic methods: **GLNS**, **ACP**, **LIH**, **MIH**, **LNS**, and **RINS**. The results, summarized in Table 9 (objective function values) and Table 10 (GAP estimations), provide a comprehensive view of their performance across diverse problem instances.

From Table 9, we observe that **GLNS** achieves competitive objective function values in a range of instances, particularly in problems such as `MIS_easy` and `MIS_medium`, where its results closely match

Table 11: Objective function value of machine learning-based baselines. The bold numbers indicate the best results achieved by the solvers. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. -represents MILP problems that cannot be solved by the IP framework, GNN&GBDT.

Problem	Learn2Branch	GNN&GBDT	Predict&Search	Hybrid-Learn2Branch	GNN-MILP	Nerual Diving	Time
MIS_easy	+	**4507.545**	4358.907	+	3469.052	25.188	600s
MIS_medium	+	22703.459	**23169.727**	+	!	13.449	2000s
MIS_hard	+	**227103.684**	!	+	!	!	4000s
MVC_easy	+	5473.262	5613.360	+	6753.481	**5370.605**	600s
MVC_medium	+	**27326.389**	!	+	!	28311.713	4000s
MVC_hard	+	**272090.841**	!	+	!	!	4000s
SC_easy	+	3285.568	**3229.912**	+	3573.991	3233.711	600s
SC_medium	+	**16485.787**	!	+	!	18143.986	2000s
SC_hard	+	**228879.997**	!	+	!	!	4000s
MIPlib	18915.476	-	**18436.034**	18862.514	!	191707.118	150s
Coral	+	-	**3.380e+09**	+	!	351.354	4000s
Cut	37080.667	-	**29306.667**	37336.333	41758.333	27584.333	4000s
ECOGCNN	+	-	**755952.240**	+	!	755952.240	4000s
HEM_knapsack	**422.600**	**422.600**	**422.600**	420.600	0.000	0.000	100s
HEM_mis	**228.800**	216.600	**228.800**	**228.800**	192.400	0.000	100s
HEM_setcover	**231.600**	231.800	**231.600**	+	256.000	**231.600**	100s
HEM_corlat	+	-	**251.000**	+	252.124	134.000	100s
HEM_mik	+	-	**-62758.400**	+	-48692.200	-62758.400	100s
item_placement	16.511	-	5.531	23.832	!	**4.985**	4000s
load_balancing	+	-	**708.800**	+	726.500	726.700	1000s
anonymous	+	-	**245680.647**	+	3.130e+06	292932.705	4000s
Nexp	1.175e+08	-	**1.163e+08**	+	!	1.163e+08	4000s
Transportation	1.306e+06	-	1.248e+06	+	**1.244e+06**	4.499e+06	4000s
vary_bounds_s1	12890.400	-	12384.800	13054.400	**12381.800**	**12381.800**	400s
vary_bounds_s2	+	-	355.000	+	**351.000**	**351.000**	1000s
vary_bounds_s3	+	-	355.000	+	**351.000**	**351.000**	1000s
vary_matrix_s1	62.660	-	**61.594**	62.922	!	61.594	100s
vary_matrix_rhs_bounds_s1	+	-	**2.002e+09**	2.002e+09	!	2.002e+09	100s
vary_matrix_rhs_bounds_obj_s1	-47567.400	-	**-51638.297**	-27727.006	-51637.903	-51638.285	100s
vary_obj_s1	8633.600	**8625.400**	**8625.400**	8629.000	**8625.400**	8625.400	100s
vary_obj_s2	+	-	**1169.488**	+	!	1169.489	150s
vary_obj_s3	-2180.098	-	**-2180.098**	+	!	-2180.098	100s
vary_rhs_s1	+	-	**-349.464**	+	-349.464	-299.585	100s
vary_rhs_s2	-17167.332	-	-17168.351	-17168.245	!	**-17168.351**	100s
vary_rhs_s3	+	57285.200	**57258.800**	+	!	71352.600	100s
vary_rhs_s4	-17164.917	-	-17166.247	**-17166.460**	!	-17166.460	100s
vary_rhs_obj_s1	+	-	**-179337.619**	+	!	-179336.972	600s
vary_rhs_obj_s2	-807901.267	-	**-807964.313**	-807870.988	!	-807962.296	100s
Aclib	+	-	**82427.000**	82427.000	506721.006	121733.279	100s
fc.data	+	-	**378.600**	378.600	**378.600**	**378.600**	100s
nn_verification	+	-	**-8.251**	+	!	-25.114	100s

those of classical solvers. However, its performance deteriorates in harder instances such as `MIS_hard` and `MVC_hard`, indicating limitations in its ability to scale effectively. In contrast, **ACP** demonstrates robustness across both small-scale and large-scale problems, achieving the best objective function values in instances like `MIS_medium` and `SC_hard`. This suggests that ACP's adaptive neighborhood adjustment mechanism is well-suited for complex problem structures.

LIH and **MIH**, which use lightweight heuristics, show mixed performance. While they perform reasonably well in simpler instances such as `HEM_knapsack` and `HEM_mis`, they struggle with more challenging problems like `MVC_hard` and `SC_medium`, where their solutions are often suboptimal. **LNS**, on the other hand, exhibits strong scalability and achieves competitive results in large-scale instances such as `MVC_hard` and `SC_hard`, where it outperforms GLNS and even some classical solvers. This highlights the effectiveness of its large-scale neighborhood search strategy in exploring the solution space efficiently.

RINS stands out for its ability to integrate exact and heuristic methods, achieving the best objective function values in several instances, including `MIS_hard` and `vary_bounds_s2`. Its hybrid approach leverages the strengths of classical solvers while maintaining the computational efficiency of heuristics, making it particularly effective for challenging instances. However, its performance is less consistent in simpler problems, where it is often outperformed by other heuristics like ACP and LNS.

The GAP estimations in Table 10 reveal further insights into the quality of solutions produced by these heuristics. **RINS** achieves the smallest GAP in many instances, such as `MIS_hard` and `vary_bounds_s1`, indicating its ability to find high-quality solutions close to the optimal. **LNS** also demonstrates strong performance in terms of GAP, particularly in larger and more complex instances. In contrast, **LIH** and **MIH** show larger GAP values in harder problems, confirming their limitations in producing near-optimal solutions under such conditions.

When analyzing performance by problem type, heuristics like **ACP** and **RINS** excel in structured instances such as `SC_hard` and `vary_rhs_s2`, where their adaptive search strategies allow them to outperform other methods. For simpler problems like `HEM_knapsack` and `HEM_mis`, all heuristics achieve similar results with negligible GAP values, reflecting the relative ease of these instances. However, in specialized problems like `Cut`, **RINS** demonstrates a clear advantage over other heuristics, achieving the smallest GAP and the best objective function value.

In summary, heuristic approaches provide a balance between solution quality and computational efficiency, with each method exhibiting unique strengths. **RINS** and **ACP** emerge as the most competitive heuristics overall, excelling in both objective function

Table 12: Gap estimation of machine learning-based baselines. The bold numbers indicate the best results achieved by the solvers. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. -represents MILP problems that cannot be solved by the IP framework, GNN&GBDT.

Problem	Learn2Branch	GNN&GBDT	Predict&Search	Hybrid-Learn2Branch	GNN-MILP	Nerual Diving	Time
MIS_easy	+	**0.113**	0.151	+	0.446	583.007	600s
MIS_medium	+	0.115	**0.092**	+	!	1908.708	2000s
MIS_hard	+	**0.117**	!	+	!	!	4000s
MVC_easy	+	0.090	0.113	+	0.263	**0.073**	600s
MVC_medium	+	**0.098**	!	+	!	0.129	4000s
MVC_hard	+	**0.095**	!	+	!	!	4000s
SC_easy	+	0.039	**0.022**	+	0.117	0.024	600s
SC_medium	+	**0.985**	!	+	!	0.986	2000s
SC_hard	+	**0.989**	!	+	!	!	4000s
MIPlib	0.336	-	**1.233e-06**	0.194	!	1.420	150s
Coral	+	-	**16644.947**	+	!	27175.456	4000s
Cut	0.578	-	**0.157**	0.558	0.506	0.638	4000s
ECOGCNN	+	-	**0.251**	+	!	26992.105	4000s
HEM_knapsack	**0.000**	**0.000**	**0.000**	0.005	!	+∞	100s
HEM_mis	**0.000**	0.057	**0.000**	**0.000**	0.191	+∞	100s
HEM_setcover	**0.000**	0.001	**0.000**	+	0.097	**0.000**	100s
HEM_corlat	+	-	**0.000**	+	0.005	1.755	100s
HEM_mik	+	-	**3.001e-15**	+	0.287	3.082e-15	100s
item_placement	4.168e+07	-	0.659	2.000e+19	!	**0.592**	4000s
load_balancing	+	-	**0.003**	+	0.027	0.042	1000s
anonymous	+	-	**0.291**	+	1.829	0.811	4000s
Nexp	0.163	-	**0.076**	+	!	1.696e+06	4000s
Transportation	0.272	-	0.157	+	**0.155**	0.306	4000s
vary_bounds_s1	0.152	-	2.737e-4	0.143	**1.729e-05**	**1.729e-05**	400s
vary_bounds_s2	+	-	0.011	+	**0.000**	**0.000**	1000s
vary_bounds_s3	+	-	0.011	+	**0.000**	**0.000**	1000s
vary_matrix_s1	0.400	-	**1.391e-16**	0.551	!	6.046e-16	100s
vary_matrix_rhs_bounds_s1	+	-	**3.158e-05**	0.012	!	0.082	100s
vary_matrix_rhs_bounds_obj_s1	0.095	-	**3.810e-05**	2.000e+19	4.584e-05	3.834e-05	100s
vary_obj_s1	0.005	**0.000**	**0.000**	0.003	**0.000**	0.001	100s
vary_obj_s2	+	-	**1.748e-06**	+	!	1.420	150s
vary_obj_s3	0.067	-	**0.000**	+	!	0.104	100s
vary_rhs_s1	+	-	**0.000**	+	**0.000**	0.215	100s
vary_rhs_s2	0.002	-	1.299e-05	**0.000**	!	1.299e-05	100s
vary_rhs_s3	+	0.001	**6.208e-05**	+	!	0.236	100s
vary_rhs_s4	0.003	-	3.648e-05	**0.000**	!	7.487e-05	100s
vary_rhs_obj_s1	+	-	**8.362e-05**	+	!	8.707e-05	600s
vary_rhs_obj_s2	0.001	-	**6.921e-05**	0.001	!	7.166e-05	100s
Aclib	+	-	**2.244e-06**	0.004	0.836	1.232	100s
fc.data	+	-	**0.000**	0.012	**0.000**	**0.000**	100s
nn_verification	+	-	**5.621e-05**	+	!	1.109	100s

values and GAP across a wide range of problem instances. **LNS** demonstrates strong scalability, making it particularly effective for large-scale problems, while **GLNS** and **LIH** are better suited for simpler instances. These results highlight the complementary nature of heuristic methods and their potential to address specific challenges in MILP problem-solving. The benchmarking study establishes a solid foundation for comparing these heuristics with classical solvers and machine learning-based methods in subsequent sections.

C.5.3 Machine Learning-Based Methods. Machine learning-based approaches have emerged as a promising alternative for solving MILP problems by leveraging data-driven models to improve or replace traditional optimization components. These methods aim to learn from historical instances or optimization processes, offering potential advantages in scalability and adaptability. In this study, we evaluate six machine learning-based methods: **Learn2Branch**, **GNN&GBDT**, **Predict&Search**, **Hybrid-Learn2Branch**, **GNN-MILP**, and **Neural Diving**. The results, presented in Table 11 (objective function values) and Table 12 (GAP estimations), shed light on their performance across diverse problem instances.

From Table 11, it is evident that machine learning methods exhibit highly variable performance depending on the problem instance. **Learn2Branch**, while conceptually innovative, struggles to scale to larger instances such as `MIS_hard` and `MVC_hard`, where it fails to provide results within a reasonable time frame. Similarly, **GNN&GBDT** shows limitations in handling complex instances, as indicated by its inability to solve problems such as `SC_hard` and `vary_bounds_s2`. However, it achieves competitive results in smaller and medium-scale problems like `MIS_medium` and `MVC_easy`, where its data-driven branching strategies are effective.

Predict&Search demonstrates a notable advantage in structured problems, achieving the best objective function values in instances like `SC_medium` and `vary_bounds_s3`. Its ability to integrate predictive models with heuristic search allows it to explore the solution space efficiently. **Hybrid-Learn2Branch**, which combines traditional optimization techniques with learning-based methods, shows promising results in several instances, such as `MIPLib` and `Cut`, where it produces high-quality solutions comparable to classical solvers.

GNN-MILP and **Neural Diving**, which rely heavily on graph neural networks and neural heuristics, respectively, display mixed performance. While **GNN-MILP** achieves the best results in some instances, such as `Cut`, it fails to solve a significant number of problems due to limitations in its current implementation. **Neural Diving**, on the other hand, performs well in certain structured problems like `vary_obj_s1` and `HEM_knapsack`, achieving the smallest GAP values and demonstrating its capacity to refine solutions effectively.

Table 13: The standard deviations of objective function value of solver and heuristic baselines. The bold numbers indicate the best results achieved by the solvers. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. -represents MILP problems that cannot be solved by the IP framework, GNN&GBDT.

Problem	Gurobi	SCIP	CPLEX	GLNS	ACP	LIH	MIH	LNS	RINS
MIS_easy	24.038	32.461	24.042	22.916	22.585	28.387	17.696	20.990	25.419
MIS_medium	21.688	65.470	74.563	42.086	36.982	45.460	48.256	36.343	41.596
MIS_hard	135.162	109.108	201.787	138.940	1016.420	10487.384	16292.748	359.211	304.515
MVC_easy	15.437	20.986	23.485	15.294	18.107	24.782	20.671	17.216	18.313
MVC_medium	69.776	145.855	789.424	78.105	76.664	48.319	83.410	87.901	81.702
MVC_hard	229.946	191.881	91705.231	187.002	1168.383	4856.392	15127.128	242.641	247.772
SC_easy	183.543	22.321	16.744	22.758	16.102	1762.987	1102.241	29.464	94.006
SC_medium	68.306	119.404	56.618	65.325	55.094	54.688	671.294	152.293	54.688
SC_hard	327.741	681.758	!	1085.688	3739.504	237.732	2496.007	663.285	456.309
MIPlib	37265.995	37265.675	37265.995	39920.872	37265.853	1.196e+07	1.696e+07	1.696e+07	72830.393
Coral	10033.250	2.245e+08	15153.675	1.237e+09	3.706e+08	8.944e+09	8.944e+09	8.944e+09	8.944e+09
Cut	23807.714	37168.697	17196.370	26797.947	26851.840	24041.473	27891.189	16320.429	27124.440
ECOGCNN	1.309e+06	1.309e+06	1.309e+06	1.313e+06	1.311e+06	1.311e+06	1.326e+06	1.314e+06	1.326e+06
HEM_knapsack	14.293	14.293	14.293	14.293	14.293	12.621	15.786	12.104	15.786
HEM_mis	4.087	4.087	4.087	4.393	4.087	9.274	10.281	3.912	4.930
HEM_setcover	31.966	31.966	31.966	30.627	31.966	31.966	116.858	139.346	32.337
HEM_corlat	151.013	151.013	151.013	152.344	151.013	151.721	154.834	151.721	152.472
HEM_mik	16320.035	16320.035	16320.035	16198.720	15757.903	14604.395	14604.395	16320.035	14604.395
item_placement	1.170	2.574	1.865	3.253	2.932	31.311	31.311	32.553	31.311
load_balancing	22.900	22.568	22.900	21.781	22.867	52.650	52.650	52.650	52.650
anonymous	184649.651	1.319e+06	199121.999	840294.450	484864.719	367257.658	1.520e+06	1.519e+06	1.180e+06
Nexp	2.346e+08	2.352e+08	2.346e+08	2.368e+08	2.346e+08	2.360e+08	3.891e+08	3.034e+08	2.400e+08
Transportation	13287.596	17717.764	19644.600	41174.332	23052.206	16604.906	125931.259	125931.259	16440.097
vary_bounds_s1	968.206	967.275	968.206	1605.772	968.206	1955.014	3278.601	1683.083	2412.291
vary_bounds_s2	0.000	0.000	0.000	8.173	0.000	0.000	0.000	3.493	0.000
vary_bounds_s3	0.000	0.000	0.000	0.447	5.842e-07	0.000	0.000	4.604	0.000
vary_matrix_s1	1.010	0.661	1.010	0.990	1.006	2.279	2.279	2.029	2.279
vary_matrix_rhs_bounds_s1	1.466e+08	1.466e+08	1.466e+08	6.636e+08	4.475e+09	3.074e+09	3.074e+09	3.074e+09	3.074e+09
vary_matrix_rhs_bounds_obj_s1	24157.854	16278.878	24156.798	14826.419	20602.783	3921.385	3921.385	3921.385	3921.385
vary_obj_s1	113.742	112.374	113.742	117.235	113.742	113.898	88.898	81.726	113.948
vary_obj_s2	1253.171	1252.775	1253.171	2713.451	1253.168	2463.035	2463.035	2463.035	2463.035
vary_obj_s3	393.906	497.695	6.552	3815.850	3799.435	3337.141	3716.218	3664.684	3695.642
vary_rhs_s1	6.552	16.133	6.552	7.924	21.633	40.145	11150.580	70.655	27.906
vary_rhs_s2	1.890	0.645	2.100	62.879	8.524	0.154	0.152	0.152	0.152
vary_rhs_s3	12448.472	12445.817	12448.177	12446.838	12448.015	12448.797	12522.711	10644.555	12447.544
vary_rhs_s4	3.372	3.356	3.634	166.246	19.668	0.980	0.980	0.980	0.980
vary_rhs_obj_s1	45777.795	45530.734	45781.289	43189.537	45405.629	43342.644	43342.644	43342.644	43342.644
vary_rhs_obj_s2	394749.995	394800.883	394758.618	209713.995	308900.252	185521.612	185521.612	185521.612	185521.612
Aclib	99060.560	99060.560	99060.560	99057.608	99592.097	99302.016	139221.664	173439.252	99666.782
fc.data	219.889	219.889	219.889	270.879	219.889	447.244	949.217	866.149	676.331
nn_verification	10.199	10.259	10.199	10.021	10.021	10.297	10.297	10.297	4.644

The GAP estimations in Table 12 further highlight the strengths and weaknesses of these methods. **Predict&Search** and **Hybrid-Learn2Branch** achieve the smallest GAP values in many instances, underscoring their ability to produce high-quality solutions. In contrast, methods like **Learn2Branch** and **GNN&GBDT** exhibit larger GAP values or fail to generate results for more complex problems, revealing scalability challenges. **Neural Diving** consistently achieves near-optimal solutions in simpler instances, reflecting its effectiveness in focused problem-solving scenarios.

When analyzing performance by problem type, machine learning methods generally excel in structured and smaller-scale problems. For example, in `HEM` problems such as `HEM_knapsack` and `HEM_setcover`, all methods achieve identical or near-identical results with negligible GAP values, emphasizing their suitability for such instances. However, for larger and more complex problems like `SC_hard` and `vary_rhs_obj_s2`, traditional solvers and heuristic methods still outperform machine learning approaches due to their maturity and robustness.

In summary, machine learning-based approaches represent a valuable addition to the MILP-solving toolkit, offering unique advantages in specific scenarios. **Predict&Search** and **Hybrid-Learn2 Branch** emerge as the most competitive methods, demonstrating strong performance in both objective function values and GAP estimations. However, scalability remains a significant challenge for methods like **Learn2Branch** and **GNN&GBDT**, which struggle to handle larger instances effectively. These results highlight the potential of machine learning methods while also underscoring the need for further research and development to improve their scalability and robustness.

C.5.4 Summary of Benchmarking Study. This benchmarking study provides a comprehensive evaluation of classical solvers, heuristic approaches, and machine learning-based methods for solving MILP problems. By systematically comparing their performance across diverse problem instances, we gain valuable insights into the strengths and limitations of each category, highlighting their practical applicability and areas for future improvement.

Classical solvers, represented by **Gurobi**, **SCIP**, and **CPLEX**, remain the gold standard for MILP-solving due to their robustness and generalizability. Gurobi and CPLEX, in particular, excel in solving large-scale and complex problems, as well as in achieving near-optimal solutions with minimal GAP values. SCIP, while competitive in smaller instances, faces challenges in scaling to harder problems. The primary advantage of classical solvers lies in their maturity and efficiency, which make them suitable for a broad range of MILP tasks. However, their reliance on computationally

Table 14: The standard deviations of gap estimation of solver and heuristic baselines. The bold numbers indicate the best results achieved by the solvers. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. -represents MILP problems that cannot be solved by the IP framework, GNN&GBDT.

Problem	Gurobi	SCIP	CPLEX	GLNS	ACP	LIH	MIH	LNS	RINS
MIS_easy	0.001	0.008	0.002	0.002	0.002	0.016	0.014	0.010	0.011
MIS_medium	0.001	0.006	0.005	0.002	0.002	0.003	0.004	0.002	0.002
MIS_hard	0.001	0.677	0.514	0.001	0.006	0.075	0.161	0.002	0.004
MVC_easy	0.001	0.003	0.004	0.001	0.002	0.002	0.002	0.002	0.002
MVC_medium	0.001	0.005	0.024	0.001	0.001	0.001	0.001	0.002	0.003
MVC_hard	0.001	1.067	0.398	0.000	0.004	0.013	0.033	0.001	0.000
SC_easy	0.048	0.000	0.001	0.003	0.001	0.274	0.123	0.005	0.028
SC_medium	0.000	4.472e+19	1.366e-05	0.000	0.000	0.000	0.000	0.000	0.000
SC_hard	5.880e-05	123650.533	!	9.830e-05	0.000	3.825e-05	0.012	0.000	6.823e-05
MIPlib	4.941e-05	0.061	5.732e-05	0.214	0.001	0.452	0.456	0.456	0.177
Coral	75323.856	4.880e+19	1.512e+10	80608.590	80608.670	94490.184	93251.879	93251.887	93251.879
Cut	0.216	0.725	0.132	0.178	0.246	0.103	0.114	0.094	0.112
ECOGCNN	0.432	7.958	0.502	0.461	0.432	0.469	0.480	0.479	0.471
HEM_knapsack	0.000	0.000	0.000	0.000	0.000	0.009	0.013	0.012	0.013
HEM_mis	0.000	0.000	0.000	0.005	0.000	0.086	0.125	0.005	0.005
HEM_setcover	0.000	0.000	0.000	0.010	0.000	0.000	0.009	0.014	0.002
HEM_corlat	0.000	0.000	0.000	0.029	0.000	0.015	0.076	0.015	0.032
HEM_mik	6.892e-15	0.000	2.714e-06	0.008	0.014	0.028	0.028	6.892e-15	0.028
item_placement	0.170	6.667e+07	0.083	0.086	0.093	0.001	0.001	0.001	0.001
load_balancing	9.120e-05	0.009	0.000	0.004	0.001	0.071	0.071	0.071	0.071
anonymous	0.106	4.111	0.185	0.021	0.272	0.061	0.009	0.009	0.083
Nexp	0.157	0.277	0.160	0.149	0.147	0.167	0.266	0.144	0.145
Transportation	0.001	0.004	0.010	0.019	0.014	0.004	0.014	0.014	0.004
vary_bounds_s1	3.866e-05	0.014	5.183e-05	0.089	3.866e-05	0.073	0.065	0.030	0.072
vary_bounds_s2	0.000	0.003	0.000	0.017	0.000	0.000	0.000	0.010	0.000
vary_bounds_s3	0.000	0.002	0.000	0.001	1.664e-09	0.000	0.000	0.012	0.000
vary_matrix_s1	0.000	0.078	3.368e-05	0.002	0.001	0.019	0.019	0.016	0.019
vary_matrix_rhs_bounds_s1	3.521e-05	6.045e-06	3.894e-05	0.099	0.174	0.008	0.008	0.008	0.008
vary_matrix_rhs_bounds_obj_s1	5.007e-05	0.224	2.930e-06	1.055	0.075	3.076	3.076	3.076	3.076
vary_obj_s1	0.000	0.001	0.000	0.001	0.000	0.001	0.011	0.004	0.001
vary_obj_s2	4.395e-07	9.156	0.013	0.161	8.491e-07	0.102	0.102	0.102	0.102
vary_obj_s3	0.000	5.477e+19	0.000	3.050	17.884	3.032	2.940	2.922	3.037
vary_rhs_s1	0.001	0.074	0.000	0.778	0.079	0.291	4.553e-05	2.601	0.658
vary_rhs_s2	1.937e-05	0.000	7.638e-06	0.004	0.001	0.000	0.000	0.000	0.000
vary_rhs_s3	1.982e-05	6.281e-05	1.753e-05	2.639e-05	9.841e-06	2.280e-05	0.040	0.064	2.493e-05
vary_rhs_s4	3.500e-05	0.000	8.529e-07	0.010	0.001	0.001	0.001	0.001	0.001
vary_rhs_obj_s1	1.487e-05	0.006	1.353e-05	0.011	0.005	0.011	0.011	0.011	0.011
vary_rhs_obj_s2	2.178e-05	0.000	3.588e-06	0.176	0.067	0.212	0.212	0.212	0.212
Aclib	7.093e-06	0.000	6.138e-06	0.002	0.004	0.007	0.157	0.103	0.005
fc.data	0.000	0.000	0.000	0.234	1.458e-08	0.070	0.062	0.060	0.063
nn_verification	3.266e-05	0.227	3.713e-05	0.167	0.167	0.157	0.157	0.157	0.672

intensive techniques can limit their applicability in real-time or resource-constrained scenarios.

Heuristic approaches, including **GLNS**, **ACP**, **LIH**, **MIH**, **LNS**, and **RINS**, offer a practical alternative to exact solvers by providing approximate solutions in a computationally efficient manner. Methods like **RINS** and **ACP** demonstrate strong performance in complex and structured problems, where their adaptive and hybrid strategies allow them to explore solution spaces effectively. **LNS**, with its scalability, excels in large-scale instances, while simpler heuristics like **GLNS** are better suited for smaller and easier problems. Despite their efficiency, heuristics often struggle to match the solution quality of classical solvers, particularly in achieving smaller GAP values in harder instances.

Machine learning-based methods, represented by **Learn2Branch**, **GNN&GBDT**, **Predict&Search**, **Hybrid-Learn2Branch**, **GNN-MILP**, and **Neural Diving**, bring a novel perspective to MILP-solving by leveraging data-driven models. Methods like **Predict& Search** and **Hybrid-Learn2Branch** emerge as the most competitive, achieving high-quality solutions in structured and medium-scale problems. However, scalability remains a significant challenge for many learning-based approaches, as evidenced by the inability of some methods to handle large or complex instances. While machine learning methods show great promise, their current limitations highlight the need for further development, particularly in improving their robustness, efficiency, and applicability to diverse MILP tasks.

The results of this study also emphasize the importance of MILPBench as a standardized and reproducible benchmarking platform. By encompassing a wide range of problem instances with varying complexity and structure, MILPBench enables a fair and systematic comparison of different methods. Its versatility facilitates the evaluation of traditional, heuristic, and learning-based approaches, providing a holistic view of the current state of MILP-solving. Furthermore, MILPBench serves as a valuable resource for researchers and practitioners, offering a foundation for developing and testing new algorithms while addressing the challenges identified in this study.

In conclusion, the benchmarking study underscores the complementary nature of the three categories of methods. Classical solvers provide robust and reliable solutions for a wide range of problems, heuristics offer practical alternatives for specific scenarios, and machine learning approaches hold the potential to revolutionize MILP-solving in the future. By leveraging the strengths of each method and addressing their respective limitations, future research can pave the way for more efficient and scalable solutions to MILP

Table 15: The error bar's width of objective function value of solver and heuristic baselines. The bold numbers indicate the best results achieved by the solvers. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. -represents MILP problems that cannot be solved by the IP framework, GNN&GBDT.

Problem	Gurobi	SCIP	CPLEX	GLNS	ACP	LIH	MIH	LNS	RINS
MIS_easy	36.744	49.506	29.424	34.602	34.167	34.947	21.654	31.740	39.485
MIS_medium	30.610	82.564	122.522	59.613	54.273	77.328	85.540	47.509	57.217
MIS_hard	197.151	159.777	313.529	222.354	1648.756	18635.409	17913.076	554.049	489.697
MVC_easy	21.318	26.287	33.564	18.273	27.614	28.946	34.093	29.319	28.368
MVC_medium	101.297	170.399	882.710	113.369	112.325	74.937	98.030	144.798	125.685
MVC_hard	303.527	263.670	163638.551	262.730	1719.396	5256.134	27059.879	286.492	281.978
SC_easy	326.526	31.308	25.735	29.414	23.812	1947.455	1207.848	34.397	114.152
SC_medium	99.399	174.722	73.114	82.836	79.540	85.960	730.835	210.132	85.960
SC_hard	493.936	1122.909	!	1415.595	6466.733	380.188	2815.709	1072.820	583.556
MIPlib	55897.295	55896.809	55897.295	59879.697	55897.087	1.793e+07	2.544e+07	2.544e+07	109244.838
Coral	22753.221	5.091e+08	34365.272	2.805e+09	8.404e+08	2.028e+10	2.028e+10	2.028e+10	2.028e+10
Cut	27475.333	42903.333	19837.000	30891.364	30993.000	27696.667	32142.333	18821.667	31275.667
ECOGCNN	1.512e+06	1.512e+06	1.512e+06	1.516e+06	1.514e+06	1.514e+06	1.531e+06	1.517e+06	1.531e+06
HEM_knapsack	22.600	22.600	22.600	22.600	22.600	20.400	26.800	20.000	26.800
HEM_mis	6.800	6.800	6.800	7.600	6.800	11.000	12.200	6.600	8.600
HEM_setcover	47.400	47.400	47.400	46.000	47.400	47.400	199.400	181.400	48.200
HEM_corlat	269.000	269.000	269.000	271.200	269.000	270.200	275.000	270.200	271.400
HEM_mik	29114.400	29114.400	29114.400	28879.200	28115.600	26032.200	26032.200	29114.400	26032.200
item_placement	2.507	3.998	4.326	5.991	6.442	60.436	60.436	57.629	60.436
load_balancing	41.200	42.000	41.200	40.200	41.700	128.500	128.500	128.500	128.500
anonymous	258504.164	1.947e+06	270867.857	1.108e+06	677494.474	503517.810	1.969e+06	1.968e+06	1.560e+06
Nexp	4.181e+08	4.191e+08	4.181e+08	4.218e+08	4.181e+08	4.206e+08	6.950e+08	5.415e+08	4.276e+08
Transportation	22325.400	26139.800	30560.400	49015.800	29393.800	21579.600	161605.000	161605.000	22019.200
vary_bounds_s1	1130.200	1256.800	1130.200	2502.600	1130.200	2372.800	5027.200	2174.400	3314.400
vary_bounds_s2	0.000	0.000	0.000	14.600	0.000	0.000	0.000	4.800	0.000
vary_bounds_s3	0.000	0.000	0.000	0.800	1.045e-06	0.000	0.000	7.200	0.000
vary_matrix_s1	1.545	0.762	1.545	1.497	1.540	3.505	3.505	3.077	3.505
vary_matrix_rhs_bounds_s1	2.115e+08	2.115e+08	2.115e+08	1.177e+09	7.986e+09	4.578e+09	4.578e+09	4.578e+09	4.578e+09
vary_matrix_rhs_bounds_obj_s1	34559.345	28532.650	34558.737	20423.643	32648.555	5830.361	5830.361	5830.361	5830.361
vary_obj_s1	184.600	181.000	184.600	190.000	184.600	182.600	130.400	129.200	182.200
vary_obj_s2	2120.358	2118.823	2120.358	3292.140	2120.353	3543.945	3543.945	3543.945	3543.945
vary_obj_s3	598.545	650.142	10.216	6178.054	6588.985	5325.024	6049.489	6199.815	6033.224
vary_rhs_s1	10.216	28.304	10.216	12.824	34.008	63.424	19916.064	111.184	37.512
vary_rhs_s2	3.380	1.149	3.746	74.509	13.005	0.275	0.272	0.272	0.272
vary_rhs_s3	21894.200	21889.400	21893.400	21891.400	21893.400	21894.400	22136.000	18780.800	21892.800
vary_rhs_s4	3.964	3.979	4.648	278.784	26.118	1.163	1.163	1.163	1.163
vary_rhs_obj_s1	63272.867	63899.797	63281.191	60468.907	63478.284	60897.253	60897.253	60897.253	60897.253
vary_rhs_obj_s2	694335.306	694415.705	694350.350	352827.451	537044.786	307580.252	307580.252	307580.252	307580.252
Aclib	178411.999	178412.000	178412.000	178388.200	180249.601	178052.100	250024.200	291294.400	178222.400
fc.data	384.400	384.400	384.400	307.600	384.400	640.000	1512.000	1390.400	820.000
nn_verification	24.581	24.720	24.581	24.558	24.558	24.553	24.553	24.553	8.678

problems, ultimately expanding the scope of their practical applications.

C.6 Stability Analysis of Algorithms

To further evaluate the reliability and robustness of different algorithms, we conducted a stability analysis focusing on the variability of their performance. This analysis is based on eight tables summarizing the standard deviation (std) and error bar width for the objective function values and gap estimates across classical solvers, heuristic methods, and ML-based approaches.

The standard deviation (`std`) is calculated using the unbiased estimator:

$$\text{std} = \sqrt{\frac{\sum (x_i - \overline{x})^2}{n-1}},$$

where x_i represents individual results, $\overline{x}$ is the mean, and n is the sample size. A smaller standard deviation indicates less variability and greater consistency across problem instances.

The error bar width is defined as the maximum absolute deviation from the mean:

$$\text{width} = \max(|x_i - \overline{x}|).$$

This metric captures the worst-case deviation, providing additional insights into the stability of each algorithm.

The tables are divided into two main categories: (1) classical solvers and heuristic methods, and (2) ML-based approaches. Each category includes four tables, measuring the variability of objective function values and gap estimates both in terms of standard deviation and error bar width. These metrics allow us to systematically compare the stability of different algorithm types and highlight their strengths and weaknesses.

In the following sections, we analyze the results for classical solvers and heuristic methods (§C.6.1), ML-based approaches (§C.6.2), and summarize the overall findings (§C.6.3).

C.6.1 Stability Analysis of Classical Solvers and Heuristic Methods. This section examines the stability of classical solvers (Gurobi, SCIP, and CPLEX) and heuristic methods (GLNS, ACP, LIH, MIH, LNS, and RINS) based on the standard deviation and error bar width of their objective function values and gap estimates. The results are summarized in Tables 13, 14, 15, and 16.

The standard deviation (`std`) results reveal notable differences in the stability of classical solvers and heuristic methods across problem instances. Among classical solvers, Gurobi consistently achieves the lowest variability for both objective function values

Table 16: The error bar's width of gap estimation of solver and heuristic baselines. The bold numbers indicate the best results achieved by the solvers. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. -represents MILP problems that cannot be solved by the IP framework, GNN&GBDT.

Problem	Gurobi	SCIP	CPLEX	GLNS	ACP	LIH	MIH	LNS	RINS
MIS_easy	0.002	0.010	0.003	0.002	0.002	0.019	0.019	0.014	0.016
MIS_medium	0.001	0.009	0.008	0.002	0.003	0.005	0.006	0.003	0.003
MIS_hard	0.001	1.019	0.919	0.001	0.010	0.134	0.177	0.003	0.005
MVC_easy	0.002	0.004	0.004	0.001	0.003	0.002	0.003	0.002	0.003
MVC_medium	0.001	0.007	0.028	0.001	0.001	0.001	0.002	0.003	0.005
MVC_hard	0.001	1.708	0.605	0.001	0.005	0.014	0.059	0.002	0.001
SC_easy	0.086	0.000	0.002	0.004	0.001	0.306	0.134	0.007	0.031
SC_medium	0.000	8.000e+19	2.094e-05	0.001	0.001	0.000	0.000	0.000	0.000
SC_hard	8.468e-05	218472.237	!	0.000	0.001	5.861e-05	0.014	0.000	0.000
MIPlib	7.412e-05	0.059	5.010e-05	0.222	0.001	0.678	0.685	0.684	0.256
Coral	170818.450	7.143e+19	3.429e+10	182803.100	182803.280	214283.595	211475.385	211475.402	211475.385
Cut	0.248	0.836	0.148	0.197	0.283	0.111	0.125	0.105	0.126
ECOGCNN	0.499	9.189	0.580	0.533	0.498	0.541	0.554	0.553	0.544
HEM_knapsack	0.000	0.000	0.000	0.000	0.000	0.016	0.017	0.022	0.017
HEM_mis	0.000	0.000	0.000	0.005	0.000	0.103	0.154	0.008	0.009
HEM_setcover	0.000	0.000	0.000	0.014	0.000	0.000	0.015	0.018	0.003
HEM_corlat	0.000	0.000	0.000	0.051	0.000	0.027	0.136	0.027	0.056
HEM_mik	1.233e-14	0.000	3.866e-06	0.014	0.018	0.047	0.047	1.233e-14	0.047
item_placement	0.347	1.889e+08	0.200	0.148	0.173	0.003	0.003	0.003	0.003
load_balancing	0.000	0.015	0.001	0.008	0.001	0.159	0.159	0.159	0.159
anonymous	0.154	6.054	0.226	0.023	0.334	0.088	0.010	0.010	0.098
Nexp	0.279	0.495	0.285	0.260	0.262	0.246	0.412	0.192	0.251
Transportation	0.001	0.006	0.015	0.027	0.019	0.006	0.020	0.020	0.006
vary_bounds_s1	6.916e-05	0.022	5.663e-05	0.105	6.916e-05	0.106	0.090	0.038	0.088
vary_bounds_s2	0.000	0.004	0.000	0.031	0.000	0.000	0.000	0.013	0.000
vary_bounds_s3	0.000	0.003	0.000	0.002	2.977e-09	0.000	0.000	0.019	0.000
vary_matrix_s1	0.000	0.111	5.765e-05	0.003	0.001	0.027	0.027	0.022	0.027
vary_matrix_rhs_bounds_s1	5.929e-05	1.020e-05	6.906e-05	0.172	0.302	0.011	0.011	0.011	0.011
vary_matrix_rhs_bounds_obj_s1	5.711e-05	0.400	4.277e-06	1.693	0.113	4.569	4.569	4.569	4.569
vary_obj_s1	0.000	0.001	0.000	0.001	0.000	0.001	0.020	0.007	0.001
vary_obj_s2	7.860e-07	12.556	0.020	0.203	1.407e-06	0.180	0.180	0.180	0.180
vary_obj_s3	0.000	6.000e+19	0.000	5.245	31.517	5.277	5.121	4.975	5.288
vary_rhs_s1	0.001	0.133	0.000	1.157	0.130	0.499	7.351e-05	3.348	0.904
vary_rhs_s2	3.034e-05	0.001	1.363e-05	0.005	0.001	0.000	0.000	0.000	0.000
vary_rhs_s3	3.085e-05	0.000	1.986e-05	4.208e-05	1.125e-05	3.754e-05	0.069	0.112	3.355e-05
vary_rhs_s4	4.838e-05	0.000	1.253e-06	0.017	0.001	0.001	0.001	0.001	0.001
vary_rhs_obj_s1	2.106e-05	0.008	2.410e-05	0.014	0.008	0.016	0.016	0.016	0.016
vary_rhs_obj_s2	2.637e-05	0.000	6.412e-06	0.314	0.118	0.378	0.378	0.378	0.378
Aclib	2.019e-05	0.000	1.150e-05	0.005	0.008	0.019	0.250	0.166	0.009
fc.data	0.000	0.000	0.000	0.399	2.608e-08	0.112	0.094	0.092	0.083
nn_verification	5.108e-05	0.604	7.188e-05	0.394	0.394	0.396	0.396	0.396	1.742

and gap estimates, demonstrating high reliability. CPLEX also performs well in structured problems such as `Cut`, but its stability decreases in more complex scenarios like `SC_hard`. In contrast, SCIP exhibits significantly higher variability, particularly in large-scale and time-constrained problems. For heuristic methods, ACP shows relatively low variability in large-scale problems such as `MIS_hard` and `SC_hard`, reflecting its adaptability to these scenarios. However, other heuristics, including GLNS and LNS, display higher variability across diverse problem types, particularly in constrained instances like `HEM_corlat`.

The error bar width results provide additional insights into the stability of the algorithms by quantifying their worst-case deviations. Gurobi again achieves the smallest error bar widths in most instances, confirming its robustness and reliability across problem types. CPLEX follows closely in many cases but demonstrates larger deviations in highly complex problems, which impacts its consistency. SCIP exhibits the largest error bar widths among classical solvers, particularly in datasets like `SC_hard` and `Coral`, where its performance fluctuates significantly. For heuristic methods, ACP and MIH achieve narrower error bars in large-scale problems, indicating more stable performance in these scenarios. In contrast, methods such as LNS and RINS tend to have wider error bars, particularly in constrained problems, suggesting greater sensitivity to problem structure.

In summary, classical solvers, and particularly Gurobi, exhibit superior stability in both objective function values and gap estimates, making them the most reliable choice for diverse MILP tasks. Heuristic methods, while computationally efficient, show mixed stability. ACP and MIH perform consistently in large-scale problems, but other heuristics struggle to maintain stability in highly constrained instances. These findings underscore the importance of balancing computational efficiency with solution stability when evaluating and selecting algorithms for specific problem types.

C.6.2 Stability Analysis of Machine Learning-Based Methods. This section evaluates the stability of machine learning-based methods, including Learn2Branch, GNN&GBDT, Predict&Search, Hybrid-Learn2Branch, GNN-MLP, and Neural Diving, using the standard deviation and error bar width of their objective function values and gap estimates. The results are summarized in Tables 17, 18, 19, and 20.

The standard deviation results (Tables 17 and 18) indicate that machine learning-based methods exhibit varying levels of stability across different problem types. Predict&Search and GNN&GBDT

Table 17: The standard deviations of objective function value of machine learning-based baselines. The bold numbers indicate the best results achieved by the solvers. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. -represents MILP problems that cannot be solved by the IP framework, GNN&GBDT.

Problem	Learn2Branch	GNN&GBDT	Predict&Search	Hybrid-Learn2Branch	GNN-MILP	Nerual Diving
MIS_easy	+	17.892	20.611	+	18.025	35.438
MIS_medium	+	44.335	35.951	+	!	1.774
MIS_hard	+	187.283	!	+	!	!
MVC_easy	+	19.738	32.386	+	36.464	19.417
MVC_medium	+	82.201	!	+	!	94.698
MVC_hard	+	388.435	!	+	!	!
SC_easy	+	18.620	20.861	+	14.912	17.017
SC_medium	+	52.195	!	+	!	79.782
SC_hard	+	29312.555	!	+	!	!
MIPlib	38218.073	-	37266.005	38114.382	!	356922.764
Coral	+	-	8.944e+09	+	!	894.124
Cut	37345.502	-	24236.301	38161.619	13878.449	21495.375
ECOGCNN	+	-	1.309e+06	+	!	1.309e+06
HEM_knapsack	14.293	14.293	14.293	12.542	0.000	0.000
HEM_mis	4.087	7.021	4.087	4.087	7.503	0.000
HEM_setcover	31.966	32.337	31.966	+	26.627	31.966
HEM_corlat	+	-	151.013	+	151.013	99.582
HEM_mik	+	-	16320.035	+	12499.839	16320.035
item_placement	2.906	-	1.110	5.236	!	1.118
load_balancing	+	-	22.900	+	24.546	22.755
anonymous	+	-	192552.907	+	2.586e+06	213294.026
Nexp	2.368e+08	-	2.346e+08	+	!	2.346e+08
Transportation	14835.951	-	14807.757	+	17086.662	7.297e+06
vary_bounds_s1	1194.468	-	965.519	1385.008	968.206	968.206
vary_bounds_s2	+	-	0.000	+	0.000	0.000
vary_bounds_s3	+	-	0.000	+	0.000	0.000
vary_matrix_s1	1.400	-	1.010	0.822	!	1.010
vary_matrix_rhs_bounds_s1	+	-	1.466e+08	1.467e+08	!	1.466e+08
vary_matrix_rhs_bounds_obj_s1	22019.680	-	24157.854	23948.013	24157.864	24157.854
vary_obj_s1	114.321	113.742	113.742	112.628	113.742	113.742
vary_obj_s2	+	-	1253.170	+	!	1253.171
vary_obj_s3	393.906	-	393.906	+	!	393.906
vary_rhs_s1	+	-	6.552	+	6.552	59.756
vary_rhs_s2	1.333	-	1.890	1.958	!	1.890
vary_rhs_s3	+	12418.570	12448.377	+	!	7302.063
vary_rhs_s4	4.020	-	3.373	3.356	!	3.356
vary_rhs_obj_s1	+	-	45778.114	+	!	45777.962
vary_rhs_obj_s2	394761.511	-	394760.816	394756.053	!	394759.678
Aclib	+	-	99060.560	99060.560	587270.422	135835.853
fc.data	+	-	219.889	219.889	219.889	219.889
nn_verification	+	-	10.199	+	!	13.378

generally achieve lower standard deviations in both objective function values and gap estimates for structured problems such as `HEM_knapsack` and `HEM_corlat`, highlighting their consistency in these scenarios. However, methods like Neural Diving and GNN-MLP show higher variability, particularly in large-scale or complex problems such as `SC_hard` and `Nexp`, where their performance fluctuates significantly. Learn2Branch and Hybrid-Learn2Branch achieve intermediate levels of stability, with reduced variability in specific problem types but less consistency across the entire dataset.

The error bar width results (Tables 19 and 20) provide further insights into the worst-case deviations. Predict&Search and GNN&GBDT maintain narrower error bars in structured and small-scale problems, confirming their robustness in these controlled environments. Conversely, Neural Diving and GNN-MLP exhibit wider error bars across several problem types, particularly in large-scale datasets such as `Transportation` and `SC_hard`, indicating significant performance fluctuations. Hybrid-Learn2Branch shows moderate error bar widths, suggesting a balance between stability and adaptability.

Overall, machine learning-based methods demonstrate potential for stability in structured and small-scale problems, but their performance becomes less consistent in large-scale or highly complex scenarios. Predict&Search and GNN&GBDT emerge as relatively stable approaches, while Neural Diving and GNN-MLP exhibit greater sensitivity to problem complexity. These findings underscore the need for further refinement of machine learning methods to enhance their stability and generalizability across diverse problem types.

C.6.3 Summary of Stability Analysis. The stability analysis highlights the contrasting behaviors of classical solvers, heuristic methods, and machine learning-based approaches. Classical solvers, particularly Gurobi, consistently demonstrate superior stability in both objective function values and gap estimates, with low standard deviations and narrow error bar widths across most problem types. This reliability makes them a strong choice for solving diverse MILP problems, especially in scenarios requiring consistent performance.

Heuristic methods, while computationally efficient, show mixed stability. Methods such as ACP and MIH perform well in large-scale problems, displaying moderate variability and controlled worst-case deviations. However, other heuristics, including LNS and RINS, exhibit greater sensitivity to problem structure, leading to higher variability in constrained or complex instances.

Machine learning-based methods demonstrate promising stability in structured and small-scale problems, with Predict&Search and GNN&GBDT emerging as relatively robust approaches. However, their performance becomes less consistent in large-scale or

Table 18: The standard deviations of gap estimation of machine learning-based baselines. The bold numbers indicate the best results achieved by the solvers. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. -represents MILP problems that cannot be solved by the IP framework, GNN&GBDT.

Problem	Learn2Branch	GNN&GBDT	Predict&Search	Hybrid-Learn2Branch	GNN-MILP	Nerual Diving
MIS_easy	+	0.003	0.002	+	0.007	496.683
MIS_medium	+	0.003	0.002	+	!	264.896
MIS_hard	+	0.001	!	+	!	!
MVC_easy	+	0.002	0.005	+	0.003	0.003
MVC_medium	+	0.001	!	+	!	0.003
MVC_hard	+	0.001	!	+	!	!
SC_easy	+	0.002	0.005	+	0.001	0.005
SC_medium	+	0.000	!	+	!	0.000
SC_hard	+	0.001	!	+	!	!
MIPlib	0.325	-	2.466e-06	0.165	!	0.831
Coral	+	-	44035.222	+	!	71894.505
Cut	0.749	-	0.217	0.730	0.112	0.544
ECOGCNN	+	-	0.432	+	!	46749.986
HEM_knapsack	0.000	0.000	0.000	0.010	!	!
HEM_mis	0.000	0.021	0.000	0.000	0.055	!
HEM_setcover	0.000	0.002	0.000	+	0.045	0.000
HEM_corlat	+	-	0.000	+	0.002	2.065
HEM_mik	+	-	6.710e-15	+	0.030	6.892e-15
item_placement	9.738e+07	-	0.173	4.216e+19	!	0.242
load_balancing	+	-	9.120e-05	+	0.008	0.027
anonymous	+	-	0.144	+	1.724	0.406
Nexp	0.275	-	0.150	+	!	3.793e+06
Transportation	0.009	-	0.006	+	0.003	0.355
vary_bounds_s1	0.040	-	0.001	0.063	3.866e-05	3.866e-05
vary_bounds_s2	+	-	0.000	+	0.000	0.000
vary_bounds_s3	+	-	0.000	+	0.000	0.000
vary_matrix_s1	0.033	-	1.516e-16	0.105	!	4.791e-16
vary_matrix_rhs_bounds_s1	+	-	3.861e-05	0.025	!	0.053
vary_matrix_rhs_bounds_obj_s1	0.122	-	5.007e-05	4.472e+19	6.168e-05	5.038e-05
vary_obj_s1	0.001	0.000	0.000	0.001	0.000	0.001
vary_obj_s2	+	-	2.931e-06	+	!	1.397
vary_obj_s3	0.150	-	0.000	+	!	0.145
vary_rhs_s1	+	-	0.001	+	0.001	0.306
vary_rhs_s2	0.000	-	1.937e-05	0.000	!	1.723e-05
vary_rhs_s3	+	0.001	9.635e-06	+	!	0.167
vary_rhs_s4	0.000	-	3.457e-05	0.000	!	7.709e-05
vary_rhs_obj_s1	+	-	1.368e-05	+	!	1.338e-05
vary_rhs_obj_s2	0.001	-	1.959e-05	0.001	!	1.778e-05
Aclib	+	-	7.091e-06	0.005	0.042	2.358
fc.data	+	-	0.000	0.027	0.000	0.000
nn_verification	+	-	4.022e-05	+	!	1.307

highly complex scenarios, as evidenced by higher standard deviations and wider error bars for methods like Neural Diving and GNN-MLP. These findings suggest that while machine learning methods have potential, further refinement is needed to improve their generalizability and stability across diverse problem types.

The results underscore the utility of MILPBench as a benchmarking tool for systematically evaluating algorithm stability across diverse problem types. By providing detailed insights into the variability and robustness of different approaches, MILPBench facilitates the development and selection of reliable optimization algorithms, bridging the gap between theoretical advancements and practical applications.

Table 19: The error bar's width of objective function value of machine learning-based baselines. The bold numbers indicate the best results achieved by the solvers. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. -represents MILP problems that cannot be solved by the IP framework, GNN&GBDT.

Problem	Learn2Branch	GNN&GBDT	Predict&Search	Hybrid-Learn2Branch	GNN-MILP	Nerual Diving
MIS_easy	+	21.417	30.653	+	22.860	62.831
MIS_medium	+	57.973	50.546	+	!	2.310
MIS_hard	+	273.179	!	+	!	!
MVC_easy	+	27.301	54.594	+	61.105	27.038
MVC_medium	+	116.601	!	+	!	136.500
MVC_hard	+	647.370	!	+	!	!
SC_easy	+	30.751	33.531	+	21.677	25.014
SC_medium	+	66.290	!	+	!	116.732
SC_hard	+	38999.409	!	+	!	!
MIPlib	57325.427	-	55897.309	57169.901	!	535180.504
Coral	+	-	2.028e+10	+	!	2027.121
Cut	43110.333	-	27969.333	44054.667	14661.667	24804.667
ECOGCNN	+	-	1.512e+06	+	!	1.512e+06
HEM_knapsack	22.600	22.600	22.600	20.600	0.000	0.000
HEM_mis	6.800	10.600	6.800	6.800	10.400	0.000
HEM_setcover	47.400	48.200	47.400	+	37.000	47.400
HEM_corlat	+	-	269.000	+	269.000	177.000
HEM_mik	+	-	29114.400	+	22194.200	29114.400
item_placement	5.231	-	2.706	8.029	!	2.156
load_balancing	+	-	41.200	+	47.500	40.700
anonymous	+	-	277335.944	+	3.096e+06	293808.183
Nexp	4.219e+08	-	4.181e+08	+	!	4.181e+08
Transportation	21496.400	-	25053.600	+	27216.800	1.305e+07
vary_bounds_s1	1333.600	-	1127.200	1513.600	1130.200	1130.200
vary_bounds_s2	+	-	0.000	+	0.000	0.000
vary_bounds_s3	+	-	0.000	+	0.000	0.000
vary_matrix_s1	1.673	-	1.545	0.988	!	1.545
vary_matrix_rhs_bounds_s1	+	-	2.115e+08	2.117e+08	!	2.115e+08
vary_matrix_rhs_bounds_obj_s1	30762.455	-	34559.345	27802.006	34558.953	34559.333
vary_obj_s1	186.400	184.600	184.600	181.000	184.600	184.600
vary_obj_s2	+	-	2120.358	+	!	2120.358
vary_obj_s3	598.545	-	598.545	+	!	598.545
vary_rhs_s1	+	-	10.216	+	10.216	103.609
vary_rhs_s2	2.217	-	3.380	3.486	!	3.380
vary_rhs_s3	+	21841.200	21893.800	+	!	12988.600
vary_rhs_s4	5.106	-	3.775	3.979	!	3.979
vary_rhs_obj_s1	+	-	63273.600	+	!	63275.325
vary_rhs_obj_s2	694353.046	-	694355.144	694346.141	!	694353.030
Aclib	+	-	178412.000	178412.000	966103.130	315081.812
fc.data	+	-	384.400	384.400	384.400	384.400
nn_verification	+	-	24.581	+	!	25.708

Table 20: The error bar's width of gap estimation of machine learning-based baselines. The bold numbers indicate the best results achieved by the solvers. + represents the problem of scale being too large to accept the time to collect training samples. ! represents the problem of errors during band training. -represents MILP problems that cannot be solved by the IP framework, GNN&GBDT.

Problem	Learn2Branch	GNN&GBDT	Predict&Search	Hybrid-Learn2Branch	GNN-MILP	Nerual Diving
MIS_easy	+	0.005	0.003	+	0.009	748.749
MIS_medium	+	0.004	0.003	+	!	367.429
MIS_hard	+	0.002	!	+	!	!
MVC_easy	+	0.003	0.007	+	0.005	0.004
MVC_medium	+	0.001	!	+	!	0.004
MVC_hard	+	0.002	!	+	!	!
SC_easy	+	0.002	0.009	+	0.001	0.009
SC_medium	+	0.000	!	+	!	0.000
SC_hard	+	0.002	!	+	!	!
MIPlib	0.328	-	3.699e-06	0.186	!	1.246
Coral	+	-	99862.498	+	!	163041.413
Cut	0.865	-	0.249	0.842	0.114	0.582
ECOGCNN	+	-	0.499	+	!	53982.234
HEM_knapsack	0.000	0.000	0.000	0.019	!	!
HEM_mis	0.000	0.030	0.000	0.000	0.070	!
HEM_setcover	0.000	0.003	0.000	+	0.056	0.000
HEM_corlat	+	-	0.000	+	0.003	3.665
HEM_mik	+	-	1.200e-14	+	0.054	1.233e-14
item_placement	2.660e+08	-	0.339	8.000e+19	!	0.536
load_balancing	+	-	0.000	+	0.020	0.060
anonymous	+	-	0.187	+	2.586	0.591
Nexp	0.490	-	0.268	+	!	6.785e+06
Transportation	0.013	-	0.008	+	0.004	0.634
vary_bounds_s1	0.054	-	0.001	0.072	6.916e-05	6.916e-05
vary_bounds_s2	+	-	0.000	+	0.000	0.000
vary_bounds_s3	+	-	0.000	+	0.000	0.000
vary_matrix_s1	0.044	-	2.096e-16	0.174	!	5.573e-16
vary_matrix_rhs_bounds_s1	+	-	6.426e-05	0.045	!	0.082
vary_matrix_rhs_bounds_obj_s1	0.203	-	5.711e-05	8.000e+19	8.306e-05	5.687e-05
vary_obj_s1	0.001	0.000	0.000	0.001	0.000	0.001
vary_obj_s2	+	-	5.185e-06	+	!	2.079
vary_obj_s3	0.269	-	0.000	+	!	0.190
vary_rhs_s1	+	-	0.001	+	0.001	0.539
vary_rhs_s2	0.000	-	3.034e-05	0.000	!	2.487e-05
vary_rhs_s3	+	0.002	1.338e-05	+	!	0.296
vary_rhs_s4	0.000	-	3.682e-05	0.000	!	9.921e-05
vary_rhs_obj_s1	+	-	1.999e-05	+	!	2.344e-05
vary_rhs_obj_s2	0.001	-	3.022e-05	0.001	!	2.777e-05
Aclib	+	-	2.018e-05	0.008	0.071	6.661
fc.data	+	-	0.000	0.049	0.000	0.000
nn_verification	+	-	7.086e-05	+	!	3.310