Distributional MIPLIB: a Multi-Domain Library for Advancing ML-Guided MILP Methods

Weimin Huang, Taoan Huang
University of Southern California

Aaron Ferber
Cornell University

Bistra Dilkina
University of Southern California

Abstract

Mixed Integer Linear Programming (MILP) is a fundamental tool for modeling combinatorial optimization problems. Recently, a growing body of research has used machine learning to accelerate MILP solving. Despite the increasing popularity of this approach, there is a lack of a common repository that provides distributions of similar MILP instances across different domains, at different hardness levels, with standardized test sets. In this paper, we introduce *Distributional MIPLIB*, a multi-domain library of problem distributions for advancing ML-guided MILP methods. We curate MILP distributions from existing work in this area as well as real-world problems that have not been used, and classify them into different hardness levels. It will facilitate research in this area by enabling comprehensive evaluation on diverse and realistic domains. We empirically illustrate the benefits of using *Distributional MIPLIB* as a research vehicle in two ways. We evaluate the performance of ML-guided variable branching on previously unused distributions to identify potential areas for improvement. Moreover, we propose to learn branching policies from a mix of distributions, demonstrating that mixed distributions achieve better performance compared to homogeneous distributions when there is limited data and generalize well to larger instances.

1 Introduction

Mixed Integer Linear Programming (MILP) is an essential technique for modeling and solving Combinatorial Optimization (CO) problems, covering a wide range of applications such as production planning and scheduling [83]. Many CO problems are NP-complete or NP-hard [61, 31, 51, 43, 86] and are thus inherently challenging to solve. Exact algorithms [79] and heuristics [72, 8] have been studied for MILPs. However, solving MILPs remains challenging as problems scale in size and complexity, coupled with the increasing demand for real-time solutions.

Many algorithmic decisions in exact and heuristic algorithms for MILPs traditionally rely on intuition from problem structures and/or are manually made based on evaluation on specific instances [68, 59]. However, manual tuning requires domain-specific knowledge and may fail to realize the full performance potential of algorithms. In recent years, Machine Learning (ML) has been proposed to address this shortcoming. There has been an increasing interest in enhancing MILP-solving frameworks with adaptable learning components that exploit the correlation between algorithmic patterns and the performance of the algorithm [22, 25]. For example, [46, 26, 33, 85, 37, 52] improves Branch-and-Bound (B&B), a tree search algorithm used in MILP solvers [35, 14], with ML.

Despite the increasing popularity of ML-guided MILP solving, there is a lack of a common repository containing distributions of MILPs along with standardized test sets for ML approaches. Some researchers [63, 64] use MIPLIB, a library containing various MILP instances that differ in difficulties, structures, and sizes [29]. While MIPLIB has been traditionally used for benchmarking MILP solvers, its instances are heterogeneous, making it less suited for ML-based methods. As ML typically benefits from a large amount of data from a certain distribution, it remains a challenge for ML methods to

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

deliver state-of-the-art performance on MIPLIB instances [63, 64]. To leverage data in distributional settings, much of the existing work independently generates MILP distributions. This leads to two issues. First, the lack of standardized test sets makes it hard to benchmark and compare between different methods. Second, a small set of synthetic domains has been repeatedly used and there is a lack of evaluation on real-world domains, making the evaluation not comprehensive. In Table 1, we summarize problem domains used in representative papers for different learning tasks. Classical problems such as Set Covering (SC), Combinatorial Auction (CA), Maximum Independent Set (MIS), and Capacitated Facility Location Problem (CFLP) are commonly used. Although these problems are NP-hard, the instances are synthetic and less challenging compared to many real-world problems. A few papers [64, 75, 76] have used instances from real-world problems such as the production packing problem and electric grid optimization [64], but few of the real-world datasets are publicly accessible, making it hard to reproduce the results.

Table 1: MILP distributions used in previous work. † indicates those that are not publicly available. For each work, we mark whether the ML component is trained on distributions of Single Domain (SD), Mixed Domain (MD), or MIPLIB. We also mark whether it is tested in problem domains that are the same as training (ID), in the same domain but on larger distributions (ID(L)), or out of domains (OD). The names of domains corresponding to the abbreviations are in Sec 3.1 and Appendix A.

Track	Paper	Training	Testing	Problem Domains
Learning for B&B				
Branching	Khalil et al. [46]	SD	ID	MIPLIB
	Gasse et al. [26]	SD	ID(L)	SC, CA, CFLP, MIS
	Nair et al. [64]	SD + MIPLIB	ID + MIPLIB	CORLAT, NNV, Google Production Packing†, Electric Grid Optimization†, MIPLIB
	Gupta et al. [33]	SD	ID(L)	SC, CA, CFLP, MIS
	Zarpellon et al. [85]	MIPLIB	MIPLIB	MIPLIB
	Gupta et al. [34]	SD	ID(L)	SC, CA, CFLP, MIS
	Scavuzzo et al. [74]	SD	ID(L)	SC, CA, CFLP, MIS, MK
	Lin et al. [56]	SD	ID(L)	SC, CA, CFLP, MIS
Backdoor prediction	Ferber et al. [24]	SD	ID	NNV, CFLP, GISP
	Cai et al. [15]	SD	ID(L)	SC, CA, CFLP, MIS, GISP, NNV
Node selection	He et al. [37]	SD	ID + OD	CA, CORLAT, MK
	Labassi et al. [52]	SD	ID(L)	GISP, Fixed Charge Network Flow, MAXSAT
Cut selection	Tang et al. [77]	SD	ID(L) + OD	Packing, Production Planning, Binary Packing, MC
	Huang et al. [44]	SD	ID	SC, MK, Production Planning†
	Li et al. [55]	SD + MIPLIB	ID + MIPLIB	CA, CFLP, MIS, Packing, Binary Packing, MC
Run heuristics	Khalil et al. [47]	SD + MIPLIB	ID + MIPLIB	GISP, MIPLIB
	Chmiela et al. [17]	SD	ID(L)	GISP, Fixed Charge Network Flow
Learning for meta heuristics				
LNS	Song et al. [75]	SD	ID	CA, MVC, MC, Risk-Aware Path Planning†
	Wu et al. [84]	SD + MIPLIB	ID(L) + MIPLIB	SC, CA, MIS, MC, MIPLIB
	Sonnerat et al. [76]	SD + MIPLIB	ID + MIPLIB	NNV, Google Production Packing†, Electric Grid Optimization†, MIPLIB, Google Production Planning†
	Liu et al. [57]	SD + MIPLIB + MD	MIPLIB + OD	SC, CA, MIS, GISP, MIPLIB
	Huang et al. [41]	SD	ID(L)	SC, CA, MIS, MVC
Solution prediction	Ding et al. [22]	SD	ID(L)	SC, CFLP, MIS, MK, Fixed Charge Network Flow, TSP, VRP, Generalized Assignment
	Nair et al. [64]	SD + MIPLIB	ID + MIPLIB	CORLAT, NNV, Google Production Packing†, Electric Grid Optimization†, MIPLIB
	Khalil et al. [48]	SD	ID	GISP, Fixed Charge Network Flow
	Han et al. [36]	SD	ID(L)	CA, MIS, IP, LB
	Huang et al. [42]	SD	ID(L)	CA, MIS, MVC, IP

This paper introduces *Distributional MIPLIB* (d-MIPLIB), a comprehensive, multi-purpose MILP library encompassing various MILP problem distributions to support the development of ML-guided MILP-solving methods. In this context, a distribution refers to MILPs of the same problem formulation constructed from data parameters sampled from a given distribution. We curate distributions from ten synthetic and real-world problems used in the existing literature on ML for MILPs and three real-world problems for which no ML methods have been attempted. For each problem, distributions are classified into multiple hardness levels. 100 test instances are pre-generated for each distribution, and 900 are pre-generated for training and validation [1]. Additionally, a generator is provided for most problems to generate additional instances for training.

[1]The number of test instances in 3 distributions is less than 100 due to limited available data.

Distributional MIPLIB will significantly accelerate research in MILP solving and data-driven algorithm design by providing distributional data for ML-based methods and enabling benchmarking. The set of distributions covers various hardness levels and a diverse set of application domains, making it suitable for different types of MILP algorithms (e.g., smaller problems for exact solving and larger problems for heuristic methods). Moreover, the standardized benchmark sets that d-MIPLIB provides not only enable better comparison analysis of different methods, but also enable evaluation across broader domains at different problem scales and on realistic problems, allowing researchers to identify gaps and open up new avenues of novel research.

To demonstrate the potential of *Distributional MIPLIB* in facilitating research, we evaluate the performance of ML-guided variable branching [26] on previously unused distributions and uncover potential areas for improvement. Moreover, we propose to learn branching policies in B&B from a mix of distributions, demonstrating that mixed distributions achieve better performance compared to homogeneous distributions when there is limited data and generalize better to larger instances. Furthermore, we propose several additional directions for utilizing the dataset, highlighting its potential for opening up new research avenues. To encourage further research and facilitate the curation of future distributions, we provide a website for *Distributional MIPLIB*. The URL to the website will be included in the final version of the paper.

2 Background and Related Work

Formally, a MILP with n decision variables and m constraints is defined by a coefficient matrix $A \in \mathbb{R}^{m \times n}$, a vector $b \in \mathbb{R}^m$, a cost vector $c \in \mathbb{R}^n$, and a partition (B, I, C) of variables. B, I, C are the sets of indices of binary, general integer, and continuous variables, respectively. The goal is to find x such that $c^T x$ is maximized, subject to linear constraints $Ax \leq b$ and integrality constraints on binary decision variables $x_j \in \{0, 1\}, \forall j \in B$ and integer decision variables $x_j \in \mathbb{Z}, \forall j \in I$.

MILP solvers such as Gurobi [35] and SCIP [14] use Branch-and-Bound (B&B), an exact tree search algorithm, as the core component. B&B starts with the root node representing the original input MILP. It then repeatedly chooses a leaf node and creates two smaller subproblems by splitting the domain of a variable. This step is referred to as *branching*. Besides B&B, meta-heuristics, such as Large Neighborhood Search (LNS) and Predict-and-Search (PaS), are also popular MILP search algorithms that can find high-quality solutions to MILPs much faster than B&B for hard problems, but do not provide optimality guarantees.

2.1 Machine Learning for MILP Solving

ML has been proposed to accelerate MILP solving in different ways. A large body of research improves B&B by learning to select which variables to branch on [46, 26, 33, 85] or which nodes to expand in the search tree [37, 52]. There are also works on learning to schedule or execute primal heuristics [47, 17] and to select cutting planes [77, 66, 44] in B&B. ML has also been applied to improve meta-heuristics. [75, 76, 84, 41] apply learning techniques, such as imitation, reinforcement, and contrastive learning, to learn to select which subset of variables to reoptimize in LNS. [64, 36, 42] focus on PaS, where they learn to predict the optimal assignment for part of the variables to get a reduced-size MILP that is easier to solve. A comprehensive literature review is provided Appendix B.

2.2 Existing Libraries and Software Packages

MIPLIB [29] is a library that provides access to heterogeneous real-world MILP instances, containing 1065 instances from various domains that are diverse in size, structure, and hardness. It has become a standard test set used to compare the performance of MILP solvers, and several ML methods for MILP solving have been tested on MIPLIB instances [76, 84, 55]. Despite some early success, it remains a challenge for ML methods to deliver state-of-the-art performance on MIPLIB instances, due to the heterogeneous nature [63, 64].

There are also a few open-source software packages built to facilitate research in ML-guided CO. MIPLearn [73] is a software for ML-guided MILP solving that provides access to a complete ML pipeline, including data collection, training, and testing. MIPLearn provides generators for one real-world problem, which is a simplified formulation for Unit Commitment (UC). However, domain knowledge is required to generate realistic UC instances that are more complex and more challenging

to solve. Ecole [71] is a library designed to facilitate research on using ML to improve CO solvers. It exposes the sequential decision-making processes in MILP-solving as control problems over Markov Decision Processes. Currently, Ecole provides instance generators for four classical problems. OR-Gym [45] is a framework for developing RL algorithms to produce high-quality solutions for CO, without using MILP solvers. Ecole and OR-Gym are designed for augmenting solvers and finding high-quality solutions without solvers, respectively. MIPLearn supports both tasks.

Comparison with Existing Datasets and Contributions. *Distributional MIPLIB* provides a counterpart for the MIPLIB that provides distributions of similar MILP instances of the same problem formulation, intended for the development and evaluation of ML-guided MILP methods. Similar to MIPLIB, it covers a broad range of application domains, allowing for a more comprehensive evaluation on problems with different structures, especially in real-world domains. All the instances are pre-compiled, and no domain knowledge is required to access complex real-world instances included in d-MIPLIB. It covers multiple hardness levels, making it suitable for a wide range of MILP methods (e.g., exact solving for easier instances and meta-heuristics for harder instances), compared to libraries designed for specific avenues.

3 Distributional MIPLIB

We pre-generate MILP distributions from both synthetic and real-world problem domains, classifying them into different hardness levels. Table 2 shows the sources where the distributions were initially used in existing work on ML for MILPs, along with instance statistics. While we pre-compile a fixed number of instances for consistency in comparing methods, an instance generator is available for generating additional training instances for all synthetic problems and one real-world problem (Optimal Transmission Switching).

3.1 Data Sources

We curate synthetic instances from domains that have been used in multiple ML tasks in existing work (Table 1). Among variants of a problem in synthetic domains, we choose the most representative one among the variants. As for real-world domains, as many problems used in existing work are proprietary and not publicly available, we cover the associated domain by including an available problem from the same area (e.g., Optimal Transmission Switching as a surrogate for Electric Grid Optimization used in [76]).

Synthetic Problems. We curate synthetic instances from domains commonly used in the literature on ML for MILPs. As shown in Table 1, the most frequently used NP-hard problem domains are Combinatorial Auctions (CA) [53], Set Covering (SC) [5], Maximum Independent Set (MIS) [7], Capacitated Facility Location Problem (CFLP) [20], and Minimum Vertex Cover (MVC) [23]. Additionally, we compile distributions of the Generalized Independent Set Problem (GISP), a graph optimization problem proposed for forestry management [39, 18]. We used the instance generators provided in the existing work to compile MILP distributions as described in their work and generate additional distributions covering different hardness levels for frequently used domains such as Minimum Vertex Cover (MVC). Finally, we include Item Placement (IP), which involves spreading items across containers to utilize them evenly [62], and Load Balancing (LB) [82], which deals with apportioning workloads across workers, used in the NeurIPS 2021 Machine Learning for Combinatorial Optimization Competition (ML4CO) [25].

Real-world Problems. In addition to synthetic instances, we include MILP instances from five real-world domains. The Maritime Inventory Routing Problem (MIRP) [65] determines routes from production ports to consumption ports to minimize transportation costs and manages the inventory at these ports, covering both ship routing and inventory management. MIRP was used as a hidden test set in ML4CO [25]. Neural Network Verification (NNV) is an optimization problem in ML that verifies the robustness of a neural network on a given input example [16, 78]. The NNV instances we include were derived from verifying a convolutional network on MNIST examples, which was used in [64] for learning for branching and solution prediction.

Furthermore, we compile distributions from problems where no ML method has been applied, covering applications in energy, e-commerce, and sustainability. In energy planning, the Optimal Transmission Switching (OTS) problem under high wildfire ignition risk [70] is a type of Network

Topology Optimization problem. Transmission grids are represented as a series of buses (vertices) connected by power lines (edges). During high wildfire ignition risk, transmission lines can start wildfires; methods to mitigate this risk include de-energizing and undergrounding transmission lines. De-energizing lines prevent fires but interrupt power delivery to customers, whereas undergrounding lines can deliver power without the risk of igniting a fire but at a higher cost. OTS examines the optimal way to de-energize and underground transmission lines to reduce wildfire risk while minimizing power outages within a resource budget. In e-commerce, the Middle-Mile Consolidation Network (MMCN) [31] problem is a network design problem that creates load consolidation plans to transport shipments from stocking locations, including vendors and fulfillment centers, to last-mile delivery locations. It determines a minimum-cost allocation of transportation capacity on network arcs that satisfies shipment lead-time constraints. For MMCN, we include distributions containing binary and integer variables (denoted as BI) and distributions containing binary and continuous variables (denoted as BC) [2]. The Seismic-Resilient Pipe Network (SRPN) Planning [40] is another network design problem in infrastructure resilience space that chooses which water pipes are to be upgraded in earthquake hazard zones to ensure water delivery to critical customers and households during disasters, while minimizing the cost. The SRPN instances in this library are generated based on earthquake hazard zones in Los Angeles.

As shown in Table 2, most synthetic MILP instances contain only binary decision variables, except for CFLP, LB, and IP, which include continuous variables. The real-world problems, on the other hand, encompass diverse distributions with integer and continuous decision variables, enabling comprehensive benchmarking on more realistic and complex problems.

3.2 Evaluation

Instances Data Generation. For each synthetic distribution, we generate a total of 1000 instances, with 900 intended for training and validation in ML-guided methods and 100 for testing and evaluation. For the real-world problems OTS and MMCN, we follow the same practice as the synthetic problems, providing 100 test instances for each distribution. For NNV, since precompiled train, validation, and test splits are publicly available, we respect the established splits, including the same 588 instances in the test set. However, for MIRP and SRPN, the number of test instances is less than 100 as the total number of instances available is limited. MIRP contains 20 test instances. SRPN contains 22 and 20 test instances in the Easy and Hard group, respectively [3].

Performance Metrics and Problem Instance Statistics. We design a set of evaluation metrics that characterize performance well from easy to hard settings. We report the number of instances in the test set that are solved to optimality in 1 hour (# Opt). For instances solved to optimality, we report the average solving time in seconds (Opt Time). For instances not solved to optimality, we report the average primal-dual gap after 1 hour (NonOpt Gap). The primal-dual gap represents the gap between the lower and upper objective bounds. Specifically, let z_P be the primal objective bound (i.e., the value of the best feasible solution found so far, serving as the upper bound for minimization problems), and z_D be the dual objective bound (i.e., linear relaxation of the MILP, serving as the lower bound for minimization problems). The primal-dual gap is defined as $gap = |z_P - z_D|/|z_P|$ [35]. Additionally, we report the primal-dual integral (Integral), which is defined as the integral of the primal-dual gap over time [9], with lower values indicating faster (better) convergence. For instance statistics, we report the average number of binary (# Var B), integer (# Var I), and continuous (# Var C) variables, and the average number of constraints (# Constr).

Hardness Levels. We classify distributions into 5 hardness levels based on the runtime statistics. For distributions with at least some instances solved to optimality within 1 hour, we classify them into three levels based on the average solving time. Distributions with average solving times under 100 seconds are categorized as *Easy*, 100-1000 seconds as *Medium*, and those exceeding 1000 seconds as *Hard*. For distributions with no instances solved to optimality within 1 hour, we further classify them into *Very hard* and *Extremely hard* based on the primal-dual gap. *Very hard* and *Extremely hard* (Ext hard) distributions are groups where the primal-dual gap is less than 1 and greater than 1, respectively.

[2] BI and BC distributions correspond to 2 variants of MMCN. In the BI variant, all arcs in the network have the same transit mode. In the BC variant, multiple transit modes are allowed.

[3] As MIRP has been used in ML4CO, we adhere to the train, validation, and test split established by ML4CO. For SRPN, we randomly selected 10% of the total instances as the test set for each distribution.

Table 2: Synthetic and Real-world problems in *Distributional* MIPLIB. † indicates domains for which generators are available. ‡ indicates distributions where # test instances is not 100. For the performance metrics, we use Gurobi (v10.0.3) [35] with 1 hour time limit, on a cluster with Intel Xeon Silver 4116 CPUs @ 2.10GHz, with a RAM allocation of 5G (For SRPN-Hard, MMCN-Very Hard, and OTS-Hard instances, we increased the RAM to 15G, due to memory errors at 5GB.)

Domain	Hardness Level	Dist. Source: ML4MILPs	Instance Statistics				Performance Metrics			
			# Var B	# Var I	# Var C	# Constr	# Opt	Opt Time(s)	NonOpt Gap	Integral
Synthetic										
CA†	Easy	Gasse et al. [26]	1000	0	0	385.04	100	47.14	N/A	2.30
	Medium	Gasse et al. [26]	1500	0	0	578.07	100	358.14	N/A	7.29
	Very hard	Huang et al. [41]	4000	0	0	2676.32	0	N/A	0.10	400.28
SC†	Easy	Gasse et al. [26]	1000	0	0	500	100	18.05	N/A	0.99
	Medium	Gasse et al. [26]	1000	0	0	1000	100	214.11	N/A	15.78
	Hard	Gasse et al. [26]	1000	0	0	2000	56	1603.66	0.04	180.25
	Very hard	Huang et al. [41]	4000	0	0	5000	0	N/A	0.20	847.11
MIS†	Easy	Gasse et al. [26]	1000	0	0	3946.25	100	50.52	N/A	0.86
	Medium	Gasse et al. [26]	1500	0	0	5941.14	88	470.44	0.01	11.28
	Very hard	Huang et al. [41]	6000	0	0	23994.82	0	N/A	0.30	1132.69
MVC†	Easy	New	1200	0	0	5975	100	27.26	N/A	0.27
	Medium	New	2000	0	0	9975	97	244.11	0.01	2.28
	Hard	New	500	0	0	30100	55	1821.04	0.02	102.74
	Very hard	Huang et al. [41]	1000	0	0	65100	0	N/A	0.12	454.02
GISP†	Easy	New	605.81	0	0	1967.05	100	43.09	N/A	15.59
	Medium	Ferber et al. [24]	988.81	0	0	3353.03	100	671.89	N/A	204.83
	Hard	Ferber et al. [24]	1317.03	0	0	4567.83	85	2623.16	0.08	866.16
	Very hard	Cai et al. [15]	6017	0	0	7821.87	0	N/A	0.44	2104.04
	Ext hard	Khalil et al. [47]	12675.83	0	0	16515.44	0	N/A	2.01	8139.33
CFLP†	Easy	Gasse et al. [26]	100	0	10000	10201	100	44.44	N/A	0.57
	Medium	Gasse et al. [26]	200	0	20000	20301	100	103.51	N/A	0.88
LB †	Hard	Gasse et al. [25]	1000	0	60000	64307.17	9	2665.11	0.00	33.48
IP †	Very hard	Gasse et al. [25]	1050	0	33	195	0	N/A	0.44	1770.42
Real-world										
MIRP	Medium	Gasse et al. [25]	0	15080.57	19576.15	44429.70	10‡	697.24	0.23	728.75
NNV	Easy	Nair et al. [64]	171.49	0	6972.60	6533.70	588‡	37.98	N/A	21.81
OTS†	Easy	New	4181	0	17137	48582	100	45.86	N/A	3.72
	Medium	New	7525	0	33202	92992	100	419.55	N/A	25.80
	Hard	New	6546	0	46423	111804	52	2564.00	0.20	1926.19
MMCN	Medium^{BI}	New	1156.94	263.23	0	437.81	100	114.93	N/A	3.01
	Medium^{BC}	New	4271.59	0	324.04	3171.23	100	468.17	N/A	37.30
	Hard^{BI}	New	2074.76	346.39	0	642.57	34	1998.57	0.01	79.79
	Very hard^{BI}	New	21596.72	1127.29	0.00	3944.01	0	N/A	0.10	369.15
	Very hard^{BC}	New	68345.21	0	2425.87	96272.60	0	N/A	0.61	2761.52
SRPN	Easy	New	3016.42	0	3016.42	5917.27	21‡	77.91	0.02	10.00
	Hard	New	11485.33	0	11485.33	22430.84	9‡	1321.43	0.03	134.12

4 Computation Experiments

We illustrate the value of *Distributional MIPLIB* through computational experiments on its MILP distributions. First, we identify distributions that are unexplored in previous work in ML-guided MILP solving and identify potential areas for improvement (subsection 4.1). Furthermore, we propose a novel setting where we learn ML policies from a diverse mix of domains, contrasting with existing work that either trains models on single distributions or completely heterogeneous distributions such as MIPLIB (Table 1) (subsection 4.2). We show that the proposed mixed-domain strategy is particularly effective in data-scarce settings.

Most ML-guided MILP approaches discussed in subsection 2.1 require a computationally expensive data collection procedure before training ML models, as they replace computationally intensive algorithmic components with ML oracles. The output of the expensive algorithmic component (e.g., high-quality neighborhood candidate variables obtained via local branching in LNS) is used as the ground truth in supervised learning. Given that *Distributional MIPLIB* spans many problem

domains and hardness levels, complete benchmarking is beyond the scope of this paper. In this paper, we focus our experiments on Learning to Branch (Learn2Branch) [26], which imitates Strong Branching — a branching rule that effectively reduces the search tree size in B&B but is time-consuming. Learn2Branch encodes a MILP with a variable-constraint bipartite graph, employs a Graph Convolution Network (GCN) to learn variable representations, and trains a policy using imitation learning.

Throughout the experiments, we use SCIP 6.0.1 [28] as the solver [4]. Following existing work [56, 26], we compare the ML methods against Reliability Pseudocost Branching (RPB), a state-of-the-art human-designed branching policy in B&B. We report the mean and standard deviation over 5 seeds for all metrics. We briefly introduce the setup; details on data collection and hyperparameters are deferred to Appendix C.

4.1 Learning on Previously Unused Distributions

Table 3: Learn2Branch evaluated on previously unused domains. Note that the solving time differs from Table 2 because results in 2 were evaluated with Gurobi and under different RAM allocations.

Dist.	Method	Integral	# Opt	Opt Time(s)	NonOpt Gap	# Nodes	Infer Pct(%)	Node Integral
GISP (Medium)	SCIP	**118.0 ± 1.5**	**100**	**376.6 ± 4.7**	N/A	158866.1 ± 1693.4	N/A	38596.6 ± 508.0
	ML	139.0 ± 5.8	98.0 ± 1.5	472.8 ± 16.2	0.121 ± 0.007	**89354.6 ± 2622.5**	21.2 ± 0.4	**22636.5 ± 683.3**
OTS (Easy)	SCIP	**20.0 ± 4.7**	**94.4 ± 2.2**	179.4 ± 5.5	0.003 ± 0.002	**1798.7 ± 242.9**	N/A	**5.8 ± 2.2**
	ML	27.9 ± 10.4	80.4 ± 7.7	**129.7 ± 15.2**	0.003 ± 0.002	4073.2 ± 1416.0	4.9 ± 1.1	20.6 ± 8.2
SRPN (Easy)	SCIP	**47.0 ± 1.7**	**13.8 ± 0.4**	54.8 ± 15.1	**0.152 ± 0.006**	15420.2 ± 1937.0	N/A	1594.2 ± 160.9
	ML	52.3 ± 4.3	12.8 ± 0.7	**41.8 ± 8.4**	0.16 ± 0.011	**13003.5 ± 1739.8**	14.9 ± 1.9	**1371.6 ± 93.6**

We evaluate the performance of Learn2Branch on three unused distributions. To our knowledge, GISP has not been used in learning variable branching (Table 1), and OST and SRPN have never been used in any ML-guided methods. We focus on Easy and Medium distributions as learning for branching is typically used on smaller instances in the literature.

Setup. We use a train, validation, and test split of 80%, 10%, 10%, respectively. This results in 800 MILP instances used for collecting training data for GISP and OTS and 175 for SRPN-Easy, as SRPN instances are limited. We collect 10 Strong Branching expert samples from each instance. We report the performance metrics described in 3.2 with a time limit of 800s.

Results and discussions. As shown in Table 3, the trained ML policy did not outperform SCIP in any of the 3 distributions. We investigate the reason for failure by measuring the number of explored nodes in B&B (# Nodes), the integral of the primal-dual gap with respect to the nodes (Node Integral), and the % of time spent in ML inferences (Infer Pct (%)), which includes feature extraction, forward pass, and ranking. The reason why Learn2Branch did not work well on GISP and SRPN could be the overhead of the ML inference time, as they outperform SCIP on the number of Nodes and Node Integral. For OTS, the reasons why Learn2Branch fails to beat SCIP are less obvious and pose an open research question.

4.2 Learning with Mixed Distributions

Collecting expert samples for imitation learning in Learn2Branch is computationally intensive [56]. While collecting a large number of expert samples from a large number of training instances can lead to stronger performance, it could be prohibitively costly. One simple strategy to make the best use of limited data is to pool data and train policies on mixed domains, as opposed to existing work that trains models on a single distribution, distributions from variants of a single problem family [11, 10], or completely heterogeneous distributions such as MIPLIB (Table 1). Empirically, we show that pooling data from a diverse mix of domains achieves better performance when limited training data is used.

Setup. We collect samples from training instances from 5 different domains: MIS-Easy, GISP-easy, CFLP-easy, CA-Medium, and SC-Medium. We use the collected data in two different ways. First,

[4] We use SCIP in the Experiments as opposed to Gurobi, since Gurobi does not provide needed API for ML-guided branching

Table 4: Performance comparison under two training strategies, evaluated on five domains. Under the first strategy, a separate model is trained for each domain on expert samples collected from instances drawn from homogeneous distributions from the corresponding domain: ML-MIS, ML-GISP, ML-CFLP, ML-CA, and ML-SC. ML-mix5 is trained under the second strategy, where pooled data collected from instances in the 5 domains are used to train one single model. We present results when using different numbers n of training instances per domain: $n = 80$ (left) and $n = 320$ (right).

Dist.	Policy	**Collected samples from 80 instances per domain**				**Collected samples from 320 instances per domain**			
		Integral	# Opt	Opt Time(s)	NonOpt Gap	Integral	# Opt	Opt Time(s)	NonOpt Gap
MIS (Easy)	SCIP	4.412 ± 0.118	**99.0 ± 0.0**	145.4 ± 3.9	0.022 ± 0.004	4.412 ± 0.118	99.0 ± 0.0	145.4 ± 3.9	0.022 ± 0.004
	ML-MIS	5.408 ± 5.309	82.6 ± 31.3	140.5 ± 66.5	**0.016 ± 0.003**	**2.434 ± 0.074**	99.0 ± 0.6	**89.2 ± 5.1**	**0.015 ± 0.001**
	ML-mix5	**2.781 ± 0.197**	98.0 ± 1.1	**107.3 ± 13.0**	**0.016 ± 0.004**	2.545 ± 0.107	99.0 ± 0.6	97.4 ± 5.4	0.016 ± 0.003
GISP (Easy)	SCIP	12.509 ± 0.242	100	40.0 ± 0.7	N/A	12.509 ± 0.242	100	40.0 ± 0.7	N/A
	ML-GISP	11.299 ± 0.885	100	41.0 ± 3.4	N/A	10.700 ± 0.442	100	38.5 ± 1.6	N/A
	ML-mix5	**10.823 ± 0.383**	100	**39.1 ± 2.0**	N/A	**10.420 ± 0.279**	100	**37.3 ± 0.8**	N/A
CFLP (Easy)	SCIP	0.644 ± 0.021	100	48.5 ± 0.5	N/A	0.644 ± 0.021	100	48.5 ± 0.5	N/A
	ML-CFLP	0.642 ± 0.036	100	47.8 ± 3.4	N/A	**0.606 ± 0.028**	100	42.4 ± 1.9	N/A
	ML-mix5	**0.638 ± 0.020**	100	**46.7 ± 2.8**	N/A	0.610 ± 0.021	100	**42.1 ± 1.0**	N/A
CA (Med)	SCIP	2.347 ± 0.034	97.2 ± 0.4	157.4 ± 4.8	0.009 ± 0.001	2.347 ± 0.034	97.2 ± 0.4	157.4 ± 4.8	**0.009 ± 0.001**
	ML-CA	1.927 ± 0.063	97.0 ± 0.0	144.9 ± 6.2	**0.007 ± 0.001**	**1.775 ± 0.056**	98.2 ± 0.7	**136.8 ± 2.1**	**0.009 ± 0.003**
	ML-mix5	**1.815 ± 0.015**	**98.2 ± 0.7**	**141.0 ± 3.2**	0.009 ± 0.002	1.795 ± 0.199	**98.6 ± 0.8**	142.1 ± 12.5	0.011 ± 0.002
SC (Med)	SCIP	6.465 ± 0.023	100	90.3 ± 0.6	N/A	6.465 ± 0.023	100	90.3 ± 0.6	N/A
	ML-SC	5.602 ± 0.156	100	84.6 ± 2.2	N/A	4.965 ± 0.095	100	72.5 ± 1.7	N/A
	ML-mix5	**5.362 ± 0.131**	100	**79.8 ± 2.2**	N/A	**4.796 ± 0.104**	100	**68.4 ± 1.7**	N/A

we train a separate model for each domain. Second, we pool expert samples collected for all domains and train a single model from the mixed distribution (denoted as ML-mix5). The number of training samples fed into ML-mix5 is five times the first strategy, but the data collection costs for the two strategies aggregated across the 5 domains are the same. We first start with $n = 80$ training instances per domain, which is 10% what we used in 4.1. We then quadruple the number of training instances to $n = 320$. Following 4.1, we collect 10 expert samples per instance. We compare the performance of the two training strategies (single domain vs. mixed domains) on each domain separately.

Results and Discussions. As shown in Table 4, when the total number of instances used for data collection is small (80), ML-mix5 outperforms the models trained on homogeneous distributions and SCIP across multiple evaluation metrics for all domains. However, as the number of training instances increases (320), the models trained on a homogeneous distribution outperform ML-mix5 in some domains. This indicates that learning with mixed distributions can improve data collection efficiency in the case when we have a limited budget for data collection (e.g., under time or computational resource constraints), but does not surpass training on homogeneous distributions when training samples can be collected from a larger number of instances. Additionally, Table 4 suggests that when the number of training data points fed into the model is the same, using a training set where the data is drawn from mixed distributions is unlikely to surpass the performance of using a training set where the data is drawn from homogeneous distributions. The performance of ML-mix5 under 80 instances per domain, which was trained with samples collected from 400 training instances in total, did not outperform the separately trained models under 320 instances per domain. This underscores the benefits of having domain-specific distributional datasets as provided in our library.

Table 5: Performance comparison under two training strategies when transferred to different hardness. ML-MIS (trained on *Easy*), ML-SC (trained on *Medium*), and ML-mix5 are the ones presented in Table 4 (under $n = 320$). The time cutoff is 800s, except for *Very hard* distributions where it is 3600s.

Policy	Integral	# Opt	Opt Time(s)	NonOpt Gap	Integral	# Opt	NonOpt Gap	Infer Pct(%)	Node Integral
	MIS (Medium)				**MIS (Very hard)**				
SCIP	23.4 ± 0.1	11.4 ± 1.0	483.2 ± 10.5	0.024 ± 0.0	1479.3 ± 2.3	0	0.393 ± 0.002	N/A	223.6 ± 41.7
ML-MIS	21.9 ± 2.4	10.2 ± 10.4	377.2 ± 20.0	0.023 ± 0.003	1461.5 ± 4.8	0	**0.390 ± 0.002**	0.1 ± 0.0	179.0 ± 56.4
ML-mix5	**16.5 ± 0.2**	**24.2 ± 3.9**	**335.3 ± 19.6**	**0.017 ± 0.0**	**1459.0 ± 2.4**	0	**0.390 ± 0.001**	0.1 ± 0.0	**139.4 ± 42.9**
	SC (Hard)				**SC (Very hard)**				
SCIP	53.3 ± 0.2	35.0 ± 3.5	378.5 ± 10.8	0.066 ± 0.000	**767.0 ± 1.0**	0	**0.239 ± 0.001**	N/A	3853.6 ± 82.6
ML-SC	49.6 ± 0.4	37.8 ± 1.0	367.1 ± 7.3	**0.062 ± 0.001**	870.3 ± 13.6	0	0.297 ± 0.009	9.2 ± 1.6	**2622.8 ± 579.1**
ML-mix5	**48.2 ± 0.3**	**40.2 ± 0.7**	**358.2 ± 3.3**	**0.062 ± 0.001**	830.3 ± 19.5	0	0.275 ± 0.010	13.3 ± 4.7	4051.3 ± 1734.7

Transferring to Different Distributions. We further evaluate the performance of trained models when applied to distributions of different hardness levels from the same domain, for MIS and SC. Table 5 shows that on MIS, ML-mix5 exhibits better generalization to harder instances compared to the model trained on homogeneous distributions, even though ML-mix5 did not outperform ML-MIS at the trained hardness level (Table 4). On SC, again ML-mix5 exhibits better performance than ML-SC on harder distributions of SC, however on the *Very hard* distribution neither is able to outperform SCIP, possibly due to the larger overhead of the GCN inference time on larger instances.

5 Potential Research Paths

Below we outline suggestions for potential research paths using *Distributional MIPLIB* to facilitate a significant step-change in the ability to solve hard real-world MILP problems.

Faster Inference. Due to computational constraints, prior work has focused on training and testing on relatively small and/or easy MILP distributions. In addition to Learn2Branch, much of the existing work on ML for MILP focuses on replacing an expensive procedure with an ML oracle, such as ML for LNS. Our empirical results highlighted that often the advantage of the ML policy is outweighed by its cost of inference on large MILPs. This calls for investigations of ML model architectures or hardware solutions that specifically target this challenge.

Synthetic Data Generation. Synthetic Data Generation (SDG) captures the underlying distribution of a dataset and synthesizes targeted data through a generative process [3]. SDG has been applied to finance [4] and healthcare [38] to address the problem of limited data or preserve the privacy of real data. SDG could also be used to improve ML-based methods for MILPs, as collecting algorithmic decision data from solving instances can be expensive, as discussed in Section 4. There has been existing work that uses data augmentation to generate MILP instances [58, 27, 80, 32] or algorithm decision data inside B&B [56]. *Distributional MIPLIB* could be used to develop theoretical and algorithmic frameworks that generate targeted data forming the same distributions.

Foundation Model for Combinatorial Optimization. Deep learning foundation models that leverage vast amounts of data to learn general-purpose representation can adapt to a wide range of downstream tasks, which has drastically transformed the domains of language, vision, and scientific discovery [13]. [54] took a step towards foundation models for MILP by using a LLM-based framework to generate MILPs and training a single model on a diverse set of MILP problems. Moreover, *Distributional MIPLIB* contains MILPs from a wide range of domains and hardness levels, which can be suited for a wide range of tracks (B&B, LNS, and finding primal solutions). Much of the existing works (e.g., learning for backdoors, LNS, and branching) use a common subset of features to learn a representation of MILP variables, which could be unified as a shared latent representation. *Distributional MIPLIB* could be used to develop and train such foundation models for the discrete optimization world.

6 Conclusion and Discussion

We introduce *Distributional MIPLIB*, a curated dataset of more than 35 MILP distributions from 13 synthetic and real-world domains, making it a large-scale resource for developing ML-guided MILP solving and comprehensive evaluation. Compared to existing datasets and generators, it provides data in distributional settings which is better suited for ML-guided methods. It provides MILP distributions from a wide range of applications and requires no domain knowledge to access these instances. We intend for the library to continue to grow with domain contributions from the community.

We ran experiments on Learn2Branch focused on variable selection policies in B&B. We identified that in past research only a few distributions/domains were used to assess state of the art, and evaluated the performance of Learn2Branch on unused domains, identifying open challenges. Moreover, we propose to train a Learn2Branch model with mixed distributions and show that this offers advantages in the low-data regime. We also identified potential future directions that can benefit from this library.

We also would like to acknowledge some limitations of our work. Due to computational constraints, we did not experiment with other GNN architectures, with a larger number of samples, or on better GPUs. These could change our empirical conclusions, but do not affect the value of the library.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[2] Alvarez, A. M., Louveaux, Q., and Wehenkel, L. (2017). A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195.

[3] An, S. and Jeon, J.-J. (2024). Distributional learning of variational autoencoder: Application to synthetic data generation. *Advances in Neural Information Processing Systems*, 36.

[4] Assefa, S. A., Dervovic, D., Mahfouz, M., Tillman, R. E., Reddy, P., and Veloso, M. (2020). Generating synthetic data in finance: opportunities, challenges and pitfalls. In *Proceedings of the First ACM International Conference on AI in Finance*, pages 1–8.

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

[6] Balcan, M.-F., Dick, T., Sandholm, T., and Vitercik, E. (2018). Learning to branch. In *International conference on machine learning*, pages 344–353. PMLR.

[7] Bergman, D., Cire, A. A., Van Hoeve, W.-J., and Hooker, J. (2016). *Decision diagrams for optimization*, volume 1. Springer.

[8] Berthold, T. (2006). *Primal heuristics for mixed integer programs*. PhD thesis, Zuse Institute Berlin (ZIB).

[9] Berthold, T. (2013). Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614.

[10] Berto, F., Hua, C., Park, J., Luttmann, L., Ma, Y., Bu, F., Wang, J., Ye, H., Kim, M., Choi, S., et al. (2023). Rl4co: an extensive reinforcement learning for combinatorial optimization benchmark. *arXiv preprint arXiv:2306.17100*.

[11] Bi, J., Ma, Y., Wang, J., Cao, Z., Chen, J., Sun, Y., and Chee, Y. M. (2022). Learning generalizable models for vehicle routing problems via knowledge distillation. *Advances in Neural Information Processing Systems*, 35:31226–31238.

[12] Bixby, R. E. and Rothberg, E. E. (2007). Progress in computational mixed integer programming - A look back from the other side of the tipping point. 149:37–41.

[13] Bodnar, C., Bruinsma, W. P., Lucic, A., Stanley, M., Brandstetter, J., Garvan, P., Riechert, M., Weyn, J., Dong, H., Vaughan, A., et al. (2024). Aurora: A foundation model of the atmosphere. *arXiv preprint arXiv:2405.13063*.

[14] Bolusani, S., Besançon, M., Bestuzheva, K., Chmiela, A., Dionísio, J., Donkiewicz, T., van Doornmalen, J., Eifler, L., Ghannam, M., Gleixner, A., Graczyk, C., Halbig, K., Hedtke, I., Hoen, A., Hojny, C., van der Hulst, R., Kamp, D., Koch, T., Kofler, K., Lentz, J., Manns, J., Mexi, G., Mühmer, E., Pfetsch, M. E., Schlösser, F., Serrano, F., Shinano, Y., Turner, M., Vigerske, S., Weninger, D., and Xu, L. (2024). The SCIP Optimization Suite 9.0. Technical report, Optimization Online.

[15] Cai, J., Huang, T., and Dilkina, B. N. (2024). Learning backdoors for mixed integer linear programs with contrastive learning. In *European Conference on Artificial Intelligence*.

[16] Cheng, C.-H., Nührenberg, G., and Ruess, H. (2017). Maximum resilience of artificial neural networks. In *15th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, pages 251–268. Springer.

[17] Chmiela, A., Khalil, E., Gleixner, A., Lodi, A., and Pokutta, S. (2021). Learning to schedule heuristics in branch and bound. *Advances in Neural Information Processing Systems*, 34:24235–24246.

[18] Colombi, M., Mansini, R., and Savelsbergh, M. (2017). The generalized independent set problem: Polyhedral analysis and solution approaches. *European Journal of Operational Research*, 260(1):41–55.

[19] Conrad, J., Gomes, C. P., Van Hoeve, W.-J., Sabharwal, A., and Suter, J. (2007). Connections in networks: Hardness of feasibility versus optimality. In *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pages 16–28. Springer.

[20] Cornuéjols, G., Sridharan, R., and Thizy, J.-M. (1991). A comparison of heuristics and relaxations for the capacitated plant location problem. *European journal of operational research*, 50(3):280–297.

[21] Dey, S. S. and Shah, P. (2022). Lower bound on size of branch-and-bound trees for solving lot-sizing problem. *Operations Research Letters*, 50(5):430–433.

[22] Ding, J.-Y., Zhang, C., Shen, L., Li, S., Wang, B., Xu, Y., and Song, L. (2020). Accelerating primal solution findings for mixed integer programs based on solution prediction. In *AAAI conference on Artificial Intelligence*, volume 34, pages 1452–1459.

[23] Dinur, I. and Safra, S. (2005). On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485.

[24] Ferber, A., Song, J., Dilkina, B., and Yue, Y. (2022). Learning pseudo-backdoors for mixed integer programs. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 91–102. Springer.

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

[26] Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32.

[27] Geng, Z., Li, X., Wang, J., Li, X., Zhang, Y., and Wu, F. (2024). A deep instance generative framework for milp solvers under limited data availability. *Advances in Neural Information Processing Systems*, 36.

[28] Gleixner, A., Bastubbe, M., Eifler, L., Gally, T., Gottwald, R. L., Hendel, G., Hojny, C., Koch, T., Lübbecke, M. E., Maher, S. J., et al. (2018). The scip optimization suite 6.0.

[29] Gleixner, A., Hendel, G., Gamrath, G., Achterberg, T., Bastubbe, M., Berthold, T., Christophel, P. M., Jarck, K., Koch, T., Linderoth, J., Lübbecke, M., Mittelmann, H. D., Ozyurt, D., Ralphs, T. K., Salvagnin, D., and Shinano, Y. (2021). MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*.

[30] Gomes, C. P., Van Hoeve, W.-J., and Sabharwal, A. (2008). Connections in networks: A hybrid approach. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 5th International Conference, CPAIOR 2008 Paris, France, May 20-23, 2008 Proceedings 5*, pages 303–307. Springer.

[31] Greening, L. M., Dahan, M., and Erera, A. L. (2023). Lead-time-constrained middle-mile consolidation network design with fixed origins and destinations. *Transportation Research Part B: Methodological*, 174:102782.

[32] Guo, Z., Li, Y., Liu, C., Ouyang, W., and Yan, J. (2024). Acm-milp: Adaptive constraint modification via grouping and selection for hardness-preserving milp instance generation. In *Forty-first International Conference on Machine Learning*.

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

[34] Gupta, P., Khalil, E. B., Chetélat, D., Gasse, M., Bengio, Y., Lodi, A., and Kumar, M. P. (2022). Lookback for learning to branch. *arXiv preprint arXiv:2206.14987*.

[35] Gurobi Optimization, LLC (2023). Gurobi Optimizer Reference Manual.

[36] Han, Q., Yang, L., Chen, Q., Zhou, X., Zhang, D., Wang, A., Sun, R., and Luo, X. (2022). A gnn-guided predict-and-search framework for mixed-integer linear programming. In *The Eleventh International Conference on Learning Representations*.

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

[38] Hernandez, M., Epelde, G., Alberdi, A., Cilla, R., and Rankin, D. (2022). Synthetic data generation for tabular health records: A systematic review. *Neurocomputing*, 493:28–45.

[39] Hochbaum, D. S. and Pathria, A. (1997). Forest harvesting and minimum cuts: a new approach to handling spatial constraints. *Forest Science*, 43(4):544–554.

[40] Huang, T. and Dilkina, B. (2020). Enhancing seismic resilience of water pipe networks. In *Proceedings of the 3rd ACM SIGCAS Conference on Computing and Sustainable Societies*, pages 44–52.

[41] Huang, T., Ferber, A. M., Tian, Y., Dilkina, B., and Steiner, B. (2023). Searching large neighborhoods for integer linear programs with contrastive learning. In *International Conference on Machine Learning*, pages 13869–13890. PMLR.

[42] Huang, T., Ferber, A. M., Zharmagambetov, A., Tian, Y., and Dilkina, B. (2024). Contrastive predict-and-search for mixed integer linear programs. In *International Conference on Machine Learning*. PMLR.

[43] Huang, W. and Khalil, E. B. (2023). Walkability optimization: formulations, algorithms, and a case study of toronto. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 14249–14258.

[44] Huang, Z., Wang, K., Liu, F., Zhen, H.-L., Zhang, W., Yuan, M., Hao, J., Yu, Y., and Wang, J. (2022). Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, 123:108353.

[45] Hubbs, C. D., Perez, H. D., Sarwar, O., Sahinidis, N. V., Grossmann, I. E., and Wassick, J. M. (2020). Or-gym: A reinforcement learning library for operations research problems.

[46] Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. (2016). Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

[47] Khalil, E. B., Dilkina, B., Nemhauser, G. L., Ahmed, S., and Shao, Y. (2017). Learning to run heuristics in tree search. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 659–666.

[48] Khalil, E. B., Morris, C., and Lodi, A. (2022a). Mip-gnn: A data-driven framework for guiding combinatorial solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10219–10227.

[49] Khalil, E. B., Vaezipoor, P., and Dilkina, B. (2022b). Finding backdoors to integer programs: A monte carlo tree search framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3786–3795.

[50] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

[51] Kody, A., Pianksy, R., Daniel, K., et al. (2022). Optimizing transmission infrastructure investments to support line de-energization for mitigating wildfire ignition risk. In *IREP Symposium on Bulk Power System Dynamics and Control-XI. A 100% Renewable Energy Source Bulk Power Grid: Opportunities and Challenges*.

[52] Labassi, A. G., Chételat, D., and Lodi, A. (2022). Learning to compare nodes in branch and bound with graph neural networks. *Advances in neural information processing systems*.

[53] Leyton-Brown, K., Pearson, M., and Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 66–76.

[54] Li, S., Kulkarni, J., Menache, I., Wu, C., and Li, B. (2025). Towards foundation models for mixed integer linear programming. *International Conference on Learning Representations*.

[55] Li, S., Ouyang, W., Paulus, M., and Wu, C. (2024). Learning to configure separators in branch-and-cut. *Advances in Neural Information Processing Systems*, 36.

[56] Lin, J., Xu, M., Xiong, Z., and Wang, H. (2024). Cambranch: Contrastive learning with augmented milps for branching. In *The Twelfth International Conference on Learning Representations*.

[57] Liu, D., Fischetti, M., and Lodi, A. (2022). Learning to search in local branching. In *Proceedings of the aaai conference on artificial intelligence*, volume 36, pages 3796–3803.

[58] Liu, H., Kuang, Y., Wang, J., Li, X., Zhang, Y., and Wu, F. (2023). Promoting generalization for exact solvers via adversarial instance augmentation. *arXiv preprint arXiv:2310.14161*.

[59] Lodi, A. (2013). The heuristic (dark) side of mip solvers. In *Hybrid metaheuristics*, pages 273–284. Springer.

[60] Lodi, A. and Zarpellon, G. (2017). On learning and branching: a survey. *Top*, 25(2):207–236.

[61] Magnanti, T. L. (1981). Combinatorial optimization and vehicle fleet planning: Perspectives and prospects. *Networks*, 11(2):179–213.

[62] Martello, S. and Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc.

[63] Mattick, A. and Mutschler, C. (2024). Reinforcement learning for node selection in branch-andbound. *Transactions on Machine Learning Research*.

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

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

[66] Paulus, M. B., Zarpellon, G., Krause, A., Charlin, L., and Maddison, C. (2022). Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International conference on machine learning*, pages 17584–17600. PMLR.

[67] Pisinger, D. (1999). An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114(3):528–541.

[68] Pisinger, D. and Ropke, S. (2019). Large neighborhood search. *Handbook of metaheuristics*, pages 99–127.

[69] Poljak, S. (1995). Integer linear programs and local search for max-cut. *SIAM Journal on Computing*, 24(4):822–839.

[70] Pollack, M., Piansky, R., Gupta, S., and Molzahn, D. (2025). Equitably allocating wildfire resilience investments for power grids — the curse of aggregation and vulnerability indices. *Applied Energy*, 388:125511.

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

[72] Samà, M., Corman, F., Pacciarelli, D., et al. (2017). A variable neighbourhood search for fast train scheduling and routing during disturbed railway traffic situations. *Computers & Operations Research*, 78:480–499.

[73] Santos Xavier, A., Qiu, F., Gu, X., Becu, B., and Dey, S. S. (2024). MIPLearn: An Extensible Framework for Learning- Enhanced Optimization.

[74] Scavuzzo, L., Chen, F., Chételat, D., Gasse, M., Lodi, A., Yorke-Smith, N., and Aardal, K. (2022). Learning to branch with tree mdps. *Advances in neural information processing systems*, 35:18514–18526.

[75] Song, J., Yue, Y., Dilkina, B., et al. (2020). A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems*, 33:20012–20023.

[76] Sonnerat, N., Wang, P., Ktena, I., Bartunov, S., and Nair, V. (2021). Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*.

[77] Tang, Y., Agrawal, S., and Faenza, Y. (2020). Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pages 9367–9376. PMLR.

[78] Tjeng, V., Xiao, K. Y., and Tedrake, R. (2018). Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*.

[79] Tomlin, J. A. (1971). An improved branch-and-bound method for integer programming. *Operations Research*, 19(4):1070–1075.

[80] Wang, H., Liu, J., Chen, X., Wang, X., Li, P., and Yin, W. (2024). Dig-milp: a deep instance generator for mixed-integer linear programming with feasibility guarantee. *Transactions on Machine Learning Research*.

[81] Williams, R., Gomes, C. P., and Selman, B. (2003). Backdoors to typical case complexity. In *IJCAI*, volume 3, pages 1173–1178.

[82] Wilson, J. (1992). Approaches to machine load balancing in flexible manufacturing systems. *Journal of the Operational Research Society*, 43:415–423.

[83] Wolsey, L. A. (1997). Mip modelling of changeovers in production planning and scheduling problems. *European Journal of Operational Research*, 99(1):154–165.

[84] Wu, Y., Song, W., Cao, Z., and Zhang, J. (2021). Learning large neighborhood search policy for integer programming. *Advances in Neural Information Processing Systems*, 34:30075–30087.

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

[86] Zhu, Y., Liu, J., Hu, Y., Xie, Y., Zeng, D., and Li, R. (2024). Distributionally robust optimization model considering deep peak shaving and uncertainty of renewable energy. *Energy*, 288:129935.

A Domain Abbreviations

The abbreviations for the domains are listed in Table 6.

Abbreviation	Domain	Reference
CA	Combinatorial Auctions	[53]
SC	Set Covering	[5]
MIS	Maximum Independent Set	[7]
MVC	Minimum Vertex Cover	[23]
GISP	Generalized Independent Set Problem	[39, 18]
CFLP	Capacitated Facility Location Problem	[20]
MK	Multiple Knapsack	[67]
MC	Max Cut	[69]
CORLAT	Wildlife Management Problem	[19, 30]
LB	Load Balancing	[82]
IP	Item Placement	[62]
MIRP	Maritime Inventory Routing Problem	[65]
NNV	Neural Network Verification	[16, 78]
OTS	Optimal Transmission Switching	[70]
MMCN	Middle-Mile Consolidation Network	[31]
SRPN	Seismic-Resilient Pipe Network Planning	[40]

Table 6: Abbreviation for domains.

B Literature Review

Learning to Branch A series of papers have explored learning to branch by imitating the strong branching heuristic, a branching method that results in fewer search tree nodes but is expensive to compute [46, 60, 2, 6, 26, 33, 64, 56]. The strong branching heuristic computes a score for each branching candidate and these methods either learn to predict the variables' score or learn to rank them according to their scores. For the features and ML models, [46] develop the first ML-based framework for learning to branch using a Support Vector Machine (SVM) with hand-crafted features. [26] extend the framework by using a bipartite graph to encode the MILP and Graph Convolution Networks (GCN) to learn variable representations.

Learning Backdoors Backdoor for MILPs is a small subsets of variables such that a MILP can be solved optimally by branching *only* on the variables in the set [81]. Therefore, identifying backdoors efficiently and effectively can greatly improve the performance of B&B. [24] using ML to predict the most effective backdoor candidates generated by a LP relaxation-based sampling methods. More recently, [15] propose to use a Monte-Carlo tree search method [49] to improve the quality of training data and apply contrastive learning to directly construct backdoors.

Learning Primal Heuristics Primal heuristics refer to routines that find good feasible solutions in a short amount of time [14] and deciding which heuristics to run and when is an important task. These decisions are mostly made by hard-coded frequency rules in MILP solvers, which are static, instance-oblivious, and context-independent. To tackle this challenge, [47] propose a data-driven approach to decide when to execute primal heuristics. [17] derive a data-driven approach for scheduling primal heuristics.

Another line of research is to learn to predict solutions to MILPs. Both [64] and [36] learn to predict optimal solutions to MILPs and fix the values for a subset of variables based on the prediction to get reduced-size MILPs that are faster to solve.

Large Neighbourhood Search (LNS) LNS is a meta-heuristic that can find high quality solutions faster than B&B on large-scale MILP instances but provides no optimality guarantees. It starts with a feasible solution to the MILP and iteratively selects a subset of variables to reoptimize. Local Branching (LB) is a heuristic that finds the variables that lead to the largest improvement over the current solution in each iteration of LNS. But LB is often slow since it needs to solve a MILP of the same size as input. To mitigate this issues, [76] and [41] replace LB with imitation-learned policies. Other ML techniques, such as reinforcement learning (RL), have also been applied to learn destroy heuristics for LNS [75, 84].

Learning to Cut A cutting-plane is a constraint that is valid for feasible integer solutions but cuts into the feasible region of the linear programming (LP) relaxation, thus improving the bound on the optimal solution. Adding cutting planes has been shown to speed up B&B [12, 21]. Modern MILP solvers maintain a cut-pool that includes a large number of cutting planes of a diverse set of classes. The decisions regarding which classes of cutting planes to use, as well as the specific cutting

planes to select from each class, significantly impact solver performance. In recent advancements, [77] introduce a Reinforcement Learning (RL) framework tailored for the Gomory cutting-plane algorithm. Additionally, [44] develop a method to approach cut selection as a learning-to-rank task, while [66] devise a strategy to imitate a lookahead strategy for cut selection.

C Experiment Details

We used the Learn2Branch implementation from [26] in our experiments. Their code is publicly available at `https://github.com/ds4dm/learn2branch`.

Setup. All experiments in Section 4 were conducted on a cluster with Intel Xeon Gold 6130 CPUs @ 2.10GHz and Nvidia Tesla V100 GPUs. Each method was run with 5 different seeds. For ML-based methods, we trained the model using 5 different seeds and solved the instances using the trained policies that correspond to the 5 training seeds. For the non-ML methods, we used SCIP to solve the instances with 5 different seeds. The results report the mean and standard deviation across these 5 seeds.

Data Collection. In the orginal implementation [26], expert samples were collected by sampling from a set of training instances with replacement and solving it with SCIP. They iterated this process until the desired number of expert samples was collected. Therefore, in their implementation, the whole set of training and validation instances was not necessarily used to collect samples. In our implementation, we collected a fixed number of expert samples (10) from each instances, to ensure that all instances in the training set were used.

Hyperparameters. We used the same GCN architecture as described in [26] and trained the models in TensorFlow [1]. We used the Adam Optimizer [50] with a batch size of 32 and an initial learning rate of 0.001. In case the when the validation loss does not decrease over a period of 10 epochs, the learning rate was reduced to 20% of its previous value.

D License of existing assets

We curated new assets from the following existing assets. The NNV dataset was downloaded from `https://github.com/google-deepmind/deepmind-research/tree/master/neural_mip_solving`, which is available under the terms of the Creative Commons Attribution 4.0 International (CC BY 4.0) license `https://creativecommons.org/licenses/by/4.0/legalcode`. Datasets downloaded from ML4CO (LB, IP, MIRP) are under BSD-3-Clause license `https://github.com/ds4dm/ml4co-competition/blob/main/LICENSE`. CA, SC, MIS, CFLP instances were generated using code from [26], available at `https://github.com/ds4dm/learn2branch?tab=readme-ov-file` under the MIT license `https://github.com/ds4dm/learn2branch?tab=MIT-1-ov-file`.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: This paper introduces a library developed for advancing ML-guided MILP methods and performed computational experiments using data in this library. The abstract and introduction (Section 1) reflects this.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitations of the work in Section 6.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: Our paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provided the information needed to reproduce the main experimental results in Appendix C. This includes the setup, data collection procedure, and model training details.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The experimental results in this paper are based on the library that we introduced, which is made publicly available in this submission. The code and instructions for running the experiments is included in the supplemental material.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: The data splits and how they were chosen are described in Section 4.1. The hyperparameters and optimizers used are described in Appendix C.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: All the experiments in Section 4 were ran with 5 random seeds, and the mean and standard deviation of results over 5 random seeds are reported in Table 3, Table 4, and Table 5. For ML-based methods, we trained the model using 5 different seeds and solved the instances using the trained policies that correspond to the 5 training seeds. For the non-ML methods, we used SCIP to solve the instances with 5 different seeds.

Guidelines:

- The answer NA means that the paper does not include experiments.

- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The type of works CPU and GPU are detailed in Appendix C, and the execution time is detailed in Section 4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The library we provide has minimal privacy concerns and we respect the terms of existing datasets that have defined licenses. We also discuss the impact of our dataset in the supplemental materials.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: We discuss the potential positive societal impacts and negative societal impacts of the work performed in the supplemental materials.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The paper poses no such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We provide the license of the data sources in `https://sites.google.com/usc.edu/distributional-miplib/license`.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We document the details of the dataset in `https://sites.google.com/usc.edu/distributional-miplib/home`.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: LLM is used only for writing and editing for this paper.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.