Learning to Configure Separators in Branch-and-Cut

Sirui Li[*]
MIT
siruil@mit.edu

Wenbin Ouyang[*]
MIT
oywenbin@mit.edu

Max B. Paulus
ETH Zürich
max.paulus@inf.ethz.ch

Cathy Wu
MIT
cathywu@mit.edu

Abstract

Cutting planes are crucial in solving mixed integer linear programs (MILP) as they facilitate bound improvements on the optimal solution. Modern MILP solvers rely on a variety of separators to generate a diverse set of cutting planes by invoking the separators frequently during the solving process. This work identifies that MILP solvers can be drastically accelerated by appropriately selecting separators to activate. As the combinatorial separator selection space imposes challenges for machine learning, we *learn to separate* by proposing a novel data-driven strategy to restrict the selection space and a learning-guided algorithm on the restricted space. Our method predicts instance-aware separator configurations which can dynamically adapt during the solve, effectively accelerating the open source MILP solver SCIP by improving the relative solve time up to 72% and 37% on synthetic and real-world MILP benchmarks. Our work complements recent work on learning to select cutting planes and highlights the importance of separator management.

1 Introduction

Mixed Integer Linear Programs (MILP) have been widely used in logistics [Demirel et al., 2016], management [Cheng et al., 2003], and production planning [Floudas and Lin, 2005]. Modern MILP solvers typically employ a Branch-and-Cut (B&C) framework that utilizes a Branch-and-Bound (B&B) tree search procedure to partition the search space. As illustrated in Fig. 1, cutting plane algorithms are applied within each node of the B&B tree, tightening the Linear Programming (LP) relaxation of the node and improving the lower bound.

This paper presents a machine learning approach to accelerate MILP solvers. Modern MILP solvers implement various cutting plane algorithms, also referred to as *separators*, to generate cutting planes that tighten the LP solutions. Different separators have varying performance and execution times depending on the specific MILP instance. Typical solvers use simple heuristics to select separators, which can limit the ability to exploit commonalities across problem instances. While there is a growing body of work considering the 'branch' and 'cut' aspects of B&C Gasse et al. [2019], Labassi et al. [2022], Tang et al. [2020], Paulus et al. [2022], profiling the open-source academic MILP solver SCIP [Bestuzheva et al., 2021], we find generating cutting planes through separators is a major contributor to the total solve time, and deactivating unused separators leads to faster solves and fewer B&B tree nodes. That is, a well-configured separator setup allows the selected cutting planes to more effectively tighten the LP solution, leading to fewer nodes in the B&B tree.

To our knowledge, the problem of how to leverage machine learning for this critical task of separator configuration, namely the selection of separators to activate and deactivate during the MILP solving

[*]Equal Contribution

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

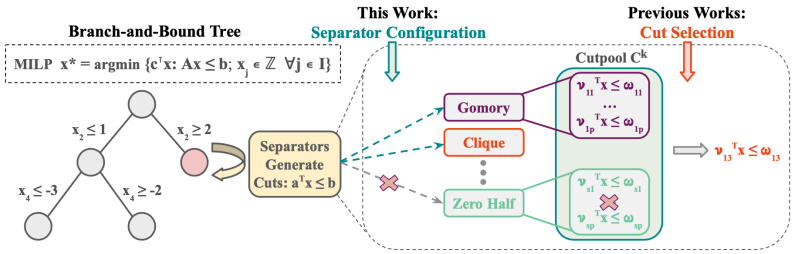


Figure 1: **Separator Configuration in Branch-and-Cut (B&C).** Modern MILP solvers perform Branch-and-Bound (B&B) tree search to solve MILPs. At each node of the B&B tree, cuts are added to tighten the Linear Programming (LP) relaxation of the MILP. To generate these cuts, a set of separators (e.g. Gomory) are invoked to first generate cuts into the cutpool $\mathcal{C}^k$. A subset of these cuts $\mathcal{P}^k \subseteq \mathcal{C}^k$ are then selected and added to the LP. The process is repeated for several separation rounds at each node. While previous works study cut selection from a pre-determined cutpool, this work focuses on the upstream task of separator configuration to generate a high quality cutpool efficiently.

process has not been considered. Therefore, the goal of this paper is to explore the extent to which tailoring the separator configuration to the MILP instance in a data-driven manner can accelerate MILP solvers. The central challenge comes from the high dimensionality of the configuration search space (induced by the large number of separators and configuration steps), which we address by introducing a data-driven search space restriction strategy that balances model fitting and generalization. We further propose a learning-guided algorithm, which is cast into the framework of neural contextual bandit, as an effective means of optimizing configurations within the reduced search space.

Our contributions can be summarized as

- We identify separator management as a crucial component in B&C, and introduce the Separator Configuration task for selecting separators to accelerate solving MILPs.
- To overcome the high dimensionality of the configuration task, we propose a data-driven strategy, directly informed by theoretical analysis, to restrict the search space. We further design a learning method to tailor instance-aware configurations within the restricted space.
- Extensive computational experiments demonstrate that our method achieves significant speedup over the competitive MILP solver SCIP on a variety of benchmark MILP datasets and objectives. Our method further accelerates the state-of-the-art MILP solver Gurobi and uncovers known facts from literature regarding separator efficacy for different MILP classes.

2 Related Work

The utilization of machine learning in MILP solvers has recently gained considerable attention. Various components in the B&B algorithm have been explored, including node selection [He et al., 2014, Song et al., 2018, Labassi et al., 2022], variable selection [Gasse et al., 2019, Gupta et al., 2020, Zarpellon et al., 2021], branching rule [Khalil et al., 2016, Gupta et al., 2020, Zarpellon et al., 2021, Scavuzzo et al., 2022], scheduling primal heuristics [Khalil et al., 2017, Hendel et al., 2019, Chmiela et al., 2021], and deciding whether to apply Dantzig-Wolfe decomposition [Kruber et al., 2017].

Our work is closely related to cutting plane selection, which can be achieved through heuristics [Wesselmann and Stuhl, 2012, Amaldi et al., 2014] or machine learning [Tang et al., 2020, Paulus et al., 2022]. The key difference, as shown in Figure 1 in Sec. 4, is that these works focus on selecting cutting planes from a pre-given cutpool generated by the available separators. That is, they consider the 'how to cut' question, whereas we focus on the equally crucial, but much less explored 'when (and what separators should we use) to cut' question Contardo et al. [2023], Dey and Molinaro [2018], Berthold et al. [2022]. For example, Wesselmann and Stuhl [2012] state that they do not use any additional scheme to deactivate specific separators. In contrast, our work configures separators to generate a high-quality cutpool.

Another closely related line of work is on algorithm selection and parameter configurations Xu et al. [2010, 2011], Balcan et al. [2021a,b], Hutter et al. [2009, 2011]. The most relevant works Xu et al.

[2011], Balcan et al. [2021b] consider portfolio-based algorithm selection by first choosing a subset of algorithm parameter settings, and then selecting a parameter setting for each problem instance from the portfolio. We specialize and extend the general framework to separator configuration, by proposing a novel data-driven subspace restriction strategy, followed by a learning method, to configure separators for multiple separation rounds. We further present a theoretical analysis that directly informs our subspace restriction strategy, whereas the generalization guarantees from the prior work Balcan et al. [2021b] is not informative for designing the portfolio-construction procedure.

It is common to restrict combinatorial space to improve the quality of solutions in discrete optimization. Previous research focuses primarily on decomposing large-scale problems, including heuristic works on Bender decomposition Rahmaniani et al. [2017] and column generation Barnhart et al. [1998], and recent learning-based works [Song et al., 2020, Li et al., 2021] that train networks to select among a set of random or heuristic decomposition strategies. Our data-driven action space restriction strategy is general and could be of interest for a broader set of combinatorial optimization tasks, as well as other applications such as recommendation systems.

3 MILP Background

Mixed Integer Linear Programming (MILP). A MILP can be written as $x^* = \arg\min\{c^\intercal x : Ax \leq b,\ x_j \in \mathbb{Z}\ \ \forall j \in I\}$, where $x \in \mathbb{R}^n$ is the set of n decision variables, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ formulate the set of m constraints, and $c \in \mathbb{R}^n$ formulates the linear objective function. $I \subseteq \{1, ..., n\}$ defines the integer variables. $x^* \in \mathbb{R}^n$ denotes the optimal solution to the MILP with an optimal objective value z^*.

Branch-and-Cut. State-of-the-art MILP solvers perform branch-and-cut (B&C) to solve MILPs, where a branch-and-bound (B&B) procedure is used to recursively partition the search space into a tree. Within each node of the B&B tree, linear programming (LP) relaxations of the MILP are solved to obtain lower bounds. B&C further invoke Cutting plane algorithms to tighten the LP relaxation.

Cutting Plane Separation. When the optimal solution x^*_{LP} to the LP relaxation is not a feasible solution to the original MILP, the cutting plane methods aim to find valid linear inequalities $\nu^\intercal x \leq \omega$ (cuts) that separate x^*_{LP} from the convex hull of all feasible solutions of the MILP. Cutting plane separation happens in rounds, where each round k consists of the following steps (1) solving the current LP relaxation, (2) calling different separators to generate a set of cuts and add them to the cutpool $\mathcal{C}_k$, (3) select a subset of cuts $\mathcal{P}_k \subseteq \mathcal{C}_k$ and update the LP with the selected cuts. Detailed background information on separators in the B&C framework can be found in Appendix A.1.

4 Problem Formulation

Different separators are designed to exploit different structures of the solution polytope defined by the MILP instance. The solution polytope also varies at different separation rounds, as changes to the constraints (e.g. after a branch) lead to different structures and thus different effective separators. Moreover, multiple separators can combine to exploit more sophisticated structures. The inherently combinatorial nature of the problem hence presents a challenge in assigning the appropriate separators to each MILP instances. This work aims to enhance the MILP solving process via intelligent separator configuration. We formally introduce the separator configuration task as follows.

Definition 1 (Separator Configuration). Suppose the MILP solver implements M different separator algorithms. Given a set $\mathcal{X}$ of N MILP instances (where $|\mathcal{X}| = N$), and a maximum number of separation rounds R in a MILP solving process, we want to select a configuration $s_{x,n} \in \{0, 1\}^M$ for each instance $x \in \mathcal{X}$ and separation round $1 \leq n \leq R$, where the w^{th} entry of $s_{x,n}$ equaling one means we activate the w^{th} separator in separation round n, and equaling zero means we deactivate the w^{th} separator in the corresponding round.

Figure 1 illustrates the separator configuration task and highlights the difference between our task and the downstream cutting plane selection task in previous works [Tang et al., 2020, Paulus et al., 2022].

We measure the success of an algorithm for the separator configuration task by the relative time improvement from SCIP's default configuration. Denote a proposed configuration policy as $\pi : \mathcal{X} \to \prod_{n=1}^{R}\{0, 1\}^M$, where for each MILP instance $x \in \mathcal{X}$, we have $\pi(x) = \{s_{x,1}, ..., s_{x,R}\}$ as the

proposed configurations. Let $t_\pi(x)$ be the solve time of instance x using the configuration sequence $\pi(x)$ and $t_0(x)$ be the solve time using the default SCIP configuration (both to optimality or a fixed gap). We evaluate the effectiveness of π by the relative time improvement

$$\Delta(\pi) := \mathbb{E}_{x\in\mathcal{X}}[\delta(\pi(x), x)] \quad \text{where} \quad \delta(\pi(x), x) := (t_0(x) - t_\pi(x))\,/t_0(x) \tag{1}$$

The search space for the separator configuration task is enormous, with a size of $N \times 2^{M\times R}$. SCIP contains $M = 17$ separators, and a typical solve run yields $R \geq 30$, making the task highly challenging. In the next section, we discuss our data-driven approach to finding high quality configurations.

5 Learning to Separate

Two sources of high dimensionality in the search space come from (1) combinatorial number $|O| = 2^M$ of configurations, where each element of $O := \{0, 1\}^M$ is a combination of separators (e.g., Gomory, Clique) to activate, and (2) a large number of configuration updates that results in the $|O|^R$ factor. We address the first challenge in Sec. 5.1 by restricting the number of configuration options, and the second challenge in Sec. 5.2 by reducing the frequency of configuration updates. The resulting restricted search space allows efficient learning in Sec. 5.3 to find high quality customized configurations for each MILP instance, which we term as instance-aware configurations.

5.1 Configuration space restriction

For simplicity, we first consider a single configuration update such that we apply the same configuration for all separation rounds, and our goal is to learn an instance-aware configuration predictor $\tilde{f} : \mathcal{X} \to O$. That is, we set $s_{x,1} = ... = s_{x,R} = \tilde{f}(x)$ for each $x \in \mathcal{X}$; Sec. 5.2 discusses extensions to multiple configuration updates. To address the challenge of learning the predictor in the high dimensional space O, we constrain the predictor $\tilde{f}_A$ to select from a subset $A \subseteq O$ of configurations with $|A|$ reasonably small, i.e. $\tilde{f}_A(x) \in A\ \forall x \in \mathcal{X}$. We design a data-driven strategy, supported by theoretical rationale, to identify a subspace A for $\tilde{f}_A$ to achieve high performance.

Preliminary definitions. Let $\mathcal{X}$ be a class of MILP instances, and $\mathcal{K} = \{x_1, ..., x_K\} \subseteq \mathcal{X}$ be a given training set where we can acquire the time improvement $\{\delta(s, x_i)\}_{i=1..K; s\in O}$. The true performance of $\tilde{f}_A$ on $\mathcal{X}$ is $\Delta(\tilde{f}_A) = \mathbb{E}_{x\in\mathcal{X}}[\delta(\tilde{f}_A(x), x)]$, and the empirical counterpart on $\mathcal{K}$ is $\hat{\Delta}(\tilde{f}_A) = \frac{1}{K}\sum_{i=1}^{K} \delta(\tilde{f}_A(x_i), x_i)$. We further denote the true instance-agnostic performance of applying a single configuration $s \in \{0, 1\}^M$ to all MILP instances as $\bar{\delta}(s) = \mathbb{E}_{x\in\mathcal{X}}[\delta(s, x)]$, and the empirical counterpart as $\hat{\bar{\delta}}(s) = \frac{1}{K}\sum_{i=1}^{K} \delta(s, x_i)$. Appendix A.2.1 details all relevant definitions.

Restriction algorithm. To find a subspace A that optimizes the true performance $\Delta(\tilde{f}_A)$ for the predictor $\tilde{f}_A$, we employ the following training performance v.s. generalization decomposition:

$$\Delta(\tilde{f}_A) = \underbrace{\hat{\Delta}(\tilde{f}_A)}_{\text{training perf.}} - \underbrace{(\hat{\Delta}(\tilde{f}_A) - \Delta(\tilde{f}_A))}_{\text{generalization}} \tag{2}$$

The first term measures how well $\tilde{f}_A$ performs on the training set $\mathcal{K}$, while the second term reflects the generalization gap of $\tilde{f}_A$ to the entire distribution $\mathcal{X}$. Notably, a similar trade-off exists in standard supervised learning Shalev-Shwartz and Ben-David [2014], where regularizations are used to balance fitting and generalization by implicitly restricting the hypothesis class. Relatedly, in this problem, we can balance the two terms by explicitly restricting the *output space* of the predictor. Intuitively, a larger subspace A can improve training performance (more configuration options to leverage), but hurt generalization (more options that could perform poorly on unseen instances). This intuition is formalized next.

First, since the second term in Eq. (2) is unobserved, the following proposition imposes assumptions that allow us to restrict the configuration space. A detailed proof can be found in Appendix A.2.2.

Proposition 1. Assume the predictor $\tilde{f}_A$, when evaluated on the entire distribution $\mathcal{X}$, achieves perfect generalization (i.e., zero generalization gap) with probability $1 - \alpha$; with probability α, the predictor makes mistake and outputs a configuration $s \in A$ uniformly at random. Then, the trainset performance v.s. generalization decomposition can be written as $\Delta(\tilde{f}_A) = (1 - \alpha)\hat{\Delta}(\tilde{f}_A) + \alpha\frac{1}{|A|}\sum_{s\in A} \bar{\delta}(s)$.

As $\bar{\delta}(s)$ is also unobservable, we further rely on its empirical counterpart $\hat{\bar{\delta}}_t(s)$ (see Appendix A.2.2 for a discussion of the reduction) and select the subspace A based on the following objective:

$$(1-\alpha)\hat{\Delta}(\tilde{f}_A) + \alpha \frac{1}{|A|} \sum_{s \in A} \hat{\bar{\delta}}(s) \tag{3}$$

The impact of the subspace A on these two terms further depends on the nature of $\tilde{f}_A$; we assume that the predictor $\tilde{f}_A$ uses empirical risk minimization (ERM) and performs optimally on the training set $\mathcal{K}$, i.e. $\tilde{f}_A^{ERM}(x) = \arg\max_{s \in A} \delta(s, x)\ \forall x \in \mathcal{K}$, hence bypassing the need to train any predictor for constructing A. The discussion of the ERM assumption's validity and the extension to predictors with training error are provided in Appendix A.2.4 (See Lemma 3 for the extension).

Eq. (3) then sheds light on how to construct a good A under the ERM assumption: an ideal subset A allows $\tilde{f}_A^{ERM}$ to have (1) high training performance $\hat{\Delta}(\tilde{f}_A^{ERM})$, obtained when *some* configuration in A achieves good performance for any MILP instance in a training set, and (2) low generalization gap, achieved when *each* configuration in A has good performance across MILP instances in a test set, which we approximate with the average instance-agnostic performance on the training set $\frac{1}{|A|}\sum_{s \in A} \hat{\bar{\delta}}(s)$. In fact, a larger or more diverse subspace A results in better $\hat{\Delta}(\tilde{f}_A^{ERM})$, as the ERM predictor can leverage more configuration options to improve the training set performance. Meanwhile, it may also lower $\frac{1}{|A|}\sum_{s \in A} \hat{\bar{\delta}}(s)$ which harms generalization, as we may include some configurations that perform poorly on most MILP instances but well on a small subset. The following proposition (proven in Appendix A.2.3) formalizes the diminishing marginal returns of $\tilde{f}_A^{ERM}$'s training performance with respect to A, which enables an efficient algorithm to construct A:

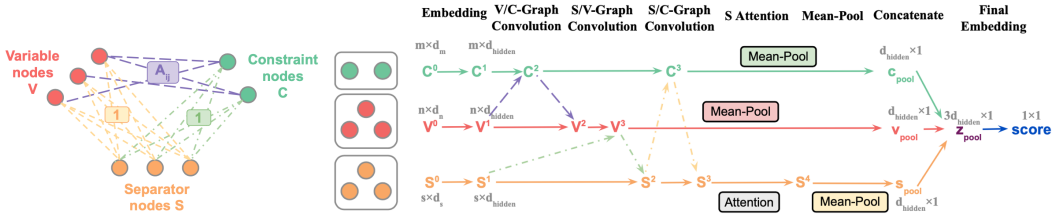
Proposition 2. The empirical performance of the ERM predictor $\hat{\Delta}(\tilde{f}_A^{ERM})$ is monotone submodular, and a greedy strategy where we include the configuration that achieves the greatest marginal improvement $\arg\max_{s \in \{0,1\}^M \setminus A} \hat{\Delta}(\tilde{f}_{A \cup \{s\}}^{ERM}) - \hat{\Delta}(\tilde{f}_A^{ERM})$ at each iteration is a $(1 - 1/e)$-approximation algorithm for constructing the subspace A that optimizes $\hat{\Delta}(\tilde{f}_A^{ERM})$.

To balance the two terms in Eq. (3), we couple the greedy selection strategy with a filtering criterion that eliminates configurations with poor instance-agnostic performance to construct the subspace A. Due to the high computational cost of calculating the marginal improvement for all 2^M configurations, we first sample a large set S of configurations, which we use to construct the subspace A. Then, at each iteration, we expand the current set A with the configuration that produces the best marginal improvement in training performance, but only considering configurations $s \in S$ whose empirical instance-agnostic performance is greater than a threshold, i.e. $\hat{\bar{\delta}}(s) > b$. The extra filtering procedure enables us to improve the second term with small concessions in the first term. We continue the process while monitoring the two opposing terms, and terminate with a reasonably small A that balances the trade-off. The detailed algorithm and discussions of the filtering and termination procedure are provided in Appendix A.3.

5.2 Configuration update restriction

Learning to update configurations at each separation round is challenging due to cascading errors from a large number of updates. Instead, we periodically update the configuration at a few intermediate rounds and hold it fixed between updates: we perform $k \ll R$ updates at rounds $\{n_j\}_{j=1}^k$ with $1 \leq n_j \leq R$, and set $s_{x,n_j} = ... = s_{x,n_{j+1}-1}$ for each $1 \leq j \leq k$ and $x \in \mathcal{X}$. Fig. 7 (Left) in Appendix A.4 shows an example of the configuration update restriction with $k = 2$ and $R = 6$, where we also discuss the trade-off of k in approximation v.s. estimation. We empirically find a small number of updates can already yield a decent time improvement (we set $k = 2$ in Experiment Sec. 6).

We use a forward training algorithm Ross and Bagnell [2010] to learn the configuration policy $\tilde{\pi}^{(k)} : \mathcal{X} \to \prod_{i=1}^k \{0,1\}^M$. The algorithm decomposes the sequential task into k single configuration update tasks $\tilde{\pi}^{(k)} = \{\tilde{f}_{\theta^T}^m\}_{m=1}^k$, where each $\tilde{f}_{\theta^T}^j : \mathcal{X} \to \{0,1\}^M$ is a separate network for the j-th configuration update. As illustrated in Fig.7 (Right) of Appendix A.4, at each iteration, we fix the weights of the trained networks for earlier updates $\{\tilde{f}_{\theta^T}^m\}_{m=1}^{j-1}$, and train the network $\tilde{f}_\theta^j$ for the j^{th} update. The detailed algorithm is provided in Alg. 2 of Appendix A.4. We incorporate the configuration space restriction in Sec. 5.1 by constraining each network $\tilde{f}_\theta^j$ to select configurations



(a) (b)

Figure 2: (a) **Our triplet graph encoding** of the MILP instance (the context) and the separator configuration (the arm / action). (b) **Our neural architecture** $\tilde{f}_\theta$**.** It involves three graph convolutions, an attention block for the separator nodes, and global poolings to extract the final score for reward prediction. We show the dimensionality of a tensor in gray if it is different from the previous size.

from a subset $A \subseteq O$, such that $\tilde{f}^j_\theta(x) \in A$ for all $x \in \mathcal{X}$. This reduces the search space from $N \times 2^{M \times R}$ to $N \times |A| \times k$, significantly easing the learning process. Notably, we construct the subspace A once at the initial update for computational efficiency benefits, as it yields comparable performances to constructing a new subspace for each update. Further details and discussions can be found in Appendix A.7.2.

5.3 Neural UCB algorithm

Given the restricted configuration space A, we frame each configuration update as a contextual bandit problem with A arms (configurations). Conditional on the context (a MILP instance $x \in \mathcal{X}$), each arm $s \in A$ has a reward (time improvement $\delta(s, x)$). We employ the neural UCB algorithm Zhou et al. [2020] to efficiently train a network $\tilde{f}^j_{\theta^0}(x, s) : \mathcal{X} \times A \to \mathbb{R}$ to estimate the reward, where the confidence bound estimation is enabled by the small size of A. We provide the complete training procedure in Alg. 3 of Appendix A.5. At each training epoch t, we randomly sample P instances from $\mathcal{X}$. For each instance, we sample D configurations based on the upper confidence bound $ucb(x, s)$, which combines a reward point estimate $\tilde{f}^j_{\theta^{t-1}}(x, s)$ and a confidence bound estimate. The confidence bound estimate incorporates the gradient $\nabla_\theta \tilde{f}^j_{\theta^{t-1}}(x, s)$ and a normalizing matrix Z_{t-1} only feasible to obtain when the number of arms is small. We run the MILP solver on each of the $P \times D$ pairs of instance-configuration and observe the reward labels. Lastly, we add all instance-configuration-reward tuples $\{x, s, r\}_{P \times D}$ to the data buffer and retrain the network $\tilde{f}^j_\theta$. At test time, we select the configuration with the highest predicted reward $s^*_{x,j} = \arg\max_{s \in A} \tilde{f}^j_{\theta^T}(x, s)$ or the highest UCB score $s^*_{x,j} = \arg\max_{s \in A} ucb(x, s)$, based on validation performance, at each update step n_j. We provide further details on the inference strategy in Appendix A.6.1.

Context encoding. We encode the context for each MILP instance x and separator configuration s as a triplet graph with three types of nodes in the graph: variable nodes $\mathbf{V}$, constraint nodes $\mathbf{C}$, and separator nodes $\mathbf{S}$. The variable and constraint nodes ($\mathbf{V}$, $\mathbf{C}$) appear in the previous works Gasse et al. [2019], Paulus et al. [2022]. We follow Paulus et al. [2022] to use the same input features for $\mathbf{V}$ and $\mathbf{C}$, and construct edges between them such that a variable node $\mathbf{V}_i$ is connected to a constraint node $\mathbf{C}_j$ if the variable appears in the constraint with the weight corresponds to the coefficient $\mathbf{A}_{ij} \neq 0$. The separator nodes $\mathbf{S}$ are unique to our problem. We represent each configuration s by M separator nodes; each node $\mathbf{S}_k$ has $M + 1$ dimensional input features, representing whether the separator is activated (the first dimension), and which separator it is (one-hot M-dimensional vector). We connect each separator node with all variable and constraint nodes, all with a weight of 1 for complete pairwise message passing. We provide detailed descriptions of the input features in Appendix A.5.

Neural architecture $\tilde{f}_\theta$**.** We extend the architecture in Paulus et al. [2022] for our network $\tilde{f}_\theta(x, s) : \mathcal{X} \times \mathcal{S} \to \mathbb{R}$. The architecture, as illustrated in Fig. 2, involves a Graph Convolutional Network (GCN) Kipf and Welling [2017], an attention block on the hidden embeddings of the separator nodes Shi et al. [2021], and a global pooling to output a single score for reward prediction. It first embeds $\mathbf{C}$, $\mathbf{V}$, and $\mathbf{S}$ input features into hidden representations, and performs message passing following the directions of $\mathbf{V} \to \mathbf{C} \to \mathbf{V}$, $\mathbf{S} \to \mathbf{V} \to \mathbf{S}$, and $\mathbf{S} \to \mathbf{C} \to \mathbf{S}$. Then, the $\mathbf{S}$ nodes pass through an attention module to emphasize the task of the separator configuration. Lastly, since we require the

Table 1: **Tang et al. and Ecole.** Absolute solve time of SCIP default, and the median (higher the better) and standard deviation (in parentheses) of relative time improvement of different methods.

		Tang			Ecole		
	Method	Bin. Pack.	Max. Cut	Pack.	Comb. Auc.	Indep. Set	Fac. Loc.
	Default Time (s)	0.076s (0.131s)	1.77s (0.56s)	8.82s (25.46s)	2.73s (4.43s)	8.21s (114.15s)	61.1s (55.37s)
Heuristic Baselines	Default	0%	0%	0%	0%	0%	0%
	Random	-23.4% (153.8%)	-108.4% (168.2%)	-91% (127.8%)	-48.6% (159.0%)	-5% (161.4%)	-33.3% (157.2%)
	Prune	13.9% (27.0%)	2.7% (26.2%)	6.6% (45.0%)	12.3% (24.2%)	18.0% (24.2%)	24.7% (47.9%)
Ours Heuristic Variants	Inst. Agnostic Configuration	33.7% (36.6%)	69.8% (10.5%)	20.1% (38.0%)	60.1% (27.6%)	57.8% (29.5%)	11.5% (21.8%)
	Random within Restr. Subspace	26.9% (33.6%)	68.0% (11.0%)	18.8% (38.7%)	58.1% (28.7%)	57.4% (75.8%)	17.7% (33.0%)
Ours Learned	L2Sep	**42.3% (34.2%)**	**71.9% (11.3%)**	**28.5% (39.3%)**	**66.2% (26.2%)**	**72.4% (27.8%)**	**29.4% (39.6%)**

model to output a single score (in contrary to Paulus et al. [2022] which outputs a score for each cut node), we perform a global mean pooling on each of the **C**, **V**, and **S** hidden embeddings to obtain three embedding vectors, concatenate them into a single vector, and finally use a multilayer perceptron (MLP) to map the vector into a scalar.

Clipped Reward Label. To account for variations in MILP solve time, we perform l MILP solver runs for each configuration-instance pair (s, x) and take the average time improvement as the unclipped reward label. Additionally, if a certain configuration s takes significantly longer solve time than SCIP default on a MILP instance x, we terminate the MILP solver run when the relative time improvement is less than a predefined threshold $r_{\text{min}} \ll 0$ to expedite data collection, and assign a clipped reward label of $r^{\text{clip}}(s, x) = (\sum_{i=1..l} \max\{\delta^{(i)}(s, x), r_{\text{min}}\})/l$. Reward clipping also simplifies learning by obviating the need to accurately fit the exact value of extreme negative improvements, which may skew the network's prediction. As long as the prediction's sign is right, we will not select such a configuration with a negative predicted value during testing.

Loss function $\mathcal{L}$. We use a L_2 loss between the prediction $\tilde{f}_\theta(x, s)$ and the clipped reward label r^{clip}:

$$L(\tilde{f}_\theta(x, s), r) = (\tilde{f}_\theta(x, s) - r^{\text{clip}})^2 \tag{4}$$

6 Experiments and Analysis

We divide the experiment section into two main parts. First, we evaluate our method on standard MILP benchmarks from Tang et al. [2020] and Ecole Prouvost et al. [2020], where the number of variables and constraints range from 60 to $10,000$. We conduct detailed ablation studies to validate the design choices made for our method. Second, we examine the efficacy of our method by applying it to large-scale real-world MILP benchmarks, including the MIPLIB Gleixner et al. [2021], NN Verification Nair et al. [2020], and Load Balancing in the ML4CO challenges Gasse et al. [2022], where the number of variables and constraints reaches up to $65,000$. We omit certain MILP classes from the benchmarks with excessively short solve times, few generated cutting planes, or small dataset sizes. Appendix A.6.5 provides a detailed description of the datasets.

6.1 Setup

Evaluation Metric. As we aim to accelerate SCIP solving through separator configuration, we evaluate our learned configuration by the relative time improvement from SCIP default, defined in Eq. (1), when both are solved to optimality (for standard instances) or a fixed gap (for large-scale instances) as described in Appendix A.6.4. We report the median and standard deviation across all test instances, and defer mean and interquartile mean to Appendix A.8 as they yield similar results.

Table 2: **Detailed ablations of different components in our L2Sep algorithm.** Learning with neural UCB in the restricted config. space and performing $k = 2$ config. updates achieves the best result.

	Config. Space Restriction		**Config. Update Restriction**		**Neural Contextual Bandit**		**Ours: L2Sep**
Ablation Method	No Restr.	Greedy Restr.	$k = 1$	$k = 3$	Supervise ($\times 4$)	ϵ-greedy	w/ Restr. + $k = 2$ + UCB
Bin. Pack.	18.6% (125.3%)	35.8% (35.6%)	40.4% (42.3%)	44.2% (32.4%)	40.2% (19.7%)	36.3% (32.3%)	42.3% (34.2%)
Pack.	19.6% (61.1%)	18.4% (49.8%)	23.8% (38.1%)	27.8% (38.1%)	24.0% (44.1%)	25.1% (44.3%)	28.5% (39.3%)
Indep. Set	38.6% (23.5%)	68.5% (28.1%)	70.2% (38.6%)	69.7% (29.1%)	68.7% (33.9%)	64.1% (48.7%)	72.4% (27.8%)
Fac. Loc.	15.5% (121.2%)	27.1% (38.7%)	20.1% (37.8%)	29.7% (29.8%)	31.0% (41.5%)	28.1% (23.6%)	29.4% (39.6%)

ML Setup. We train the networks with ADAM [Kingma and Ba, 2014] under a learning rate of 10^{-3}. The reward label collection is performed via multi-processing with 48 CPU processes. As in previous works [Tang et al., 2020, Paulus et al., 2022, Wang et al., 2023], we train separate models for each MILP class. By default, we generate a training set $\mathcal{K}_{small}$ of 100 instances for configuration space restriction, another training set $\mathcal{K}_{large}$ of 800 for predictor network training, a validation set of 100 instances, and a test set of 100 instances for each class Appendix A.6 provides full details of the setup.

Baselines. To our knowledge, our separator configuration task has not been explored in previous research. We design the following baselines to assess the effectiveness of our proposed methods: (1) **Default**, where we run SCIP with the default parameters; (2) **Random**, where for each MILP instance x, we randomly sample a configuration $s \in \{0, 1\}^M$; (3) **Prune**, where we first run SCIP default on the $\mathcal{K}_{small}$, and then at test time, we deactivate separators whose generated cutting planes are never applied to any instances in $\mathcal{K}_{small}$.

Proposed Methods. We evaluate the performances of our complete method and its sub-components: (1) **Ours (L2Sep)**, where we perform $k = 2$ instance-aware configuration updates per MILP instance (Sec. 5.2). We use forward training to learn predictors via the neural UCB algorithm (Sec. 5.3) within the restricted configuration subspace A (Sec. 5.1). (2) **Instance Agnostic Configuration**, where we select a single configuration $\tilde{s}$ with the best instance-agnostic performance $\hat{\tilde{\delta}}(\tilde{s})$ on $\mathcal{K}_{small}$ from the initial large subset S for our space restriction algorithm ($|S| \approx 2000$); $\tilde{s}$ is included in A. (3) **Random within Restricted Subspace**, where for each MILP instance, we select a random configuration within A. The latter two sub-components assess the quality of the restricted subspace and the benefit of learning instance-aware configurations. Further details can be found in Appendix A.6.1.

6.2 Standard MILP Benchmarks with Detailed Ablations

Performance. Table 1 presents the relative time improvement of different methods over SCIP default, on the datasets of Tang et al. and Ecole. Our method demonstrates a substantial speed up from SCIP default across all MILP classes, with a relative time improvement ranging from 25% to 70%. In contrast, the random baseline performs poorly, demonstrating that separator configuration is a nontrivial task. Meanwhile, although the pruning baseline generally outperforms SCIP default, its time improvement is significantly less than ours, confirming the efficacy of our proposed algorithm. Notably, both of our two heuristic sub-components achieve impressive speed-up from SCIP default, indicating the high quality of our restricted subspace (and a configuration within) to accelerate SCIP; additionally, our complete learning method outperforms the sub-components on all MILP classes, further underscoring the advantages of learning for instance-aware configurations.

We note that the high standard deviation, exhibited in all methods including SCIP default and also observed in the recent studies Wang et al. [2023], is reasonable due to instance heterogeneity, as the standard deviation is calculated based on the time improvements across instances within each MILP dataset.

Table 3: **Real-world MILPs.** Absolute solve time of SCIP default, and the median (higher the better) and standard deviation (in parentheses) of relative time improvement of different methods.

Methods	Default Times (s)	Heuristic Baselinses			Ours Heuristic Variants		Ours Learned
		Default	Random	Prune	Inst. Agnostic Configuration	Random within Restr. Subspace	L2Sep
MIPLIB	25.08s (57.05s)	0%	-149.1% (149.7%)	4.8% (107.6%)	5.5% (71.5%)	1.9% (74.9%)	**12.9% (73.1%)**
NN Verification	31.42s (22.44s)	0%	-300.0% (152.3%)	31.5% (36.3%)	31.4% (38.3%)	30.7% (34.1%)	**37.5% (33.9%)**
Load Balancing	31.86s (7.07s)	0%	-300.1% (129.5%)	21.1% (150.8%)	10.4% (8.5%)	10.0% (31.5%)	**21.2% (20.3%)**

Ablations. In Table 2, we further conduct comprehensive ablation studies to assess the effectiveness of our learning method. The ablations are performed on four representative MILP classes in Ecole and Tang, covering a wide range of problem sizes and solve times. Appendix A.7 provides detailed descriptions as well as additional ablation results. We aim to answer the following questions: (i) Does the restricted config. space improve learning performance? (ii) How does the performance vary with fewer or more updates? (iii) Does the use of neural UCB lead to efficient predictor learning?

(i) Configuration space restriction (Sec. 5.1). We train our configuration predictors to select within a restricted subspace A constructed by a greedy strategy coupled with a filtering criterion. To evaluate the importance of the space restriction in learning high quality predictors, we perform an ablation study where we train the predictors to select within (1) the unrestricted space $O = \{0, 1\}^M$ (**No Restr.**), and (2) a same-sized subspace A' constructed solely by the greedy strategy without filtering (**Greedy Restr.**). The restricted search space substantially enhances the learned predictors when compared to **No Restr.**, improving the median performance and lowering the standard deviation. We also observe the benefit of the filtering criterion when compared to **Greedy Restr.**. The filtering criterion excludes configurations with subpar instance-agnostic performance from entering the restricted configuration space, improving model generalization as demonstrated in our theoretical analysis.

(ii) Configuration update restriction (Sec. 5.2). We apply the forward training algorithm (Sec. 5.2) to perform two configuration updates ($k = 2$) for each MILP instance. To examine the impact of fewer or more updates, we conduct an ablation study where we (1) performed a single update at round $n_1 = 0$ ($\mathbf{k = 1}$), and (2) added an additional third update at a later round n_3 ($\mathbf{k = 3}$). The results show that while a single update yields decent time improvement, adding the second update leads to further time savings. Meanwhile, we observe little improvement from the third update ($\mathbf{k = 3}$). We speculate that this is because the performance improvement primarily occurs during the early stages of a solve, and holding a fixed configuration for longer may be advantageous by making the solve process more stable. We leave further investigation of more configuration updates as a future work.

(iii) Neural UCB algorithm (Sec. 5.3). Our method employs the online neural UCB algorithm to improve training efficiency for configuration predictors. We present the ablation (1) where we train the predictor using an offline regression dataset whose size is four times as ours while training the model until convergence (**Supervise** ($\times 4$)); we conduct an additional ablation (2) where we train the predictor using neural contextual bandit with ϵ-greedy exploration strategy (ϵ-**greedy**). Our model performs comparably to **Supervise** ($\times 4$) while using significantly fewer data, highlighting the importance of the contextual bandit for improving training efficiency by collecting increasingly higher quality datasets online. The ablation results with ϵ-**greedy** further confirm the benefit of the confidence bound estimation in neural UCB for more efficient contextual bandit exploration.

6.3 Large-scale Real-world MILP Benchmarks

The real-world datasets of MIPLIB, NN Verification, and Load Balancing present significant challenges due to the vast number of variables and constraints (on the order of 10^4), including nonstandard constraint types that MILP separators are not designed to handle. MIPLIB imposes a further challenge of dataset heterogeneity, as it contains a diverse set of instances from various application domains. Prior research Turner et al. [2022] struggles to learn effectively on MIPLIB due to this heterogeneity, and a recent study Wang et al. [2023] attempts to learn cutting plane selection over two homogeneous subsets (with 20 and 40 instances each). In contrast, we attempt to learn separator configuration

Table 4: **Gurobi as the MILP Solver.** Absolute solve time of Gurobi default, and the median (higher the better) and standard deviation (in parentheses) of relative time improvement of different methods.

Methods	Default Times (s)	Heuristic Baselinses		Ours Heuristic Variants		Ours Learned
		Default	Random	Inst. Agnostic Configuration	Random within Restr. Subspace	L2Sep
Max. Cut	0.087s (0.051s)	0%	18.6% (49.0%)	35.1% (35.8%)	37.3% (48.0%)	**45.4% (38.4%)**
Pack.	4.048s (3.216s)	0%	15.5% (28.2%)	22.9% (39.4%)	24.3% (32.2%)	**30.6% (29.6%)**
Comb. Auc.	1.687s (3.596s)	0%	-10.7% (69.1%)	3.1% (65.3%)	5.1% (84.2%)	**12.6% (63.5%)**
Fac. Loc.	27.872s (14.733s)	0%	13.4% (46.0%)	40.6% (48.1%)	40.2% (46.8%)	**56.7% (35.7%)**

across a larger MIPLIB subset that includes 443 of the 1065 instances in the original set, while carefully preserving the heterogeneity of the dataset. We provide our subset curation procedure in Appendix A.6.5.

Main Results. Table 3 presents the relative time improvement of various methods over SCIP default, on the large-scale real-world datasets. Again, our complete method displays a substantial speed up from SCIP default with a relative time improvement ranging from 12% to 37%. Our method also improves from our heuristic sub-components, further indicating the efficacy of our learning component on the challenging datasets. In contrast, the random baseline fails to improve from SCIP default, while the pruning baseline, despite having a reasonable median performance, suffers from a high standard deviation due to poor performance on many instances (See Appendix A.8.1 for IQM and mean results). Our results show the effectiveness of our learning method in improving the efficiency of practical applications that involve large-scale MILP optimization.

Although not a perfect comparison, for reference, we attempt to contextualize our result by examining the time improvement in the most comparable setting we found, which we provide comparison details in Appendix A.8.2: the learning method for cutting plane selection in Paulus et al. [2022] achieves a median relative time improvement of 11.67% on the NN Verification dataset, and that in Wang et al. [2023] obtains a 3% and 1% improvement in the solve time on two small homogeneous MIPLIB subsets. While the comparison is far from perfect, our learning method for separator configuration achieves much higher time improvements of 37.5% on NN Verification and 12.9% on MIPLIB.

6.4 Interpretation Analysis: L2Sep Recovers Effective Separators from Literature

Bin Packing: It is known that instances with few bins approximate the Knapsack problem (Clique cuts are known to be effective Boland et al. [2012]), and that instances with many bins approximate Bipartite Matching (Flowcover cuts can be useful Van Vyve [2011]). We analyze the separators activated by L2Sep when we gradually decrease the number of bins, and observe that the prevalence of selected Clique and Flowcover cuts increased and decreased, respectively. This is illustrated in Fig 8 in Appendix A.8.3.

Other MILP Classes: We provide visualizations and interpretations for other MILP classes in Appendix A.8.3. Notably, Clique is known to be effective for Indep. Set Dey and Molinaro [2018]; L2Sep recovers this fact by frequently selecting configurations that activate Clique. Meanwhile, L2Sep discovers the instance heterogeneity of MIPLIB, resulting in a more dispersed distribution of selected configurations.

6.5 State-of-the-art MILP Solver Gurobi

We apply our method L2Sep with Gurobi, which contains a larger set of 21 separators. As Gurobi is closed-source, we cannot change configurations after the solving process starts, so we only consider one stage of separator configuration ($k = 1$). As seen from Table 4, L2Sep achieves significant relative time improvements over the Gurobi default, with gains ranging from 12% to 56%. This result confirms the efficacy of L2Sep as an automatic instance-aware separator configuration method.

6.6 Additional Results

In Appendix A.8.5 and A.8.4, we further demonstrate (1) Separator Configuration has immediate and multi-step effects in the B&C Process. For instance, even though L2Sep does not modify branching, the branching solve time is reduced. (2) L2Sep is effective under an alternative objective, achieving 15%-68% relative gap improvements under fixed time limits.

7 Conclusion

This work identifies the opportunity of managing separators to improve MILP solvers, and further formulates and designs a learning-based method for doing so. We design a data-driven strategy, supported by theoretical analysis, to restrict the combinatorial space of separator configurations, and overall find that our learning method is able to improve the relative solve time (over the default solver) from 12% to 72% across a range of MILP benchmarks. In future work, we plan to apply our algorithm to more challenging MILP problems, particularly those that cannot be solved to optimality. We also aim to learn more fine-grained controls by increasing the frequency of separation configuration updates. Our algorithm is highly versatile, and we plan to investigate its potential to manage aspects of the MILP solvers, and further integrate with previous works on cutting plane selection. Our code is publicly available at `https://github.com/mit-wu-lab/learning-to-configure-separators`. We believe that our learning framework can be a powerful technique to enhance MILP solvers.

Acknowledgments and Disclosure of Funding

The authors would like to thank Mark Velednitsky and Alexandre Jacquillat for insightful discussions regarding an interpretative analysis of the learned model. This work was supported by a gift from Mathworks, the National Science Foundation (NSF) CAREER award (#2239566), the MIT Amazon Science Hub, and MIT's Research Support Committee. The authors acknowledge the MIT SuperCloud and Lincoln Laboratory Supercomputing Center for providing HPC resources that have contributed to the research results reported within this paper.

References

Tobias Achterberg. *Constraint integer programming*. PhD thesis, 2007.

Edoardo Amaldi, Stefano Coniglio, and Stefano Gualandi. Coordinated cutting plane generation via multi-objective separation. *Mathematical Programming*, 143:87–110, 2014.

Egon Balas, Sebastian Ceria, Gérard Cornuéjols, and N Natraj. Gomory cuts revisited. *Operations Research Letters*, 19(1):1–9, 1996.

Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? generalization guarantees for data-driven algorithm design. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 919–932, 2021a.

Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Generalization in portfolio-based algorithm selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12225–12232, 2021b.

Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

Timo Berthold, Matteo Francobaldi, and Gregor Hendel. Learning to use local cuts. *arXiv preprint arXiv:2206.11618*, 2022.

Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, et al. The scip optimization suite 8.0. *arXiv preprint arXiv:2112.08872*, 2021.

Natashia Boland, Andreas Bley, Christopher Fricke, Gary Froyland, and Renata Sotirov. Clique-based facets for the precedence constrained knapsack problem. *Mathematical programming*, 133: 481–511, 2012.

Endre Boros, Yves Crama, and Peter L. Hammer. Chvátal cuts and odd cycle inequalities in quadratic 0–1 optimization. *SIAM Journal on Discrete Mathematics*, 5(2):163–177, 1992.

Alberto Caprara and Matteo Fischetti. $\{0, 1/2\}$-chvátal-gomory cuts. *Mathematical Programming*, 74:221–235, 1996.

Steven Cheng, Christine W Chan, and Gordon H Huang. An integrated multi-criteria decision analysis and inexact mixed integer linear programming approach for solid waste management. *Engineering Applications of Artificial Intelligence*, 16(5-6):543–554, 2003.

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

Claudio Contardo, Andrea Lodi, and Andrea Tramontani. Cutting planes from the branch-and-bound tree: Challenges and opportunities. *INFORMS Journal on Computing*, 35(1):2–4, 2023.

Eray Demirel, Neslihan Demirel, and Hadi Gökçen. A mixed integer linear programming model to optimize reverse logistics activities of end-of-life vehicles in turkey. *Journal of Cleaner Production*, 112:2101–2113, 2016.

Santanu S Dey and Marco Molinaro. Theoretical challenges towards cutting-plane selection. *Mathematical Programming*, 170:237–266, 2018.

Christodoulos A Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139:131–162, 2005.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

Maxime Gasse, Simon Bowly, Quentin Cappart, Jonas Charfreitag, Laurent Charlin, Didier Chételat, Antonia Chmiela, Justin Dumouchelle, Ambros Gleixner, Aleksandr M Kazachkov, et al. The machine learning for combinatorial optimization competition (ml4co): Results and insights. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 220–231. PMLR, 2022.

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

Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

Zonghao Gu, George L Nemhauser, and Martin WP Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. *Mathematical Programming*, 85:439–467, 1999.

Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33: 18087–18097, 2020.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL `https://www.gurobi.com`.

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

Gregor Hendel, Matthias Miltenberger, and Jakob Witzig. Adaptive algorithmic behavior for solving mixed integer programs using bandit algorithms. In *Operations Research Proceedings 2018: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Brussels, Belgium, September 12-14, 2018*, pages 513–519. Springer, 2019.

Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

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

Michael Jünger and Sven Mallach. Exact facetial odd-cycle separation for maximum cut and binary quadratic optimization. *INFORMS Journal on Computing*, 33(4):1419–1430, 2021.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Ijcai*, pages 659–666, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Markus Kruber, Marco E Lübbecke, and Axel Parmentier. Learning when to use a decomposition. In *Integration of AI and OR Techniques in Constraint Programming: 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings 14*, pages 202–210. Springer, 2017.

Abdel Ghani Labassi, Didier Chételat, and Andrea Lodi. Learning to compare nodes in branch and bound with graph neural networks. *arXiv preprint arXiv:2210.16934*, 2022.

Sirui Li, Zhongxia Yan, and Cathy Wu. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34:26198–26211, 2021.

Stephen Maher, Matthias Miltenberger, João Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano. PySCIPOpt: Mathematical programming in python with the SCIP optimization suite. In *Mathematical Software – ICMS 2016*, pages 301–307. Springer International Publishing, 2016. doi: 10.1007/978-3-319-42432-3_37.

Alberto Maria Metelli, Flavio Mazzolini, Lorenzo Bisi, Luca Sabbioni, and Marcello Restelli. Control frequency adaptation via action persistence in batch reinforcement learning. In *International Conference on Machine Learning*, pages 6862–6873. PMLR, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

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

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14:265–294, 1978.

Max B Paulus, Giulia Zarpellon, Andreas Krause, Laurent Charlin, and Chris Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International conference on machine learning*, pages 17584–17600. PMLR, 2022.

Antoine Prouvost, Justin Dumouchelle, Lara Scavuzzo, Maxime Gasse, Didier Chételat, and Andrea Lodi. Ecole: A gym-like library for machine learning in combinatorial optimization solvers. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020. URL https://openreview.net/forum?id=IVc9hqgibyB.

Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3): 801–817, 2017.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

Lara Scavuzzo, Feng Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen Aardal. Learning to branch with tree mdps. *Advances in Neural Information Processing Systems*, 35:18514–18526, 2022.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1548–1554, 8 2021.

Jialin Song, Ravi Lanka, Albert Zhao, Aadyot Bhatnagar, Yisong Yue, and Masahiro Ono. Learning to search via retrospective imitation. *arXiv preprint arXiv:1804.00846*, 2018.

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

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pages 9367–9376. PMLR, 2020.

Mark Turner, Thorsten Koch, Felipe Serrano, and Michael Winkler. Adaptive cut selection in mixed-integer linear programming. *arXiv preprint arXiv:2202.10962*, 2022.

Mathieu Van Vyve. Fixed-charge transportation on a path: Linear programming formulations. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 417–429. Springer, 2011.

Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *The Eleventh International Conference on Learning Representations*, 2023.

Franz Wesselmann and Uwe Stuhl. Implementing cutting plane management and selection techniques. In *Technical Report*. University of Paderborn, 2012.

Lin Xu, Holger Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 210–216, 2010.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*, pages 16–30, 2011.

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

Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural contextual bandits with ucb-based exploration. In *International Conference on Machine Learning*, pages 11492–11502. PMLR, 2020.

A Supplementary Material: *Learning to Separate in Branch-and-Cut*

Contents

A.1 MILP and Branch-and-Cut Background

Mixed Integer Linear Programming (MILP). A MILP can be written as

$$x^* = \arg\min\{c^\intercal x : Ax \leq b,\ x_j \in \mathbb{Z}\ \ \forall j \in I\} \tag{5}$$

where $x \in \mathbb{R}^n$ is a set of n decision variables, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ formulate a set of m constraints, and $c \in \mathbb{R}^n$ formulates the linear objective function. $I \subseteq \{1, ..., n\}$ defines the variables that are required to be integral. $x^* \in \mathbb{R}^n$ denotes the optimal solution to the MILP with an optimal objective value z^*.

Branch-and-Cut. State-of-the-art MILP solvers perform branch-and-cut (B&C) to solve MILPs, where a branch-and-bound (B&B) procedure is used to recursively partition the search space into a tree. Within each node of the B&B tree, linear programming (LP) relaxations of Eq. 5 are solved to obtain lower bounds to the MILP. Specifically, a LP relaxation of Eq. (5) can be written as

$$x^*_{LP} = \arg\min\{c^\intercal x : Ax \leq b,\ x \in \mathbb{R}^n\} \tag{6}$$

where $x^*_{LP} \in \mathbb{R}^n$ denotes the optimal solution to the LP with an optimal objective value z^*_{LP} such that $z^*_{LP} \leq z^*$.

Cutting Plane Separation. Each node of the B&B tree uses cutting plane algorithms to tighten the LP relaxation. When x^*_{LP} in Eq. (6) does not satisfy $(x^*_{LP})_j \in \mathbb{Z}\ \ \forall j \in I$, it is not a feasible solution to the original MILP. The cutting plane methods aim to find valid linear inequalities $\nu^\intercal x \leq \omega$ (cuts) that separate x^*_{LP} from the convex hull of all feasible solutions of the MILP. Namely, a cut satisfies $\nu^\intercal x^*_{LP} > \omega$, and $\nu^\intercal x \leq \omega$ for each feasible solution x to the MILP. Adding cuts into the LP tightens the relaxation, leading to a better lower bound to the MILP.

Cutting plane separation happens in rounds, where each separation round k consists of the following steps (1) solving the current LP relaxation, (2) if separation conditions are satisfied, calling different separators to generate a set of cuts and add them to the cutpool $\mathcal{C}_k$, (3) select a subset of cuts $\mathcal{P}_k \subseteq \mathcal{C}_k$ and update the LP with the selected cuts.

Typical MILP solvers, such as SCIP [Bestuzheva et al., 2021] and Gurobi [Gurobi Optimization, LLC, 2023], maintain a set of separators such as Gomory Balas et al. [1996] and Flow cover Gu et al. [1999] to generate cuts. Each separator in SCIP has a priority and a frequency attribute, and once invoked, generates a set of cuts that are added to the cutpool $\mathcal{C}_k$. The frequency decides the depth level of the B&B tree node in which the separator is invoked (typically the root node and all other nodes with depth divisible by some constant d). The priority decides the order of the separators to be invoked; in each separation round, separators are invoked with a descending priority order until a predefined maximal number of cuts max_{cuts} are generated. Separators with low priority may not be invoked during a separation round. By default, the priorities and frequency attributes in SCIP are a set of predefined values that remain unchanged for all MILP instances.

Benefits of the CutPool $\mathcal{C}_k$. MILP solvers do not directly add all cuts generated by the separators to the MILP, as adding a large number of cuts increases the MILP size and slows down the solver. Instead, a cutpool $\mathcal{C}_k$ is used as an intermediate buffer to hold a diverse set of cuts generated by a variety of separators. The cutting plane selector can then compare the cuts in the cutpool and select the most effective ones for the current stage of the MILP solve. Thus, well-designed separator configurations can not only expedite cutting plane generation by deactivating time consuming separators, but can also yield a superior quality cutpool $\mathcal{C}_k$ that may in turn enhance the performance of cutting plane selection $\mathcal{P}_k$ that leads to a further reduction the MILP solve time.

Cutpool has an additional advantage of storing previously generated cuts for future separation rounds and branch-and-cut tree nodes, thereby saving time by reducing the number of calls to expensive separators Achterberg [2007]. As a consequence, configuring separators at a separation round can have both an immediate and long-term impact on the branch-and-bound process due to the presence of the cutpool.

A.2 Configuration Space Restriction: Proofs and Discussions

A.2.1 Preliminary definitions

Table 5: Definition table for key terms used in the paper, including the true and empirical performance of instance-aware predictor and instance-agnostic configuration, and the optimal and empirical risk minimizing (ERM) predictor and configuration on both the original space and the restricted subspace. The relative time improvement δ is defined in Eq. (1) of the main paper. We consider a single configuration update per MILP instance, as in Sec 5.1 of the main paper. The unrestricted space is denoted as $O = \{0,1\}^M$.

		Instance-aware predictor	**Instance-agnostic configuration**
True $\mathcal{X}$	Perf. Measure	$\Delta(f) = \mathop{\mathbb{E}}_{x\in\mathcal{X}}[\delta(f(x),x)]$	$\bar{\delta}(s) = \mathop{\mathbb{E}}_{x\in\mathcal{X}}[\delta(s,x)]$
	Optimal Action (unobserved)	function $f_O^* \to \{0,1\}^M$ s.t. $f_O^*(x) = \arg\max_{s\in\{0,1\}^M} \delta(s,x)\ \forall x \in \mathcal{X}$	config. $s_O^* \in \{0,1\}^M$ s.t. $s_O^* = \arg\max_{s\in\{0,1\}^M} \bar{\delta}(s)$
	Optimal *Subspace-restricted* Action (unobserved)	function $f_A^* \to A \subseteq \{0,1\}^M$ s.t. $f_A^*(x) = \arg\max_{s\in A} \delta(s,x)\ \forall x \in \mathcal{X}$	config. $s_A^* \in A \subseteq \{0,1\}^M$ s.t. $s_A^* = \arg\max_{s\in A} \bar{\delta}(s)$
Empirical $\mathcal{K}$	Perf. Measure	$\hat{\Delta}(f) = \frac{1}{K}\sum_{i=1}^{K} \delta(f(x_i),x_i)$	$\hat{\bar{\delta}}(s) = \frac{1}{K}\sum_{i=1}^{K} \delta(s,x_i)$
	ERM Action (observed)	predictor $\tilde{f}_O^{ERM} : \mathcal{X} \to \{0,1\}^M$ s.t. $\tilde{f}_O^{ERM}(x) = \arg\max_{s\in\{0,1\}^M} \delta(s,x)\ \forall x \in \mathcal{K}$.	config. $\tilde{s}_O^{ERM} \in \{0,1\}^M$ s.t. $\tilde{s}_O^{ERM} = \arg\max_{s\in\{0,1\}^M} \hat{\bar{\delta}}(s)$
	ERM *Subspace-restricted* Action (observed)	predictor $\tilde{f}_A^{ERM} : \mathcal{X} \to A \subseteq \{0,1\}^M$ s.t. $\tilde{f}_A^{ERM}(x) = \arg\max_{s\in A} \delta(s,x)\ \forall x \in \mathcal{K}$.	config. $\tilde{s}_A^{ERM} \in A \subseteq \{0,1\}^M$ s.t. $\tilde{s}_A^{ERM} = \arg\max_{s\in A} \hat{\bar{\delta}}(s)$

True performance. Let $\mathcal{X}$ be a class of MILP instances. Let $f : \mathcal{X} \to \{0,1\}^M$ be a configuration function. The true performance of f is defined as $\Delta(f) = \mathop{\mathbb{E}}_{x\in\mathcal{X}}[\delta(f(x),x)]$.

The optimal configuration function $f_O^* \to \{0,1\}^M$ is $f_O^*(x) = \arg\max_{s\in O} \delta(s,x)\ \forall x \in \mathcal{X}$, and the optimal subspace-restricted configuration function $f_A^* \to A \subseteq \{0,1\}^M$ is $f_A^*(x) = \arg\max_{s\in A} \delta(s,x)\ \forall x \in \mathcal{X}$.

During training, we do not have access to the optimal configuration function nor the time improvement for all configurations and MILP instances in $\mathcal{X}$. Instead, we are given a set of training instances $\mathcal{K}$, from which we can collect the time improvements for different configurations by calling the MILP solver, to learn a configuration predictor. We define the predictor's empirical performance on the training instances as follows.

Empirical performance. Let $f : \mathcal{X} \to \{0,1\}^M$ be a configuration predictor. Let $\mathcal{K} = \{x_1, ..., x_K\}$ be a set of training MILP instances. The empirical performance of f on $\mathcal{K}$ is $\hat{\Delta}(f) = \frac{1}{K}\sum_{i=1}^{K} \delta(f(x_i),x_i)$.

Given a subspace $A \subseteq \{0,1\}^M$, an empirical risk minimization (ERM) configuration predictor $\tilde{f}_A^{ERM} : \mathcal{X} \to A$ selects the best configuration within A for each instance in the training set $\mathcal{K}$, i.e. $\tilde{f}_A^{ERM}(x) = \arg\max_{s\in A} \delta(s,x)\ \forall x \in \mathcal{K}$. That is, $\tilde{f}_A^{ERM}(x)$ coincides with f_A^* on $\mathcal{K}$.

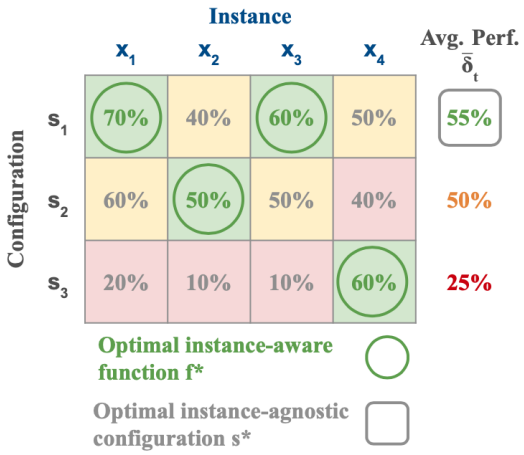


Figure 3: An illustration of the difference between the **optimal instance-aware function** f^*, which maps each MILP instance to a (possibly different) configuration that maximizes the time improvement, and the **optimal instance-agnostic configuration** s^*, which is a single configuration that achieves the highest average time improvement across the instances.

Instance-agnostic performance. For each configuration $s \in \{0,1\}^M$, We further denote the true instance-agnostic performance of applying the same s to all MILP instances as $\bar{\delta}(s) = \mathop{\mathbb{E}}_{x \in \mathcal{X}} [\delta(s,x)]$, and the corresponding empirical instance-agnostic performance as $\hat{\bar{\delta}}_t(s) = \frac{1}{K} \sum_{i=1}^{K} \delta(s, x_i)$.

The optimal and ERM instance-agnostic configuration is defined as $s^*_O = \arg\max_{s \in O} \bar{\delta}_t(s)$ and $\tilde{s}^{ERM}_O = \arg\max_{s \in O} \hat{\bar{\delta}}_t(s)$, and the optimal and ERM instance-agnostic configuration on a restricted subspace $A \subseteq \{0,1\}^M$ is similarly defined as $s^*_A = \arg\max_{s \in A} \bar{\delta}_t(s)$ and $\tilde{s}^{ERM}_A = \arg\max_{s \in A} \hat{\bar{\delta}}_t(s)$. Table 5 provides a list of all related concepts, and Fig. 3 illustrates the difference between the optimal instance-aware function and the optimal instance-agnostic configuration.

A.2.2 Proof of Proposition 1

Proposition 1. Assume that a configuration predictor $\tilde{f}_A$, when evaluated on the entire distribution $\mathcal{X}$, achieves perfect generalization (i.e., zero generalization gap) with probability $1-\alpha$. With probability α, the predictor makes mistakes and outputs a configuration $s \in A$ uniformly at random. Then, the trainset performance v.s. generalization decomposition can be written as

$$\Delta(\tilde{f}_A) = (1-\alpha)\hat{\Delta}(\tilde{f}_A) + \alpha \frac{1}{|A|} \sum_{s \in A} \bar{\delta}(s) \tag{7}$$

Proof. By definition, we have $\Delta(\tilde{f}_A) = \mathop{\mathbb{E}}_{x \in \mathcal{X}} [\delta(f(x), x)]$. From the assumption, we have

$$\Delta(\tilde{f}_A) = \begin{cases} \hat{\Delta}(\tilde{f}_A) & \text{with probability } \alpha \\ \frac{1}{A} \sum\limits_{s \in A} \mathop{\mathbb{E}}\limits_{x \in \mathcal{X}} [\delta(s, x)] = \frac{1}{A} \sum\limits_{s \in A} \bar{\delta}(s) & \text{with probability } 1-\alpha \end{cases} \tag{8}$$

Hence, from Eq. (2) of the main paper, we get

$$\begin{aligned} \Delta(\tilde{f}_A) &= \hat{\Delta}(\tilde{f}_A) - \left(\hat{\Delta}(\tilde{f}_A) - \Delta(\tilde{f}_A) \right) \\ &= \hat{\Delta}(\tilde{f}_A) - (1-\alpha) \cdot 0 - \alpha \cdot \left(\hat{\Delta}(\tilde{f}_A) - \frac{1}{A} \sum_{s \in A} \bar{\delta}(s) \right) \\ &= (1-\alpha)\hat{\Delta}(\tilde{f}_A) + \alpha \frac{1}{A} \sum_{s \in A} \bar{\delta}(s) \quad \blacksquare \end{aligned} \tag{9}$$

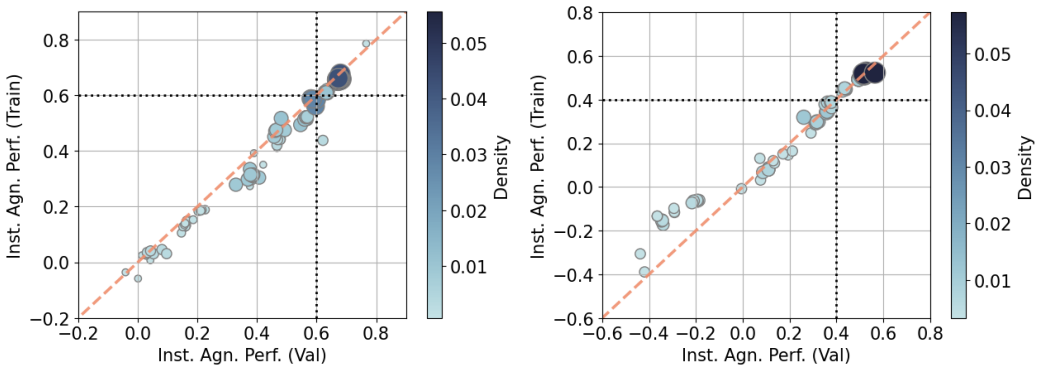
Assumption Discussion (Generalization error). The second average instance-agnostic performance term is a result of the assumption that the predictor selects a configuration randomly when it makes a mistake. In practice, the predictor's performance could be worse. For example, the predictor may select the configuration with the poorest instance-agnostic performance. In such a scenario, our algorithm's filtering strategy (See Alg. 1) that excludes configurations with an average performance below a threshold $\hat{\bar{\delta}}(s) \leq b$ remains highly beneficial: with this strategy, we can ensure that the performance of all selected configurations, including the worst one, is above the threshold value of b. Moreover, when deciding the size of the subspace A, we can track the performance of the worst selected configuration in addition to the average performance across all selected configurations to account for situations where the predictor's mistakes lead to worst-case performance.

Assumption Discussions (Empirical instance-agnostic perf.). In Eq. (3) of the main paper, we approximate the true instance-agnostic performance of each configuration $\bar{\delta}(s)$ by the empirical counterpart $\hat{\bar{\delta}}(s)$, under the assumption that different configuration s have similar generalization behavior when we apply each configuration to all instances. We test the generalization of different configurations by sampling a hold-out validation set $\mathcal{V}$, and compare the performance of $\hat{\bar{\delta}}(s)$ evaluated on the training set $\mathcal{K}_{small}$ (denoted as $\hat{\bar{\delta}}^{\mathcal{K}_{small}}(s)$) and on the hold-out set $\mathcal{V}$ (denoted as $\hat{\bar{\delta}}^{\mathcal{V}}(s)$).

The scatter plots in Fig. 4 show the instance-agnostic performances for Maximum Cut and Independent Set on a training set $\mathcal{K}_{small}$ of 100 instances and a hold-out set $\mathcal{V}$ of 100 instances. The plotted configurations are selected from the initial configuration space S (see Alg. 1) by picking from each bin in the histogram of the set $\{\hat{\bar{\delta}}^{\mathcal{K}_{small}}(s), s \in S\}$ to ensure a diverse range of instance-agnostic performances among the chosen configurations. The darkness and size of each circle (configuration) in the plot are proportional to the total number of configurations in the corresponding bin, divided by the number of samples selected from that bin.

The strong linear trend $y = x$ observed in each scatter plot, along with the perfect alignment of configurations excluded by the filtering strategy in Alg. 1 (represented by circles in the bottom left corner split by the black dotted lines) validate our approximation of the true instance-agnostic performance with the empirical counterpart.

Notably, while we observe a strong linear trend in all the MILP classes we consider, there may still be challenging MILP classes where this linear trend does not hold. In other words, there may exist



(a) Maximum Cut

(b) Independent Set

Figure 4: Instance-agnostic performance of configuration samples on the training set $\hat{\bar{\delta}}^{\mathcal{K}_{small}}(s)$ and hold-out set $\hat{\bar{\delta}}^{\mathcal{V}}(s)$ for Maximum Cut and Independent Set. The dashed orange line indicates the line of equality ($y = x$). The darkness and size of each circle (configuration) in the plot are proportional to the total number of configurations in the corresponding bin, divided by the number of samples selected from that bin. The horizontal and vertical black dotted lines indicate our choice of filtering threshold b in Alg. 1. Respectively, 38% and 58% of all configurations in the initial configuration space S surpass the filtering threshold and are considered as candidates for the final subspace A.

certain MILP classes where configurations that perform well on a training set may fail to generalize to the unseen test set. In such cases, we can modify our Alg. 1 to incorporate an additional holdout set $\mathcal{V}$, and filter configurations based on the performance on the hold out set $\hat{\bar{\delta}}^{\mathcal{V}}(s)$ instead of on the training set $\hat{\bar{\delta}}^{\mathcal{K}_{small}}(s)$ (See Line 16 in Alg. 1: $\hat{\bar{\delta}}(s) = \hat{\bar{\delta}}^{\mathcal{K}_{small}}(s)$ in our default algorithm, but can be replaced by $\hat{\bar{\delta}}^{\mathcal{V}}(s)$). In this way, we can more accurately capture the generalization behavior of each configuration, although this modification would increase the number of MILP solver calls required to collect the validation performances, which our reduction to solely monitor training performances $\hat{\bar{\delta}}^{\mathcal{K}_{small}}(s)$ avoids.

A.2.3 Proof of Proposition 2

Proposition 2. (Submodularity of $\hat{\Delta}(\tilde{\mathbf{f}}_{\mathbf{A}}^{\mathbf{ERM}})$ and the greedy approximation algorithm). The empirical performance of the ERM predictor $\hat{\Delta}(\tilde{f}_A^{ERM})$ is a monotone increasing and submodular function in A, and a greedy strategy where we include the configuration that achieves the greatest marginal improvement $\arg\max_{s\in\{0,1\}^M\setminus A} \hat{\Delta}(\tilde{f}_{A\cup\{s\}}^{ERM}) - \hat{\Delta}(\tilde{f}_A^{ERM})$ at each iteration is a $(1-1/e)$-approximation algorithm for constructing the subspace A that optimizes $\hat{\Delta}(\tilde{f}_A^{ERM})$.

Proof of monotonicity. By definition, we have $\hat{\Delta}(\tilde{f}_A^{ERM}) = \frac{1}{K}\sum_{i=1}^{K} \delta(\tilde{f}_A^{ERM}(x_i), x_i)$ on a restricted subspace A. According to the ERM rule, for each instance $x_i \in \mathcal{M}$ we have $\delta(\tilde{f}_A^{ERM}(x_i), x_i) = \max_{s\in A} \delta(s, x)$. We note that $\delta(\tilde{f}_A^{ERM}(x_i), x_i)$ is a monotone increasing function in A for each x_i, since if $B \subseteq C$, then $\delta(\tilde{f}_B^{ERM}(x_i), x_i) \leq \delta(\tilde{f}_C^{ERM}(x_i), x_i)$ due to the monotonicity of the max operator on the set. Averaging across all instances, $\hat{\Delta}(\tilde{f}_A^{ERM})$ is hence a monotone increasing function in A.

Intuition for submodularity. Adding a configuration s to a set of configurations A improves $\delta(\tilde{f}_{A\cup\{s\}}^{ERM}(x_i), x_i)$ from $\delta(\tilde{f}_A^{ERM}(x_i), x_i)$ (positive marginal improvement) if s performs better than all configurations in A on the MILP instance x_i. Intuitively speaking, with a larger subspace A, it is less likely for s to improve the performance, because there are more competing choices in A that make it more difficult for s to perform the best. Hence, we get a smaller marginal improvement when adding a configuration s to a larger set of configurations for each instance, therefore making the empirical performance $\hat{\mathbf{\Delta}}(\tilde{\mathbf{f}}_{\mathbf{A}}^{\mathbf{ERM}})$ averaged across all instances submodular in A. We provide a rigorous proof of submodularity below.

Proof of submodularity. Let $B \subseteq C \subseteq O = \{0,1\}^M$ and $s \in O \setminus C$. We want to show

$$\hat{\Delta}(\tilde{f}_{B\cup\{s\}}^{ERM}) - \hat{\Delta}(\tilde{f}_B^{ERM}) \geq \hat{\Delta}(\tilde{f}_{C\cup\{s\}}^{ERM}) - \hat{\Delta}(\tilde{f}_C^{ERM}) \tag{10}$$

We have

$$\hat{\Delta}(\tilde{f}_{B\cup\{s\}}^{ERM}) - \hat{\Delta}(\tilde{f}_B^{ERM}) = \frac{1}{K}\sum_{i=1}^{K}[\delta(\tilde{f}_{B\cup\{s\}}^{ERM}(x_i), x_i) - \delta(\tilde{f}_B^{ERM}(x_i), x_i)] \tag{11}$$

We can split the set $\mathcal{M} = \{x_1, ..., x_N\}$ into two nonoverlapping subsets $\mathcal{M}_0$ and $\mathcal{M}_1$ where

- $\forall x \in \mathcal{M}_0$, some configuration $s' \in B$ performs at least as good as s. That is, $\tilde{f}_{B\cup\{s\}}^{ERM}(x) = \arg\max_{s'\in B\cup\{s\}} \delta(s', x) \in B$, and hence

$$\delta(\tilde{f}_{B\cup\{s\}}^{ERM}(x), x) = \delta(\tilde{f}_B^{ERM}(x), x) \geq \delta(s, x) \tag{12}$$

- $\forall x \in \mathcal{M}_1$, the configuration s performs better than all configurations in B. That is, $\tilde{f}_{B\cup\{s\}}^{ERM}(x) = \arg\max_{s'\in B\cup\{s\}} \delta(s', x) = s$, and hence

$$\delta(\tilde{f}_{B\cup\{s\}}^{ERM}(x), x) = \delta(s, x) > \delta(\tilde{f}_B^{ERM}(x), x) \tag{13}$$

Then, from Eq. (11), we have

$$\hat{\Delta}(\tilde{f}_{B\cup\{s\}}^{ERM}) - \hat{\Delta}(\tilde{f}_B^{ERM}) = \frac{1}{K}\sum_{x\in\mathcal{M}_1}[\delta(s, x) - \delta(\tilde{f}_B^{ERM}(x), x)] \tag{14}$$

Now consider

$$\hat{\Delta}(\tilde{f}_{C\cup\{s\}}^{ERM}) - \hat{\Delta}(\tilde{f}_C^{ERM}) = \frac{1}{K}\sum_{i=1}^{K}[\delta(\tilde{f}_{C\cup\{s\}}^{ERM}(x_i), x_i) - \delta(\tilde{f}_C^{ERM}(x_i), x_i)] \tag{15}$$

We have the following nonoverlapping cases

- $\forall x \in \mathcal{M}_0$, due to monotonicity of $\hat{\Delta}(\tilde{f}_A^{ERM})$ in A and the fact that $B \subseteq C$, we have, extending from Eq. (12),

$$\delta(\tilde{f}_C^{ERM}(x), x) \geq \delta(\tilde{f}_B^{ERM}(x), x) \geq \delta(s, x) \tag{16}$$

  and hence some configuration $s' \in C$ performs at least as good as s.
- $\forall x \in \mathcal{M}_1$, we further split into two nonoverlapping cases:
  (i) s performs better than all configurations in C. That is, $\tilde{f}_{C\cup\{s\}}(x) = s$ and

$$\delta(\tilde{f}_{C\cup\{s\}}^{ERM}(x), x) = \delta(s, x) > \delta(\tilde{f}_C^{ERM}(x), x) \geq \delta(\tilde{f}_B^{ERM}(x), x) \tag{17}$$

  (ii) some configuration in $C \setminus B$ performs better than s. That is, $\tilde{f}_{C\cup\{s\}}^{ERM}(x) \in C \setminus B$ and

$$\delta(\tilde{f}_{C\cup\{s\}}^{ERM}(x), x) = \delta(\tilde{f}_C^{ERM}(x), x) \geq \delta(s, x) \geq \delta(\tilde{f}_B^{ERM}(x), x) \tag{18}$$

  where the last inequality is from Eq. (13).
  We let $\mathcal{M}_1 = \mathcal{M}_{11} \cup \mathcal{M}_{12}$ where $\mathcal{M}_{11}$ and $\mathcal{M}_{12}$ corresponds to (i) and (ii).

We thus have

$$\begin{aligned}
\hat{\Delta}(\tilde{f}_{C\cup\{s\}}^{ERM}) - \hat{\Delta}(\tilde{f}_C^{ERM}) &= \frac{1}{K} \sum_{x \in \mathcal{M}_{11} \subseteq \mathcal{M}_1} [\delta(s, x) - \delta(\tilde{f}_C^{ERM}(x), x)] \\
&\leq \frac{1}{K} \sum_{x \in \mathcal{M}_{11} \subseteq \mathcal{M}_1} [\delta(s, x) - \delta(\tilde{f}_B^{ERM}(x), x)] \\
&\leq \frac{1}{K} \sum_{x \in \mathcal{M}_1} [\delta(s, x) - \delta(\tilde{f}_B^{ERM}(x), x)] \\
&= \hat{\Delta}(\tilde{f}_{B\cup\{s\}}^{ERM}) - \hat{\Delta}(\tilde{f}_B^{ERM})
\end{aligned} \tag{19}$$

where the second inequality is due to the monotonicity of $\delta(\tilde{f}_A^{ERM}(x), x)$ in A for all x (see Eq. (17)), the third inequality is due to nonnegativity of the additional terms in $\mathcal{M}_1 \setminus \mathcal{M}_{11}$ (see Eq. (18)), and the last equality is from Eq. (14).

The greedy approximation algorithm. Due to the monotone submodularity of the empirical performance of the ERM predictor $\hat{\Delta}(\tilde{f}_A^{ERM})$, the greedy strategy where we include the configuration that achieves the greatest marginal improvement $\underset{s \in \{0,1\}^M \setminus A}{\arg\max}\ \hat{\Delta}(\tilde{f}_{A\cup\{s\}}^{ERM}) - \hat{\Delta}(\tilde{f}_A^{ERM})$ at each iteration is a $(1 - 1/e)$-approximation algorithm for constructing the subspace A, as proven in previous work Nemhauser et al. [1978]. ■

A.2.4 ERM assumption discussion and relaxation to predictors with training error

Assuming that the predictor $\tilde{f}_A$ is the ERM predictor $\tilde{f}_A^{ERM}$ that performs optimally on the training set, we can construct a subspace A prior to learning the actual predictor $\tilde{f}_A$ by replacing the learned predictor's prediction with the ERM selection rule. This assumption is reasonable because during training, we optimize the predictor with the empirical performance $\hat{\Delta}(\tilde{f}_A)$ as the objective, which aligns with the objective of the ERM predictor (where ERM obtains the optimal solution). To account for potential training errors, we can relax the ERM assumption with the following lemma, from which we achieve a trade-off similar to Eq. (3) of the main paper, balancing the ERM performance and a generalization term with a higher weight on the latter (see Eq. (24)). Our Alg. 1 hence still applies.

Lemma 3. Assume that a predictor $\tilde{f}_A$, when trained on $\mathcal{K}$, achieves optimal training performance (i.e., ERM $\tilde{f}_A^{ERM}$) with probability $1 - \beta$. With probability β, the predictor makes mistakes and outputs a configuration $s \in A$ uniformly at random. Then, combining with the assumption in Proposition 1 (See Appendix A.2.2), the trainset performance v.s. generalization decomposition can be written as

$$\Delta(\tilde{f}_A) = (1 - \alpha)(1 - \beta)\hat{\Delta}(\tilde{f}_A^{ERM}) + (1 - \alpha)\beta \frac{1}{|A|} \sum_{s \in A} \hat{\bar{\delta}}(s) + \alpha \frac{1}{|A|} \sum_{s \in A} \bar{\delta}(s) \tag{20}$$

Proof. By definition, $\hat{\Delta}(\tilde{f}_A) = \frac{1}{K}\sum_{i=1}^{K}\delta(f(x_i), x_i)$. Following the similar proof structure as Proposition 1, we have

$$\hat{\Delta}(\tilde{f}_A) = \begin{cases} \hat{\Delta}(\tilde{f}_A^{ERM}) & \text{with probability } \beta \\ \frac{1}{A}\sum_{s\in A}\frac{1}{K}\sum_{i=1}^{K}\delta(s, x_i) = \frac{1}{A}\sum_{s\in A}\hat{\bar{\delta}}(s) & \text{with probability } 1-\beta \end{cases} \tag{21}$$

Hence, we get

$$\hat{\Delta}(\tilde{f}_A) = (1-\beta)\hat{\Delta}(\tilde{f}_A^{ERM}) + \beta\frac{1}{A}\sum_{s\in A}\hat{\bar{\delta}}(s) \tag{22}$$

Before we further proceed in the proof, we discuss an additional trade-off when constructing the subspace A based on the empirical performance on the training set $\hat{\Delta}(\tilde{f}_A)$. While adding more configurations to A may improve the empirical performance of the ERM predictor $\tilde{f}_A^{ERM}$, some of these configurations may have low instance-agnostic performance and only perform well on a small subset of the training instances. Incorporating such configurations into A may lead to the selection of poor configurations when training error occurs, resulting in a decrease in the performance on the training set $\hat{\Delta}(\tilde{f}_A)$. Hence, to construct a subspace A that can result in the high empirical performance of the imperfect predictor, we also need to balance the size and diversity of A (measured by the empirical performance of the ERM predictor on A, the first term), and the average configuration quality in A (measured by the average instance-agnostic empirical performance, the second term).

Now, combining with the proof of Proposition 1, we hence have

$$\begin{aligned} \Delta(\tilde{f}_A) &= \hat{\Delta}(\tilde{f}_A) - \left(\hat{\Delta}(\tilde{f}_A) - \Delta(\tilde{f}_A)\right) \\ &= (1-\alpha)\hat{\Delta}(\tilde{f}_A) + \alpha\frac{1}{A}\sum_{s\in A}\bar{\delta}(s) \\ &= (1-\alpha)\left((1-\beta)\hat{\Delta}(\tilde{f}_A^{ERM}) + \beta\frac{1}{A}\sum_{s\in A}\hat{\bar{\delta}}(s)\right) + \alpha\frac{1}{A}\sum_{s\in A}\bar{\delta}(s) \\ &= (1-\alpha)(1-\beta)\hat{\Delta}(\tilde{f}_A^{ERM}) + (1-\alpha)\beta\frac{1}{A}\sum_{s\in A}\hat{\bar{\delta}}(s) + \alpha\frac{1}{A}\sum_{s\in A}\bar{\delta}(s) \quad \blacksquare \end{aligned} \tag{23}$$

Then, following Eq. (3) of the main paper, we replace $\bar{\delta}(s)$ (unobservable) by $\hat{\bar{\delta}}(s)$ and arrive at the following objective to select the subspace A:

$$\begin{aligned} \Delta(\tilde{f}_A) &= (1-\alpha)(1-\beta)\hat{\Delta}(\tilde{f}_A^{ERM}) + (\alpha+\beta-\alpha\beta)\frac{1}{|A|}\sum_{s\in A}\hat{\bar{\delta}}(s) \\ &= (1-\gamma)\hat{\Delta}(\tilde{f}_A^{ERM}) + \gamma\frac{1}{|A|}\sum_{s\in A}\hat{\bar{\delta}}(s) \quad \text{, where } \gamma = \alpha+\beta-\alpha\beta. \end{aligned} \tag{24}$$

Comparing the above with Eq. (3) where we assume the predictor performs ERM perfectly with no training error, we arrive at the same trade-off between the empirical performance of the ERM predictor $\tilde{f}_A^{ERM}$ in the subspace A (which is monotone submodular in A), and the average instance-agnostic performance of all configurations $s \in A$, with a lower weight on the first term and a higher weight on the second term due to training error ($\gamma \geq \alpha$). Hence, our algorithm that couples a greedy strategy with the filtering criterion naturally applies to this relaxed scenario. The greedy strategy can still select configuration based on the ERM predictor, as the training error from the new predictor is absorbed in the second term. Due to the increase weight on the second term, we would increase the threshold b, which we design as a hyperparameter in Alg. 1, to more aggressively filter out configuration s with a low instance-agnostic performance given by $\hat{\bar{\delta}}(s) \leq b$. We leave it as future work to analyze more complicated predictors $\tilde{f}'_A$ that incorporate other smoothness assumptions and to adapt the construction algorithm based on the performance of such predictors.

A.3 Configuration Space Restriction: Algorithm

A.3.1 Algorithm

The algorithm for our data-driven configuration space restriction in Sec. 5.1 is presented in Alg. 1.

Algorithm 1: Configuration_Space_Restriction

Input: MILP training set $\mathcal{K}_{small}$, the unrestricted configuration space $O = \{0,1\}^M$, number of initial configuration samples $|S|$, size of the restricted configuration space $|A|$, instance-agnostic performance threshold b

Output: The restricted configuration space A

1 $S \leftarrow$ large subset of O by sampling $|S|$ configurations from O // See description below
2 // construct the relative time improvement table
3 $T \leftarrow zeros(|S|, |\mathcal{K}_{small}|)$
4 **for** *Configuration* s_i *in* S **do**
5 **for** *Instance* x_j *in* $\mathcal{K}_{small}$ **do**
6 // Solve instance x_j with configuration s_i with the MILP solver
7 $T_{ij} \leftarrow \delta(s_i, x_j)$
8 **end for**
9 **end for**
10 // construct the restricted configuration space
11 $A \leftarrow \{\}$
12 **for** *Choice i = 1:* |A| **do**
13 // greedy based on marginal improvement in instance-aware perf. (see Table 5)
14 $s \leftarrow \arg\max_{s \in S \setminus A} \hat{\Delta}(\tilde{f}^{ERM}_{A \cup \{s\}}, \mathcal{X}) - \hat{\Delta}(\tilde{f}^{ERM}_{A}, \mathcal{X})$
15 // filtering based on instance-agnostic perf. of individual configuration (see Table 5)
16 **if** $\hat{\bar{\delta}}(s) > b$ **then**
17 $A \leftarrow A \cup \{x\}$
18 **else**
19 $S \leftarrow S \setminus \{x\}$
20 **end if**
21 **end for**

On Line 1, we use the following two strategies to sample the large initial configuration subset $S \subseteq O$:

1. **Near Zero:** we include all configurations that activate at most 3 separators, which results in a subset S_1 of size $\sum_{i=0}^{3} \binom{M}{i} = \sum_{i=0}^{3} \binom{17}{i} = 834$.

2. **Near Best Random:** we first sample 500 configurations uniformly at random from O and find the configuration $\tilde{s}'$ in the sample with the highest empirical instance-agnostic performance $\hat{\bar{\delta}}(s)$ on the training set $\mathcal{K}_{small}$ (see Table 5). Then, we include (1) all configurations that have at most 3 separator different from $\tilde{s}'$, resulting in a subset S_{21} of size 834, and (2) all configurations whose set of activated separators is a subset of the activated separators in $\tilde{s}'$, resulting in another subset S_{22} whose size depends on the number of activated separators in $\tilde{s}'$ and ranges from 63 to 1023 for all MILP classes considered in this paper.

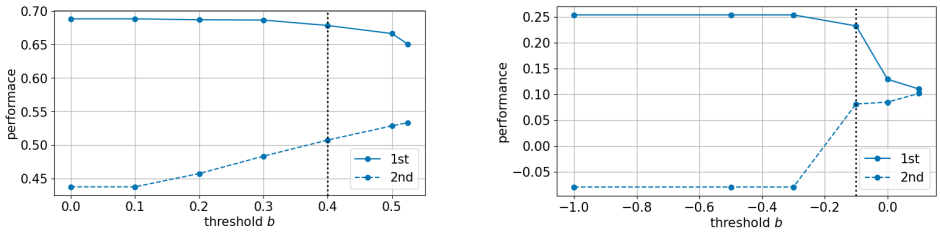
Combining all samples from above, we obtain a large initial configuration subset S with $|S| \approx 2000$.

The **Near Best Random** strategy is designed to bootstrap high quality samples around the high quality configuration $\tilde{s}'$ obtained from random search. The subset S_{21} increases sample diversity by perturbing $\tilde{s}'$ within a Hamming distance of 3, and S_{22} is designed based on the intuition that it may be possible to deactivate more separators from $\tilde{s}'$, as some activated separators in $\tilde{s}'$ may be useful for certain MILP instances but not for the others. The **Near Zero** strategy is designed based on the intuition that it may be beneficial to maintain a small set of activated separators, as it reduces the time to invoke separator algorithms (although at the cost of reducing the quality of the generated cuts).

A.3.2 Algorithm discussions: filtering and subspace size

When employing our filtering strategy, a higher (more aggressive) threshold $\hat{\bar{\delta}}(s) \leq b$ with larger b leads to a higher average empirical instance-agnostic performance (second term $\frac{1}{|A|}\sum_{s\in A}\hat{\delta}(s)$ in Eq. (3) of the main paper), which measures generalizability of the instance-aware predictor, but it also incurs a decrease in the empirical performance for the instance-aware ERM predictor (first term $\hat{\Delta}(\tilde{f}_A)$ in Eq. (3)), which measures the training performance of the instance-aware predictor).

In Figure 5, we plot the behavior of these two terms across different threshold values of b (ignoring the weight of α), using the $\mathcal{K}_{small}$ training set of Independent Set and Load Balancing. The size of the subspace A is fixed at $|A| = 15$ for Independent Set and $|A| = 25$ for Load Balancing, which equals the size of our chosen subspace in L2Sep that is constructed with filtering threshold $b^{ours} = 0.4$ for Independent Set and $b^{ours} = -0.1$ for Load Balancing. The subspaces are constructed by our Alg. 1 that combines a greedy strategy with the filtering criterion.

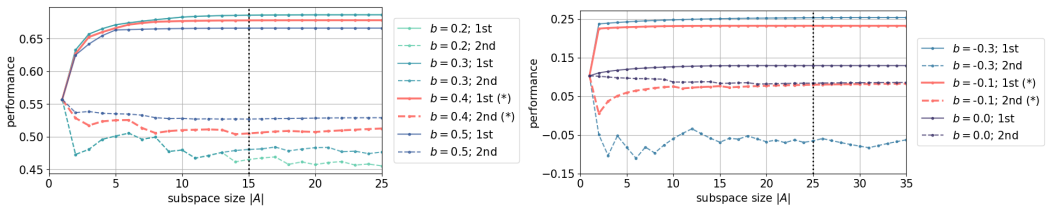


(a) Independent Set $|A| = 15$ (b) Load Balancing $|A| = 25$

Figure 5: Training set performance (1^{st}) and generalization (2^{nd}) terms in Eq. (3) of the main paper for subspaces constructed using Alg. 1 with varying thresholds b. The vertical dotted black lines indicate our chosen threshold.

An effective approach for choosing the threshold, as supported by our theoretical analysis in Sec. 5.1 of the main paper, is to find a value b that yields a substantial improvement in the generalization term (2^{nd}), while simultaneously maintaining a high training set performance term (1^{st}). Our selection of $b^{ours} = 0.4$ for Independent Set and $b^{ours} = -0.1$ for Load Balancing satisfies this criterion.

In Figure 6, we further plot the two terms during the intermediate construction process of Alg. 1 where the subspace A is expanded by adding a configuration at each iteration. The plot shows the behavior of the two terms through the construction process for a set of thresholds b (as specified in the legend), when evaluated on the same $\mathcal{K}_{small}$ training dataset.



(a) Independent Set (b) Load Balancing

Figure 6: Training set performance (1^{st}) and generalization (2^{nd}) terms in Eq. (3) for intermediate subspaces constructed using Alg 1 across a set of thresholds b. The vertical dotted black lines indicate our chosen subspace size. The asterisks (*) in the legend indicate our choice of b.

Once again, our choice of the threshold b allows a notable improvement in the second generalization term (2^{nd}), while maintaining a high level of performance in the first training set performance term (1^{st}). This trend persists throughout each step of our iterative algorithm. We choose the size of the subspace $|A|$ (the termination criterion of our algorithm) when $|A|$ is reasonably small, while both terms stabilize and at values that offer a favorable trade-off between the two terms.

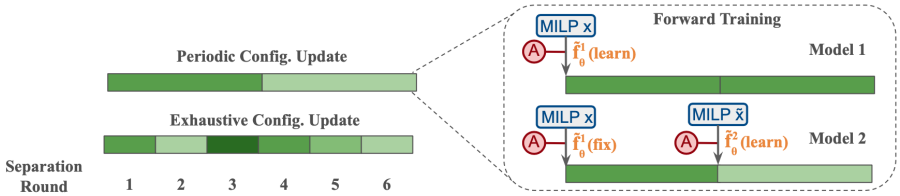


Figure 7: **Forward training.** (Left) The periodic update scheme where the configuration is updated at different separation rounds ($n_1 = 1$ and $n_2 = 4$ in the illustration) and held constant between updates (c.f. exhaustive update at each of the $R = 6$ rounds). (Right) The forward training algorithm that sequentially learn the k networks $\{\tilde{f}^m_{\theta^T}\}_{m=1}^k$, whose output spaces are constrained to the subspace A, for periodic configuration update. The MILP input to $\tilde{f}^2_\theta$ is denoted as $\tilde{x}$ as it includes both the initial MILP x and newly added cuts. Different shades of green represent different selected configurations.

A.4 Configuration Update Restriction

A.4.1 Forward training algorithm

Alg. 2 presents our forward training procedure that trains k predictor networks to perform k configuration updates for each MILP instance. As illustrated in Fig 7, when training the j^{th} network, we freeze the weights of the pre-trained networks $\{\tilde{f}^m_{\theta^T}\}_{m=1}^{j-1}$ and use them to update the configurations at separation rounds $\{n_m\}_{m=1}^{j-1}$. Then, we learn the j^{th} network $\tilde{f}^j_{\theta^0}$ using neural UCB to update the configuration at separation round n_j, and we hold the configuration constant until the solver terminates (at optimality or a fixed gap) to collect the terminal reward. We do not use intermediate rewards as both the optimality gap and solve time vary at an intermediate round, making it difficult to construct an integrated reward to compare different configurations.

Algorithm 2: Forward_Training

Input: MILP training set $\mathcal{K}_{large}$, number of configuration update steps k, configuration predictor networks $\{f^j_{\theta^0}\}_{j=1}^k$, separation rounds $\{n_j\}_{j=1}^k$, training epochs T, number of MILP instances P per epoch, number of samples to collect reward labels per epoch D, UCB scaling factor γ, configuration subspace A

Output: Trained regression networks $\{f^j_{\theta^T}\}_{j=1}^k$

for *Update j = 1: k* **do**
  Freeze the weight of $\{f^m_{\theta^T}\}_{m=1}^{j-1}$
  $f^j_{\theta^T} \leftarrow$ Neural_UCB $(\mathcal{K}_{large}, f^j_{\theta^0}, \{f^m_{\theta^T}\}_{m=1}^{j-1}, \{n_m\}_{m=1}^j, T, P, D, \gamma, A)$
end for

A.4.2 Trade-off discussion for different k's

Let $\pi^{*(R)}$ and $\pi^{*(k)}$ be the optimal configuration policies when we perform R and $k \ll R$ updates, and let $\tilde{\pi}^{(R)}$ and $\tilde{\pi}^{(k)}$ be the corresponding learned policies. Due to cascading errors over the long horizon, the learning task for $\tilde{\pi}^{(R)}$ is more challenging than for $\tilde{\pi}^{(k)}$; on the other hand, the optimal policy $\pi^{*(k)}$ performs worse than $\pi^{*(R)}$ due to the action space restriction. Hence, the frequency k trades off approximation (for $\pi^{*(k)} \approx \pi^{*(R)}$) and estimation (for $\tilde{\pi}^{(k)} \approx \pi^{*(k)}$), with more frequent updates (larger k) improves the approximation error while less frequent updates (smaller k) improves the estimation error.

Recent theoretical work by Metelli et al. [2020] investigates the impact of action persistence, namely repeating an action for a fixed number of decision steps, for infinite horizon discounted MDPs. They provide a theoretical bound on the approximation error in terms of the differences in the optimal Q-value with and without action persistence (which corresponds to the Q-value of $\pi^{*(R)}$ and $\pi^{*(k)}$ in our setting). The resulting bound is a function of the discount factor, action hold length, and the discrepancy between the transition kernel with and without action persistence. The approximation error is agnostic to the specific learning algorithm, and hence the analysis can be adapted to our

setting by extending it to the finite horizon MPD scenario. They further use fitted Q-iteration to learn the policies and establish a theoretical bound on the estimation error in terms of the differences in the Q-value of the learned policies with and without action persistence. In contrast, our learned policies $\tilde{\pi}^{(R)}$ and $\tilde{\pi}^{(k)}$ are trained via the forward training algorithm. While it is not the focus of our paper, we note a possible future research to extend their theoretical analysis of the estimation error to the forward training algorithm, and compare the theoretical bounds on approximation and estimation to analyze the trade-off associated with the configuration update frequency k.

A.5 Neural UCB Algorithm

A.5.1 Training algorithm

The Neural UCB algorithm Zhou et al. [2020] that we employ to train each configuration predictor network is presented in Alg. 3.

Algorithm 3: Neural_UCB

Input: MILP training set $\mathcal{K}_{large}$, predictor network at the current separation round $\tilde{f}^j_{\theta^0}$, trained predictor networks at previous separation rounds $\{\tilde{f}^m_{\theta^T}\}_{m=1}^{j-1}$, current separation round n_j, previous separation rounds $\{n_m\}_{m=1}^{j-1}$, training epochs T, number of MILP instances P per epoch, number of samples to collect reward labels per epoch D, UCB scaling factor γ, UCB regularization parameter λ, configuration subspace A

Output: Trained predictor network $\tilde{f}^j_{\theta^T}$

Initialize $B^0 \leftarrow$ an empty training data buffer
Initialize $Z_0 \leftarrow \lambda I_{|\theta|\times|\theta|}$
for *Epoch t = 1: T* **do**
  Initialize $B^t \leftarrow B^{t-1}$
  for *P iterations* **do**
    Sample instance x from $\mathcal{K}_{large}$
    // sampling D configurations with UCB to balance exploration and exploitation
    for *Configuration s in A* **do**
      Compute $U_{t,s} \leftarrow \tilde{f}^j_{\theta^{t-1}}(x, s_i) + \gamma\sqrt{\nabla_\theta \tilde{f}^j_{\theta^{t-1}}(x,s)^\intercal Z_{t-1}^{-1} \nabla_\theta \tilde{f}^j_{\theta^{t-1}}(x,s)}$
    end for
    Let $S \leftarrow D$ samples without replacement $\sim softmax_{s\in A}(U_{t,s})$
    Compute $Z_t \leftarrow Z_{t-1} + \sum_{s \text{ in } S} \nabla_\theta \tilde{f}^j_{\theta^{t-1}}(x,s) \nabla_\theta \tilde{f}^j_{\theta^{t-1}}(x,s)^\intercal$
    // Collect the reward label for each configuration
    for *Sample i = 1: D* **do**
      Regression Label $r_i \leftarrow$ Run the MILP solver for the instance x: use $\{\tilde{f}^m_{\theta^T}\}_{m=1}^{j-1}$ to update configurations at separation rounds $\{n_m\}_{m=1}^{j-1}$, and update configuration to s_i at separation round n_j
    end for
    $B^t \leftarrow B^t \cup \{x, s_i, r_i\}_{i=1}^{H+1}$
  end for
  $\tilde{f}^j_{\theta^t} \leftarrow$ Train $\tilde{f}^j_{\theta^{t-1}}$ with the updated buffer B^t
end for

A.5.2 Input features

Paulus et al. [2022] design a comprehensive set of input features for variable and constraint nodes (extended from Gasse et al. [2019] for their cut selection task), resulting in $\mathbf{V} \in \mathbb{R}^{n\times 17}$ and $\mathbf{C} \in \mathbb{R}^{m\times 34}$, where m and n are the number of constraints and variables in the MILP instance x. We note that the constraint nodes include the initial constraints from the MILP x as well as the newly added cuts. We adopt their input features and provide a detailed description of the features in Table 6 for completeness of the paper. Meanwhile, we set the separator nodes (unique to our task) as $\mathbf{S} \in \mathbb{R}^{M\times 1}$, where M is the number of separators in the MILP solver, and each separator node $\mathbf{S}_k$

has a single dimensional binary feature indicating whether the separator is activated (1) or deactivated (0).

For edge weights, we similarly follow Paulus et al. [2022] and Gasse et al. [2019] to connect each variable-constraint node pair if the variable appears in a constraint, and set the edge weight to be the corresponding nonzero coefficient. Meanwhile, we connect all separator-variable and separator-constraint node pairs with a weight of 1, which results in a complete pairwise message passing between each separator-variable and separator-constraint pair in the Graph Convolution Network Kipf and Welling [2017]. As we lack reliable prior knowledge of the weight for the separator-variable and separator-constraint pair, we do not provide initial weight information and instead directly use the graph convolution mechanism to automatically learn the similarity between each pair.

Table 6: Description of input features for variable and constraint nodes Paulus et al. [2022].

Node Type	Feature	Description
Vars	norm coef	Objective coefficient, normalized by objective norm
	type	Type (binary, integer, impl. integer, continuous) one-hot
	has lb	Lower bound indicator
	has ub	Upper bound indicator
	norm redcost	Reduced cost, normalized by objective norm
	solval	Solution value
	solfrac	Solution value fractionality
	sol_is_at_lb	Solution value equals lower bound
	sol_is_at_ub	Solution value equals upper bound
	norm_age	LP age, normalized by total number of solved LPs
	basestat	Simplex basis status (lower, basic, upper, zero) one-hot
Cons, Added Cuts	is_cut	Indicator to differentiate cut vs. constraint
	type	Separator type, one-hot
	rank	Rank of a row
	norm_nnzrs	Fraction of nonzero entries
	bias	Unshifted side normalized by row norm
	row_is_at_lhs	Row value equals left hand side
	row_is_at_rhs	Row value equals right hand side
	dualsol	Dual LP solution of a row, normalized by row and objective norm
	basestat	Basis status of a row in the LP solution, one-hot
	norm_age	Age of row, normalized by total number of solved LPs
	norm_nlp_creation	LPs since the row has been created, normalized
	norm_intcols	Fraction of integral columns in the row
	is_integral	Activity of the row is always integral in a feasible solution
	is_removable	Row is removable from the LP
	is_in_lp	Row is member of current LP
	violation	Violation score of a row
	rel_violation	Relative violation score of a row
	obj_par	Objective parallelism score of a row
	exp_improv	Expected improvement score of a row
	supp_score	Support score of a row
	int_support	Integral support score of a row
	scip_score	SCIP score of a row for cut selection

A.6 Experiment Setups

A.6.1 Proposed method details

Our main result tables (Table 1 and Table 3 of the main paper) present our complete method and two sub-components. We provide details on the implementation of these three methods.

(1) Ours (L2Sep): for each MILP class, we first run Alg. 1 to obtain a configuration subspace A. Then, we run Alg. 2 that uses forward training to learn $k = 2$ separate instance-aware configuration predictors $\tilde{f}^1_\theta$ and $\tilde{f}^2_\theta$ to perform two configuration updates at separation rounds n_1 and n_2. The output of the predictors is restricted to the subspace A. We train the predictors using the Neural UCB Algorithm 3. A summary of our training pipeline is shown in Alg. 4. At inference time, we have two selection strategies: we select the configuration with either the highest predicted reward $s^*_{x,j} = \arg\max_{s\in A} \tilde{f}^j_{\theta^T}(x, s)$ or the highest UCB score $s^*_{x,j} = \arg\max_{s\in A} ucb(x, s)$ for all predictors j, determined based on validation performance. In our experience, we observe that selecting configurations based solely on the point estimate $\tilde{f}^j_{\theta^T}(x, s)$ alone can sometimes be overly deterministic, leading to a limited range of configurations being chosen for most instances. The UCB score combines the reward point estimate $\tilde{f}^j_{\theta^T}(x, s)$ with an uncertainty estimate (computed in Line 8 of Alg. 3, where the normalizing matrix Z is from the same epoch as the selected model), and it hence increases the diversity of the prediction and improves the performance in most scenarios. Given the selection strategy, for each instance x, we set the configuration to be $s^*_{x,1}$ at separation rounds n_1 and $s^*_{x,2}$ at separation rounds n_2. We hold the configuration fixed as $s^*_{x,1}$ between separation rounds $[n_1, n_2]$ and as $s^*_{x,2}$ from separation round n_2 until the solving process terminates.

Algorithm 4: Learning_To_Separate (L2Sep)

Input: MILP training set $\mathcal{K}_{small}$, the unrestricted configuration space $O = \{0, 1\}^M$, number of initial configuration samples $|S|$, size of the restricted configuration space $|A|$, instance-agnostic performance threshold b, MILP training set $\mathcal{K}_{large}$, number of configuration update steps k, configuration predictor networks $\{f^j_{\theta^0}\}^k_{j=1}$, separation rounds $\{n_j\}^k_{j=1}$, training epochs T, number of MILP instances P per epoch, number of samples to collect reward labels per epoch D, UCB scaling factor γ, UCB regularization parameter λ

Output: Trained regression networks $\{f^j_{\theta^T}\}^k_{j=1}$

1 $A \leftarrow$ Configuration_Space_Restriction ($\mathcal{K}_{small}, O, |S|, |A|, b$) // See Alg. 1

2 $\{f^j_{\theta^T}\}^k_{j=1} \leftarrow$ Forward_Training ($\mathcal{K}_{large}, k, \{f^j_{\theta^0}\}^k_{j=1}, \{n_j\}^k_{j=1}, T, P, D, \gamma, \lambda, A$) // See Alg. 2

(2) Instance Agnostic Configuration: The first step of Alg. 1 is to sample a large subspace S with $|S| \approx 2000$ due to computational infeasiblity to enumerate all $\{0, 1\}^M$ configurations (See Appendix A.3.1). For each configuration $s \in S$, we compute its empirical instance-agnostic performance $\hat{\bar{\delta}}(s) = \frac{1}{|\mathcal{K}_{small}|} \sum_{x\in\mathcal{K}_{small}} \delta(s, x)$ and choose the best (ERM) configuration $\tilde{s}_S = \arg\max_{s\in S} \hat{\bar{\delta}}(s)$ as the instance agnostic configuration. We apply the same $\tilde{s}_S$ to all instances in the given MILP class to evaluate its performance. Notably, $\tilde{s}_S$ always appears in the final restricted subspace A, as it is selected at the first iteration of Alg 1 when the marginal improvement of the instance-aware function (Line 14) and the instance-agnostic performance (Line 16) coincide.

(3) Random Within Restricted Subspace: Given the subspace A constructed by Alg. 1, we choose a configuration in the subspace A uniformly at random for each MILP instance.

A.6.2 Parameters

Table 7 and Table 8 present a list of parameters along with their experimental values used by our L2Sep method in Alg. 4.

A.6.3 SCIP interface.

We use a custom version of the SCIP solver (v7.0.2) Bestuzheva et al. [2021] provided by Paulus et al. [2022] and a custom version of the PySCIPOpt interface (v3.3.0) Maher et al. [2016] to add a

Table 7: A list of parameters and their values as used in the experiments for our data-driven subspace restriction Alg. 1.

Tang	Bin. Pack.	Max. Cut	Pack.
Number of initial config. samples $\|S\|$	1795	1795	2691
Size of restricted config. space $\|A\|$	30	20	15
Filtering threshold b	0.3	0.6	0.0
Ecole	Comb. Auc	Indep. Set	Fac. Loc.
Number of initial config. samples $\|S\|$	1699	1699	1923
Size of restricted config. space $\|A\|$	25	15	20
Filtering threshold b	0.5	0.4	0.0
Real-world	MIPLIB	NNV	Load Bal.
Number of initial config. samples $\|S\|$	2179	2691	1795
Size of restricted config. space $\|A\|$	20	20	25
Filtering threshold b	-0.17	0.0	-0.1

Table 8: A list of parameters and their values as used in the experiments for our learning Alg. 2 and 3.

Parameter	**Value**
Number of separators M	17
Number of configuration updates k	2
Configuration update round n_1	0
Configuration update round n_2	5 for Tang instances 8 for Ecole instances 10 for MIPLIB, NNV, Load Balancing
Training epoch T	70
Number of MILP instances per epoch P	6
Number of samples to collect reward labels per epoch D	8
Number of solver runs to collect a reward label for each config.-instance pair l	3
Reward clipping constant r_{min}	-1.5
UCB scaling factor γ	0.9375
UCB regularization parameter λ	0.001

special separator invoked at the beginning of each separation round to activate and deactivate $M = 17$ standard separators implemented in SCIP, including *aggregation, cgmip, clique, cmir, convexproj, disjunctive, eccuts, flowcover, gauage, gomory, impliedbounds, intobj, mcf, oddcycle, rapidlearning, strongcg, zerohalf*.[2]

A.6.4 Training and evaluation details

Architecture and training hyperparameters. Our network first embeds $\mathbf{V} \in \mathbb{R}^{n\times 17}$, $\mathbf{C} \in \mathbb{R}^{m\times 34}$, and $\mathbf{S} \in \mathbb{R}^{M\times 1}$ (where n, m, M are the number of variables, constraints, and separators) into hidden representations of dimension $d_{hidden} = 64$ with a BatchNorm followed by two (Linear, ReLU) blocks. Then, our Graph Convolution module Kipf and Welling [2017] takes the hidden embeddings for message passing, following the direction of ($\mathbf{V}\to\mathbf{C}\to\mathbf{V}$, $\mathbf{S}\to\mathbf{V}\to\mathbf{S}$, and $\mathbf{S}\to\mathbf{C}\to\mathbf{S}$), with a final (LayerNorm, ReLU, Linear) block that maintains the dimension d_{hidden}. Then, the separator nodes $\mathbf{S}$ pass through a TransformerConv attention module Shi et al. [2021] with $n_{heads} = 4$ heads and a $dropout$ rate of 0.1. Lastly, we perform a global mean pooling on each of the $\mathbf{C}$, $\mathbf{V}$, and $\mathbf{S}$ hidden embeddings to obtain three embedding vectors, concatenate them into a single vector, and finally use a (Linear, ReLU, Linear) block to map the vector into a scalar output. We train with Adam optimizer with a learning rate of 0.001 and a batch size of 64 for 70 epochs with a total of 180000 gradient

[2]Detailed descriptions of the separators can be found in https://www.scipopt.org/doc-7.0.2/html/group__SEPARATORS.php

Table 9: **Architecture hyperparameters.**

Input dimension $n \times d_n$ $m \times d_n$ $s \times d_s$	$\mathbf{V} \in \mathbb{R}^{n \times 17}$ $\mathbf{C} \in \mathbb{R}^{m \times 34}$ $\mathbf{S} \in \mathbb{R}^{M \times 1}$	GCN Message Passing Order	$\mathbf{V} \to \mathbf{C} \to \mathbf{V}$, $\mathbf{S} \to \mathbf{V} \to \mathbf{S}$, $\mathbf{S} \to \mathbf{C} \to \mathbf{S}$
Output dimension	1	Attention Num. Heads	4
Embedding dimension d_{hidden}	64	Attention Dropout	0.1
		Activation	ReLU

Table 10: **Training hyperparameters.**

Optimizer	Adam
Learning rate	0.001
Batch size	64
Num. of Gradient Steps	180000

steps. All hyperparameters are selected on the validation set and frozen before evaluating on the test set. Table 9 and 10 provides a list of hyperparameters.

Data split, MILP solver termination criterion, and inference strategy. For all MILP classes except MIPLIB, we collect a small training set $\mathcal{K}_{small}$ of 100 instances for configuration space restriction, and a large training set $\mathcal{K}_{large}$ of 800 instances for predictor network training. We hold out a validation set and a test set of 100 instances each. For MIPLIB, our curated subset contains 443 instances in total. We split the instances into $\mathcal{K}_{small}$ with 30 instances, $\mathcal{K}_{large}$ with 270 instances, a validation set with 55 instances, and a test set with 88 instances.

For all MILP classes except MIPLIB and Load Balancing, we solve the instances until optimality. For MIPLIB and Load Balancing, we solve all instances until it reaches a primal-dual gap of 10%.

For Packing and Bin Packing, our best inference selection strategy, chosen based on validation performance, is to select the configuration with the highest predicted reward $s^*_{x,j} = \arg\max_{s \in A} \tilde{f}^j_\theta(x, s)$. For all other MILP classes, the best inference selection strategy is to select the configuration with the highest UCB score $s^*_{x,j} = \arg\max_{s \in A} ucb(x, s)$.

Training and evaluation. We collect data, train, validate and test all methods on a distributed compute cluster using nodes equipped with 48 Intel AVX512 CPUs. A single Nvidia Volta V100 GPU is used to train all MILP classes except MIPLIB, as the massive number of variables and constraints (up to 1.4×10^6) in certain MIPLIB instances poses memory challenges for the GPU device (with 16GB memory). Our configuration subspace restriction Alg. 1 requires approximately 36 hours, while the training time for Alg. 2 is within 48 hours for all benchmarks including MIPLIB. In certain cases, the baseline methods take excessively long solve time, making it computationally expensive to evaluate those methods. During test evaluation, we terminate the solve if the solve time exceeds three times the SCIP default (which can be up to 20 minutes), resulting in a time improvement as low as -300%. This time limit is more relaxed than the reward clipping $r_{\text{min}} = -1.5$ during training (which corresponds to $\leq -150\%$ time improvement and is applied to improve data collection efficiency). Notably, the hard stop is not used for our complete learning method (L2Sep), whose performance improvement from SCIP default remains consistently stable across instances. Instead, it is primarily used for the random or prune baseline that exhibits very poor performance or a significantly large standard deviation. Without the time limit, these baselines may exhibit worse performance than those reported in Table 1 and 3 of the main paper.

A.6.5 MILP benchmarks

We perform extensive evaluation on the following MILP benchmarks of a variety of different sizes.

Standard: Tang et al. We consider three out of four MILP classes introduced in Tang et al. [2020]: Maximum Cut, Packing, and Binary Packing. We follow Paulus et al. [2022] to only consider the large size where the number of variables and constraints $n, m \in [50, 150]$, as Paulus et al. [2022] observe the small and medium size are too easy for the SCIP solver. We do not consider the Planning class because the large size is still too small, as SCIP takes less than $0.01s$ on average to solve the instances.

We generate large size Tang instances with class specific parameters $n = 60$ variables, $m_{\text{resource}} = 60$ constraints for Packing, $n = 66$ variables, $m_{\text{resource}} = 132$ constraints for Binary Packing, and $n_{\text{vertices}} = 54, n_{\text{edges}} = 134$ for Maximum Cut.

Standard: Ecole. We consider three out of four classes of instances from [Prouvost et al., 2020]: Combinatorial Auction, Independent Set, and Capacitated Facility Location, where the number of variables and constraints in the instances $n, m \in [100, 10000]$.

We generate Ecole instances with class specific parameters $n_{\text{items}} = 100, n_{\text{bids}} = 500$ for Combinatorial Auction, $n_{\text{nodes}} = 500$ for Independent Set, $n_{\text{rows}} = 500, n_{\text{columns}} = 1000$ for Set Cover, and $n_{\text{customers}} = 100, n_{\text{facilities}} = 100$ for Capacitated Facility Location.

When we use the default parameters in the Ecole library to generate the Combinatorial Auction and Independent Set datasets[3], we notice that the instance-aware ERM (optimal) predictor sometimes only selects a small number of configurations within each class, resulting in similar performance as the instance-agnostic configuration (single configuration) due to selection homogeneity. Despite both resulting in significant relative time improvements from SCIP default, the default parameters cause the instances within each of these two classes too similar to one another, thereby limiting the benefits of instance-aware configurations. To enhance instance diversity, we adjust the parameters for both classes by sampling uniformly at random from a broad distribution, as presented in Table 11, for each MILP instance. As many large-scale real-world MILP problems exhibit significant dataset heterogeneity (for instance, see MIPLIP below), we believe that our adjusted Ecole datasets better reflect realistic MILP scenarios.

Table 11: Sampling parameter distributions for each Independent Set and Combinatorial Auction instance.

Independent Set		
graph_type	edge_probability	affinity
$U(\{\text{barabasi\_albert}, \text{erdos\_renyi}\})$	$U([0.005, 0.01])$	$U(\{2, 3, 4, 5, 6\})$
Combinatorial Auction		
value_deviation	add_item_prob	max_n_sub_bids
$U([0.25, 0.75])$	$U([0.5, 0.75])$	$U(\{3, 4, 5, 6, 7\})$
additivity	budget_factor	resale_factor
$U([-0.1, 0.4])$	$U([1.25, 1.75])$	$U([0.35, 0.65])$

We do not report results for Set Cover as we find an instance-agnostic configuration s^{ERM} that deactivates *all* separators is able to achieve a relative time improvement of 88.6% (IQM: 88.4%, mean: 87.4%, standard deviation 6%) from SCIP default, whereas the default median solve time is $5.96s$ (IQM $6.13s$, mean: $6.45s$, standard deviation $3.17s$). Furthermore, on the small training set $\mathcal{K}_{small}$, we find that the instance-aware empirical performance of the ERM predictor $\tilde{f}^{ERM}$ differs by less than 1% from the performance of s^{ERM}. We experiment with various parameters for the Set Cover problem, such as the number of rows and columns, density of the constraint matrix, and maximum objective coefficient, but observe similar outcomes. This implies that separators in the SCIP solver do not provide significant benefits for the Set Cover class, and therefore, the instance-aware ERM predictor aligns with the instance-agnostic configuration to deactivate all separators.

Large-scale: NN Verification. We consider the large-scale neural network verification instances used in Paulus et al. [2022], with a median size (n, m) of the number of variables and constraints at $(7142, 6531)$. This dataset formulates MILPs to verify whether a convolutional neural network

[3] `https://doc.ecole.ai/py/en/stable/reference/instances.html` provides a list of adjustable parameters for each Ecole MILP class.

is robust to input perturbations on each image in the MNIST dataset. The MILP formulation of the verification problem can be found in Gowal et al. [2018]. We follow Paulus et al. [2022] to exclude all infeasible instances (often trivially solved at presolve) and instances that reach a 1-hour time-limit in SCIP default mode. Due to differences in the learning tasks (separator configuration for the entire solve v.s. cut selection at the root node), we further exclude instances that cannot be solved optimally within 120 seconds, after which 74% of the instances used in Paulus et al. [2022] remain.

Large-scale: MIPLIB. MIPLIB2017 collection Gleixner et al. [2021] is a large-scale *heterogeneous* MILP benchmark dataset that contains a curated set of challenging real-world instances from various application domains. It contains 1065 instances where the number of variables and constraints (n, m) varies from 30 to $1,429,098$. Previous work Turner et al. [2022] finds it very challenging to learn over MIPLIB due to dataset heterogeneity. A recent work Wang et al. [2023] attempts to learn cutting plane selection over two subsets of similar instances (with 20 and 40 instances each), curated via instance clustering from two starting instances containing knapsack and set cover constraints.

We attempt to learn over a larger heterogeneity subset of the MIPLIB dataset to solve the separator configuration task. We adopt a similar dataset pre-filtering procedure as in Turner et al. [2022], where we discard instances that are infeasible, solved after presolving, or the primal-dual gaps are larger than 10% after 300 seconds of solve time. It is worth noting that our pre-filtering procedure preserves the dataset heterogeneity, whereas Wang et al. [2023] reduces the dataset to homogeneous subsets. We obtain a subset of 443 instances, which is around 40% of the original MIPLIB dataset.

Large-scale: Load Balancing (ML4CO Challenge). We consider the server load balancing dataset from Neurips 2021 ML4CO Challenge Gasse et al. [2022] [4], whose average number of variables and constraints (n, m) is at $(64304, 61000)$. This dataset is inspired by real-world applications in distributed computing, where the goal is to allocate data workloads to the fewest number of servers possible while ensuring that the allocation remains resilient to the failure of any worker. This is a challenging dataset due to the large number of variables and constraints, along with the presence of nonstandard robust apportionment constraints that separators in MILP solvers may not be specialized for. No instances from the original load balancing dataset are pre-filtered, as we observe that the primal-dual gaps for all instances remain within 10% after 300 seconds of solve time.

The ML4CO challenge releases two other datasets: Item Placement and Anonymous. We follow Wang et al. [2023] and exclude Item Placement since very few cutting planes are generated and used in the dataset (with an average of less than five candidate cuts on each instance), thereby limiting the influence of separators on the dataset. Additionally, we exclude Anonymous because the dataset is restrictively small for effective learning (comprising only 98 train and 20 valid instances).

[4] More details on the dataset can be found at `https://www.ecole.ai/2021/ml4co-competition/`

A.7 Ablation Details

A.7.1 Implementation details of ablation methods

Configuration space restriction (Sec. 5.1):

(1) No Restr.: We do not constraint the output space of the $k = 2$ instance-aware predictors, allowing them to output any configuration in the unrestricted space $\{0, 1\}^M$. Due to the vast number of actions, it is infeasible to use the neural UCB Alg. 3 as the normalizing matrix Z becomes excessively large, making it challenging to obtain meaningful confidence bounds as most actions remain unexplored. Hence, we use the following ϵ-greedy exploration strategy (similar to the ϵ-**greedy** ablation below): we first sample a subset of 50 configurations for training efficiency. Then, we iteratively select D configurations from the subset to collect the reward labels: at each of the D iteration, with probability $\epsilon = 0.1$ we select a configuration uniformly at random among the unselected ones in the subset, and with probability $1 - \epsilon$ we choose the configuration with the highest reward point estimate among the unselected ones in the subset. At inference time, we sample a subset of 500 configurations for each MILP instance, and select the configuration with the highest reward point estimate. All the other settings remain the same as our complete method (L2Sep). This ablation is used to examine the effectiveness of restricting configuration space in learning better configuration predictors.

(2) Greedy Restr.:
When we run Alg. 2 to obtain the configuration subspace A', we select configurations solely with the greedy strategy based on the highest marginal improvement, but we do not filter out configurations with low instance-agnostic performance (equivalent, we set the filtering threshold $b = -\infty$). The rest of the learning remains identical to our complete method (L2Sep), but with the predictor's output restricted to the greedy subspace A'. This ablation allows us to assess the effectiveness of the filtering strategy in constructing a superior configuration subspace for learning predictors.

Configuration step restriction (Sec. 5.2):

(1) $\mathbf{k = 1}$: We perform one configuration update for each MILP instance. We use the first config. predictor $\tilde{f}_\theta^1$ from our complete method (L2Sep) to set configuration at separation round n_1.

(2) $\mathbf{k = 3}$: We fix the two configuration predictors $\tilde{f}_\theta^1$ and $\tilde{f}_\theta^2$ from our complete method (L2Sep), and follow Alg. 2 to train a third configuration predictor $\tilde{f}_\theta^3$ at separation round n_3 ($= 12$ for Tang instances and 20 for Ecole instances). The predictor $\tilde{f}_\theta^3$ is similarly restricted to the subspace A. At test time, for each instance x, we follow our complete method to perform two configuration updates using $\tilde{f}_\theta^1$ and $\tilde{f}_\theta^2$ until separation round n_3. Then, we use $\tilde{f}_\theta^3$ to update the configuration at separation round n_3 and hold it fixed until the solving process terminates.

Neural UCB Algorithm (Sec. 5.3):

(1) Supervise ($\times 4$): We exactly follow our complete method (L2Sep), except that we use offline regression instead of online neural UCB to train the predictors. Offline regression first collects a large training set of instance-configuration-reward tuples offline (by randomly sampling instance-configuration pairs and using the MILP solver to obtain the reward labels). The network is then trained on the fixed training set. In contrast, neural UCB gradually expands the training buffer by collecting training data online, while using the current trained model to guide exploration (sampling configurations with high uncertainty) and exploitation (sampling configurations with high predicted reward). Model training is performed online on the continually updated dataset. Due to the difference in online v.s. offline nature of the dataset generation and training scheme, one learning method may require more gradient updates to converge than the other. To attempt at a fair comparison, we collect $\times 4$ more instance-configuration-reward tuples and train the offline network until convergence. This ablation is used to examine the efficacy of online learning through neural contextual bandit in improving the training efficiency of predictor networks.

(2) ϵ**-greedy**: We exactly follow our complete method, except when training the predictors with the Neural Contextual Bandit algorithm in Alg. 3, we use an ϵ-greedy strategy to iteratively sample the D configurations: at each of the D iteration, with probability $\epsilon = 0.1$ we select a configuration uniformly at random among the unselected ones in A, and with probability $1 - \epsilon$, we select the configuration with the highest reward point estimation among the unselected ones in A. The ϵ-

greedy strategy does not require confidence bound estimation and has been commonly used in deep reinforcement learning (such as deep Q Learning Mnih et al. [2015]) when the action space is large. This ablation allows us to assess the effectiveness of upper confidence bound (UCB) estimation for better a exploration-exploitation trade-off in neural contextual bandit.

A.7.2 Additional ablation results and analysis

Ablation: Different configuration subspaces for different configuration update steps

Our complete method (L2Sep) in the main paper constructs the subspace A only once at the initial update for computational efficiency benefits. We present the ablation where we construct a new subspace A_2 at the second update. Specifically, for each MILP instance $x \in \mathcal{K}_{small}$, we use the first trained predictor $\tilde{f}^1_{\theta^T}$ to set the initial configuration at separation round n_1 and hold the configuration between separation rounds $[n_1, n_2]$. At the start of separation round n_2, the MILP instance is updated to $\tilde{x}$, which combines the original MILP x with the newly added cuts. For ease of explanation, we let $\mathcal{K}_{small,2}$ denote the updated training set of MILP instances where each instance $\tilde{x}$ starts at separation round n_2. We run Alg. 1 on $\mathcal{K}_{small,2}$ to select a configuration subspace A_2 based on the time improvement when we set the configuration at separation round n_2 and maintain it until the solving process terminates. We use the same set of parameters (including filtering threshold b, size of the subspace $|A|$, and size of the initial samples $|S|$) as when we construct A.

We first compare our choice of reusing the previous subspace A with the ablation method of updating the subspace to A_2 by evaluating the two terms in Eq. (3) of the main paper that balances training set performance and generalization: (1) the empirical performance of the instance-aware ERM predictors, $\hat{\Delta}(\tilde{f}^{ERM}_{A,2})$ and $\hat{\Delta}(\tilde{f}^{ERM}_{A_2,2})$, which we denote as A: $1st$ and A_2: $1st$, and (2) the average empirical instance-agnostic performance, $\frac{1}{|A|}\sum_{s\in A}\hat{\bar{\delta}}(s)$ and $\frac{1}{|A_2|}\sum_{s\in A_2}\hat{\bar{\delta}}(s)$, which we denote as A: $2nd$ and A_2: $2nd$, on the updated training set $\mathcal{K}_{small,2}$. The ERM predictors $\tilde{f}^{ERM}_{A,2}$ and $\tilde{f}^{ERM}_{A_2,2}$ set the configuration once at separation round n_2 by choosing optimally within the subspace A and A_2 for each instance in the updated training set $\mathcal{K}_{small,2}$.

Table 12: Comparison of (1) the empirical perf. of the instance-aware ERM predictor $\hat{\Delta}(\tilde{f}^{ERM}_{A})$ and $\hat{\Delta}(\tilde{f}^{ERM}_{A,2})$, denoted as A: $1st$ and A_2: $1st$, and (2) the average empirical instance-agnostic perf. $\frac{1}{|A|}\sum_{s\in A}\hat{\bar{\delta}}(s)$ and $\frac{1}{|A_2|}\sum_{s\in A_2}\hat{\bar{\delta}}(s)$, denoted as A: $2nd$ and A_2: $2nd$, on the reused subspace A and the updated subspace A_2.

	A: $1st$	A: $2nd$	A_2: $1st$	A_2: $2nd$
Bin. Pack.	63.1%	28.9%	68.9%	39.3%
Pack.	47.2%	5.8%	44.3%	8.2%
Indep. Set	76.1%	53.9%	77.9%	57.8%
Fac. Loc.	37.7%	7.3%	46.7%	13.1%

Table 12 displays the relevant statistics on the four ablations MILP classes. We observe an overall decrease in both terms when we re-use the subspace A instead of using the updated subspace A_2, although the difference is relatively small. Notably, in the Packing class, the 1^{st} term of the reused subspace A is higher than that of the updated subspace A_2. This is because the filtering criterion for the updated subspace A_2 excludes certain configurations that enhance the instance-aware performance of the ERM predictor, but have low instance-agnostic performance. While these configurations pass the filtering criterion during the initial update, they are subsequently filtered out when we construct the updated space A_2 with a slight sacrifice in instance-aware performance.

We further follow our reported method (L2Sep) to train a second configuration predictor $\tilde{f}^{2,2}_{\theta^T}$ using neural UCB (Alg. 3) within the updated subspace A_2. We denote the updated method as L2Sep+. The performance results on the four ablation benchmarks are reported in Table 13.

We find that our reported model L2Sep (with a single configuration subspace A) exhibits similar performance as the updated model L2Sep+ (with the subspace A for the first update and an updated subspace A_2 for the second update), albeit L2Sep+ performing slightly better. This observation validates our decision to reuse the subspace A for the second configuration update, as it offers computational efficiency advantages by reducing the number of MILP solver calls during training (by avoiding a second invocation of Alg 1). We attribute this outcome to (i) the diversity of configurations within the subspace A, and (ii) the presence of similar characteristics within a solve for the same MILP instance. Combined, these factors allow the subspace A constructed during the initial update

Table 13: Performance (median, mean, interquartile mean, and standard deviation) of our method where we use the same subspace A for both $k = 2$ updates (L2Sep) v.s. we update a new subspace A_2 for the second update (L2Sep+).

	Bin. Pack.				Pack.			
Method	Median	Mean	IQM	STD	Median	Mean	IQM	STD
L2Sep	42.3%	33.0%	40.5%	34.2%	28.5%	17.7%	25.2%	39.3%
L2Sep+ (New A)	43.1%	34.1%	41.1%	34.9%	29.7%	19.6%	28.1%	38.5%
	Indep. Set				Fac. Loc.			
	Median	Mean	IQM	STD	Median	Mean	IQM	STD
L2Sep	72.4%	60.1%	69.8%	27.8%	29.4%	18.2%	27.5%	39.6%
L2Sep+ (New A)	68.7%	61.8%	67.7%	25.1%	29.8%	23.0%	28.7%	32.1%

to effectively cover the high-performance configurations across various separation rounds.

We also observe that L2Sep+ with the updated subspace A_2 demonstrates more significant improvements in mean performance compared to other metrics. We believe this is also because the instance-agnostic performance of a small number of configurations in the first subspace A may degrade during the second update step, resulting in subpar performance on a few outlier instances that negatively impact the mean (see a similar discussion in Appendix A.8.1). However, by reconstructing the second subspace A_2, we effectively eliminate those configurations through the second filtering pass, thereby enhancing the robustness of subspace A_2 to outlier instances. Hence, the choice to update the second subspace involves a trade-off between computational efficiency and robustness on the outlier instances, and ultimately, should be decided based on the characteristics of the specific real-world application when deploying our method.

A.8 Detailed Experiment Results

A.8.1 Interquartile mean (IQM) and mean statistics

While we report the median and standard deviation in Table 1 and 3 of the main paper, we provide the mean and interquartile mean (IQM) statistics in the following tables 14, 15 and 16. We observe that the interquartile mean performance closely aligns with the median performance reported in the main paper. Meanwhile, the mean improvements are lower than the interquartile mean for all methods. Upon examining the performance of individual instances, we observe that the performance degradation comes from a small number of outlier instances with negative time improvement; on the majority of instances, our learning method is able to achieve significant improvement from SCIP default. Addressing such outlier instances is left as a future work.

Table 14: **Tang et al.** The IQM and mean of the absolute solve time of SCIP default and relative time improvement of different methods across the test set (higher the better, best are bold-faced).

		Packing		**Bin. Packing**		**Max. Cut**	
	Method	IQM	Mean	IQM	Mean	IQM	Mean
	Default Time (s)	9.13s	17.75s	0.096s	0.14s	1.77s	1.80s
Heuristic Baselines	Default	0%	0%	0%	0%	0%	0%
	Random	-102.5%	-117.9%	-112.2%	-122.4%	-143.8%	-131.4%
	Prune	6.4%	1.1%	16.5%	17.0%	4.3%	12.3%
Ours Heuristic Variants	Inst. Agnostic ERM Config.	17.9%	13.1%	34.9%	33.3%	70.2%	68.7%
	Random within Restr. Subspace	17.6%	12.2%	28.2%	25.0%	67.2%	66.6%
Ours Learn	L2Sep	**25.2%**	**17.7%**	**40.5%**	**33.0%**	**71.8%**	**68.9%**

Table 15: **Ecole.** The IQM and mean of the absolute solve time of SCIP default and relative time improvement of different methods across the test set (higher the better, best are bold-faced).

		Indep. Set		**Comb. Auction**		**Fac. Location**	
	Method	IQM	Mean	IQM	Mean	IQM	Mean
	Default Time (s)	17.52s	67.18s	2.81s	4.18s	63.58s	77.90s
Heuristic Baselines	Default	0%	0%	0%	0%	0%	0%
	Random	-101.5s	-111.5%	-132.0%	-129.2%	-125.3%	-128.6%
	Prune	14.5%	15.0%	12.2%	14.6%	24.2%	14.3%
Ours Heuristic Variants	Inst. Agnostic ERM Config.	57.2%	51.5%	60.9%	56.8%	12.4%	10.9%
	Random within Restr. Subspace	50.0%	30.0%	59.2%	56.4%	17.8%	13.7%
Ours Learn	L2Sep	**69.8%**	**60.1%**	**65.9%**	**62.2%**	**27.5%**	**18.2%**

Table 16: **Real-world MILPs.** The IQM and mean of the absolute solve time of SCIP default and relative time improvement of different methods across the test set (higher the better, best are bold-faced).

		NN Verif.		MIPLIB		Load Balancing	
	Method	IQM	Mean	IQM	Mean	IQM	Mean
	Default Time (s)	33.48s	36.99s	16.32s	45.50s	32.47s	32.92s
Heuristic Baselines	Default	0%	0%	0%	**0%**	0%	0%
	Random	-178.0%	-154.6%	-161.7%	-150.1%	-300.1%	-229.6%
	Prune	30.8%	24.4%	5.2%	-30.5%	-14.7%	-71.1%
Ours Heuristic Variants	Inst. Agnostic ERM Config.	30.2%	25.0%	3.4%	-19.1%	11.2%	12.8%
	Random within Restr. Subspace	29.9%	26.1%	-11.3%	-31.4%	10.0%	8.4%
Ours Learn	L2Sep	**34.8%**	**29.9%**	**11.9%**	-8.0%	**21.1%**	**18.6%**

A.8.2 Result contextualization

A previous work by Paulus et al. [2022] evaluates their learning-based method for cutting plane selection on the NN Verfication dataset. As shown in Table 3 of their paper, their best model achieves a median solve time of 20.89s, whereas the default SCIP solver takes a median solve time of 23.65s, resulting in a median relative speed up of 11.67%. While the comparison is far from perfect, our method achieves a higher median relative time improvement of 37.5%. We note that it is reasonable for their reported absolute solve time to be different from ours due to differences in computational machines. We also perform a rough comparison to the MIPLIB results reported in Wang et al. [2023], which, same as Paulus et al. [2022], learns to select cutting planes. In Table 1 of their paper, they report SCIP default takes $256.58s$ and $164.61s$ on average to solve two small homogenous MIPLIB subsets, whereas their cutting plane selection method improves the solve time to $248.66s$ and $162.96s$, leading to a 3% and 1% improvement. Although not a perfect comparison, our L2Sep achieves a higher median time improvement of 12.9% (and an interquartile mean time improvement of 11.9%) on our larger heterogeneous subset (See Appendix A.6.5 for a detailed dataset description).

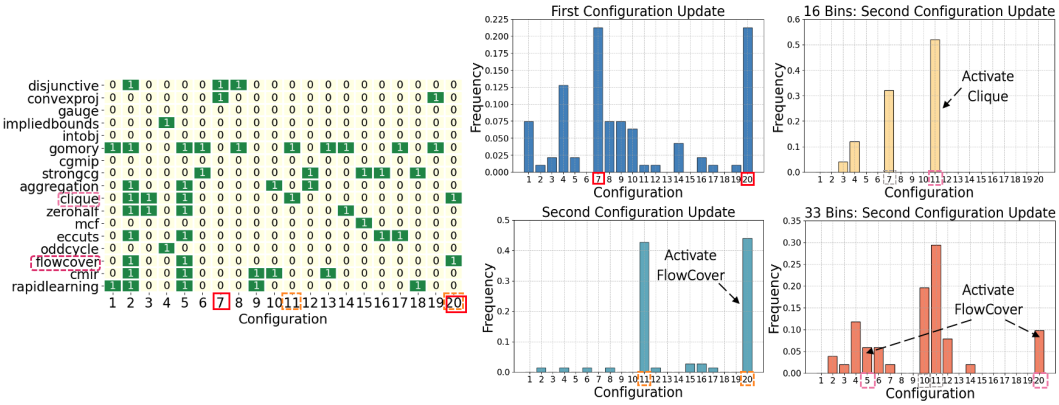


Figure 8: **Bin Packing Interpretation.** (Left / Heatmap) Each row is a separator, and each column is a configuration in our restricted space. Green and yellow cells indicate activated and deactivated separators by each configuration, respectively. (Middle / Histogram Left Column) The frequency of each configuration selected by our learned model at the 1^{st} and 2^{nd} config. update on the original test set with *66 bins*. High frequency configurations are marked with red and orange squares, respectively. The config. indices in the heatmap and the histograms align. (Right / Histogram Right Column) The frequency of each configuration selected at the 2^{nd} config. update (the 1^{st} update has similar results) when we gradually decrease the number of bins (bottom: *33 bins*; top *16 bins*). We observe the prevalence of FlowCover and Clique decreased and increased, respectively.

A.8.3 Interpretation analysis: L2Sep recovers effective separators from literature

In Fig. 8, 9, 10, and 11, we provide visualizations of (1) the restricted configuration subspace A, as shown in the heatmap plots, and (2) the frequency for each configuration to be selected by L2Sep on the test set, as shown in the histogram plots, for all MILP classes that we study. As described in the main paper, for Bin Packing, Independent Set, and MIPLIB, the visualization provides meaningful interpretations that recover known facts from the mathematical programming literature.

Besides the known results, we also observe some intriguing unexpected scenarios from the visualizations. For Independent Set, L2Sep deactivates all separators with a frequency of 20% at the 2^{nd} configuration update, whereas all selected configurations at the 1^{st} update activate a substantial amount of separators. It is an interesting question to investigate why it is better to deactivate all separators for a certain subset of Independent Set instances at later separation rounds.

For Maximum Cut, OddCycle Boros et al. [1992], Jünger and Mallach [2021] and ZeroHalf Caprara and Fischetti [1996] are known to be effective in the literature. Interestingly, none of the selected configurations activate ZeroHalf for both configuration updates; OddCycle is also completely deactivated for the 1^{st} update, but is activated with a frequency of 14% at the 2^{nd} update. Meanwhile, we observe that Disjunctive, FlowCover, and Aggregation separators are more frequently selected.

We hope that by providing the visualization results, L2Sep can serve as a driver of future works on improved (theoretical) polyhedral understanding of different MILP classes, and potentially seed investigations (empirical and theoretical) for nonstandard, newly-proposed problems (e.g. NN Verification) where few analyses exists.

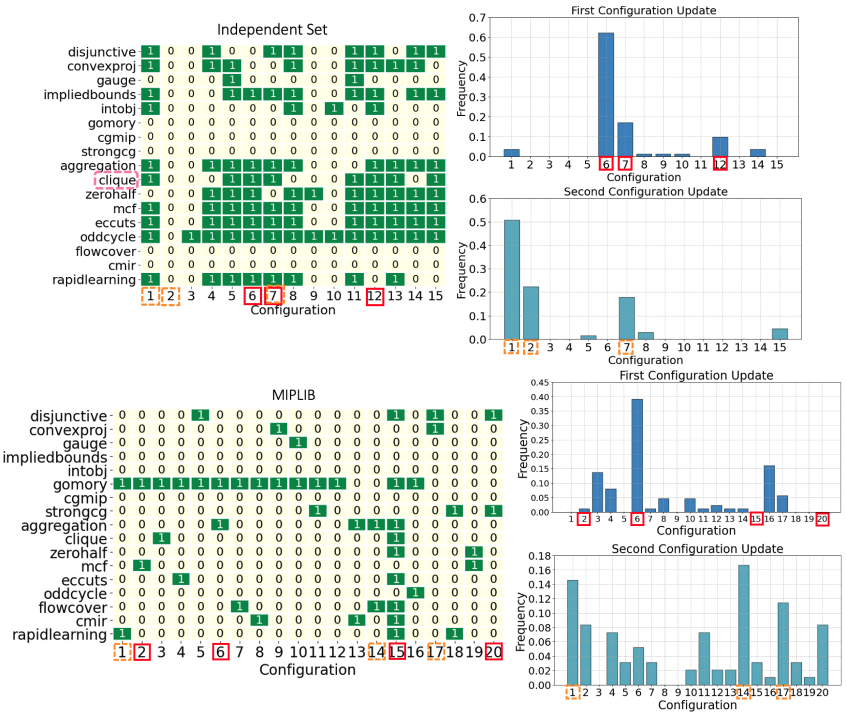
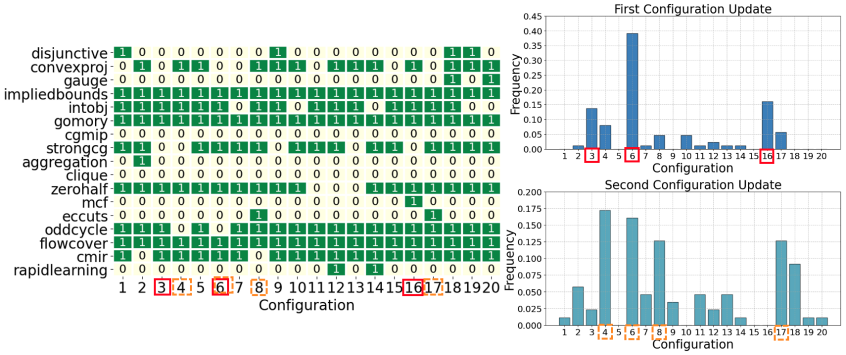
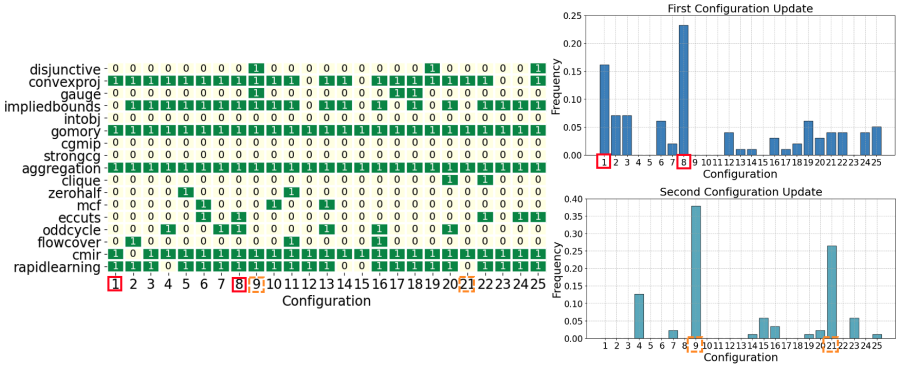


Figure 9: **(Top) Independent Set. (Bottom) MIPLIB.** The same set of figures as Fig. 8 (Left / Heatmap) and (Middle / Histogram Left Column). See interpretations in the main paper.

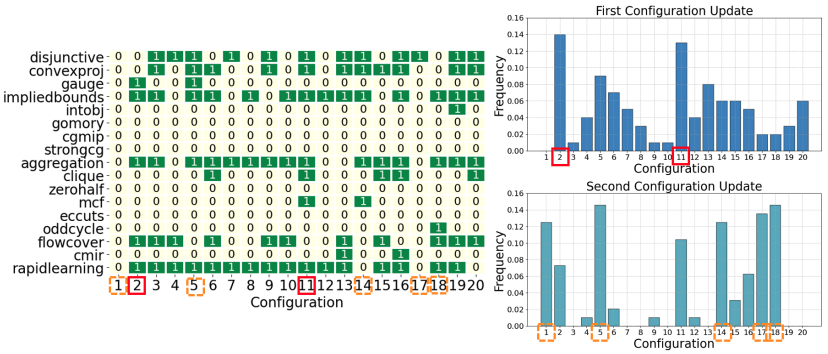


(a) NN Verification.

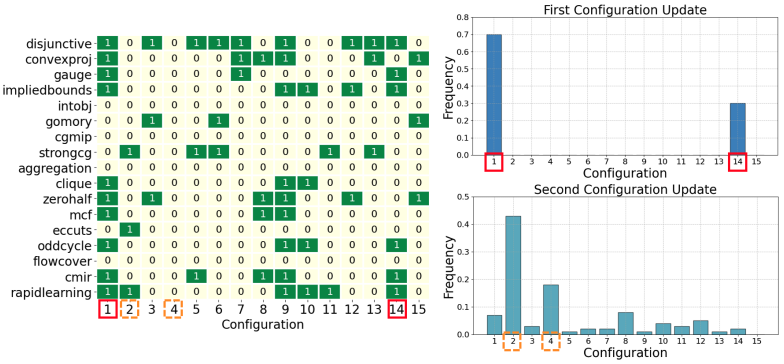


(b) Load Balancing.

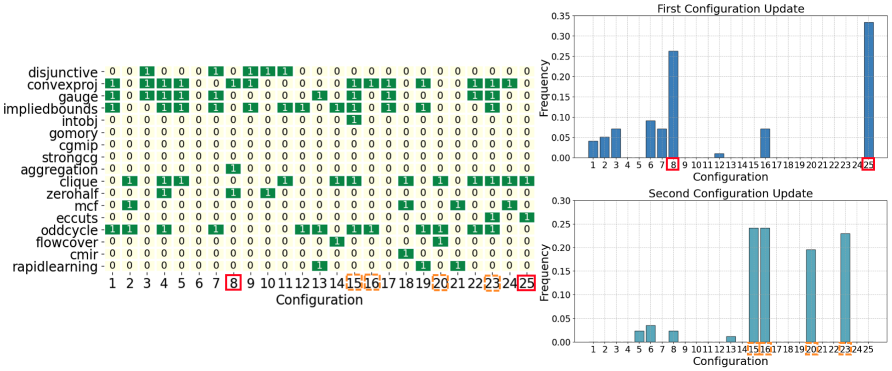
Figure 10: **Other real-world MILP classes.** The same set of figures as Fig. 8 (Left / Heatmap) and (Middle / Histogram Left Column).



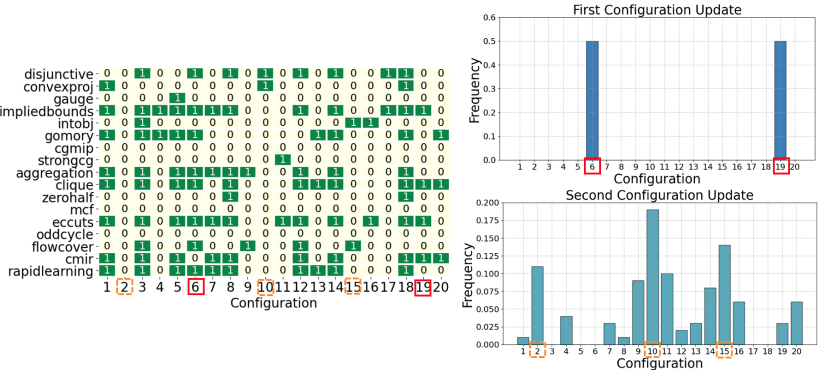
(a) Max. Cut



(b) Packing



(c) Combinatorial Auction



(d) Facility Location

Figure 11: **Other Tang et al. and Ecole MILP classes.** The same set of figures as Fig. 8 (Left / Heatmap) and (Middle / Histogram Left Column).

Table 17: **Alternative objective (relative gap improvement).** Absolute gap of SCIP default under the fixed time limit, and the mean (lower the better, best are bold-faced) and standard deviation of the relative gap improvement of different methods with respect to SCIP default.

	Methods	Pack.	Comb. Auc.	Indep. Set	NNV	Load Balancing
	Time Limit (s)	4.4s	1.4s	8.2s	16s	16s
	Default Gap	9.1e-4 (9.3e-4)	0.060 (0.098)	0.057 (0.059)	0.50 (0.80)	0.32 (0.13)
Heuristic Baselines	SCIP Default	0%	0%	0%	0%	0%
	Random	-37.1% (41.7%)	-27.3% (69.1%)	-23.2% (44.1%)	-40.3% (72.8%)	-48.0% (35.8%)
Ours Heuristic Variants	Inst. Agnostic Configuration	11.9% (38.4%)	52.4% (45.3%)	23.5% (34.5%)	33.6% (72.1%)	14.0% (18.9%)
	Random within Rest. Subspace	10.1% (42.5%)	54.1% (45.1%)	21.6% (33.7%)	24.8% (75.9%)	9.5% (17.6%)
Ours Learned	**L2Sep**	**15.4% (40.0%)**	**68.8% (38.2%)**	**29.6% (34.7%)**	**36.0% (68.2%)**	**34.2% (27.5%)**

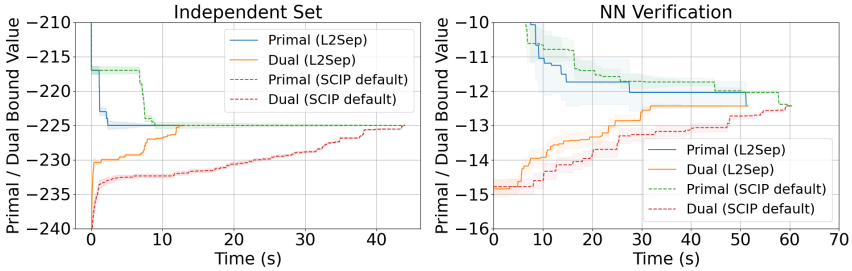


Figure 12: Primal-dual bound curves (median and standard error) for L2Sep and SCIP default on Independent Set and NN Verification. L2Sep can effectively tighten the dual bound faster.

A.8.4 The immediate and multi-step effect of separator configuration in the B&C process

Intelligently configuring separators have both immediate and multi-step effects in the B&C process:

Immediate: Some separators take a long time to run, but generate mostly low-quality cuts that are ultimately never selected by the downstream cut selector. Deactivating those separators leads to an immediate time improvement by reducing the time to generate the cut pool.

Multi-step: Improved separator configuration can tighten the dual bound faster through better-selected cuts; it may also accelerate other B&C components such as branching (e.g. strong branching requires solving many children LPs and hence may benefit from tighter dual bounds).

The Table 18 presents the total solve time and total separator execution time for our complete method L2Sep and SCIP default on several MILP classes. We report the median and standard deviation evaluated on 100 instances for each class. L2Sep significantly reduces the total separator execution time. Upon closer examination, we find that L2Sep adeptly deactivates expensive yet ineffective separators while activating effective ones.

In Fig. 12, we plot the primal-dual bound curves (median and standard error) of L2Sep and SCIP default on Independent Set and NN Verification. The significantly faster dual bound convergence of L2Sep demonstrates the multi-step effect of improved separator configurations. We further summarize the synergistic interaction effects between separator selection and other B&C components (branching, dual LP) in Table 19. Notably, even though our method does not modify branching, the branching solve time is reduced.

Table 18: **Immediate Effect of Separator Configuration.** The median (lower the better) and standard deviation (in parentheses) of the total solve time and total separator execution time for L2Sep and SCIP default.

	Total Solve Time (L2Sep)	**Total Separator Execution Time (L2Sep)**	**Total Solve Time (SCIP Default)**	**Total Separator Execution Time (SCIP Default)**
Comb. Auc.	**0.65s (2.36s)**	**0.02s (0.054s)**	3.01s (4.60s)	1.39s (1.14s)
Indep. Set	**3.81s (116.42s)**	**0.35s (9.94s)**	13.16s (120.89s)	7.38s (37.94)
NNV	**20.75s (18.56s)**	**0.16s (0.13s)**	34.76s (25.05s)	6.58s (8.05s)

Table 19: **Multi-Step Effect of Separator Configuration.** The median (lower the better) and standard deviation (in parentheses) of strong branching time, pseudocost branching time, and dual LP time for L2Sep and SCIP Default.

	Comb. Auc.		Indep. Set		NNV	
	L2Sep	SCIP Default	L2Sep	SCIP Default	L2Sep	SCIP Default
Strong Branching Time	**0.31s (1.15s)**	0.41s (1.79s)	**2.82s (21.44s)**	3.92s (19.68s)	**6.56s (4.06s)**	8.31s (5.55s)
Pseudocost Branching Time	**0.35s (1.46s)**	0.45s (2.17s)	**3.01s (22.29s)**	4.6s (21.67s)	**8.18s (4.81s)**	9.69s (6.13s)
Dual LP Time	**0.08s (0.58s)**	0.18s (0.83s)	**0.3s (73.95s)**	1.14s (55.51s)	**3.67s (6.52s)**	5.07s (5.81s)

A.8.5 Alternative objective: relative gap improvement

We analyzed an alternative objective of the relative gap improvement under a fixed time limit. Let $g_0(x)$ and $g_\pi(x)$ be the primal-dual gaps of instance x using the SCIP default and another configuration strategy $\pi(x)$ under a fixed time limit T. We define the relative gap improvement as $\delta_g(\pi(x), x) := (g_0(x) - g_\pi(x))/(\max\{g_0(x), g_\pi(x)\} + \epsilon)$. We choose the denominator to avoid division by zero when the instance is solved to optimality.

As seen from Table 17, L2Sep achieves a 15%-68% relative gap improvement over SCIP default. Specifically, the table presents the relative gap improvement (mean and standard deviation) of each method over SCIP default, along with the fixed time limit for various MILP classes (mostly around 50% of medium SCIP default solve time), and the absolute gap of SCIP default at the time limit. In Fig. 13, we further plot histograms of the gap distribution on the entire dataset for L2Sep and SCIP default, where we observe that L2Sep effectively shifts the entire gap distribution to a lower range. These results demonstrate the effectiveness of our method across different objectives, and its ability to improve primal-dual gaps for instances that cannot be solved to optimality within the time limit.

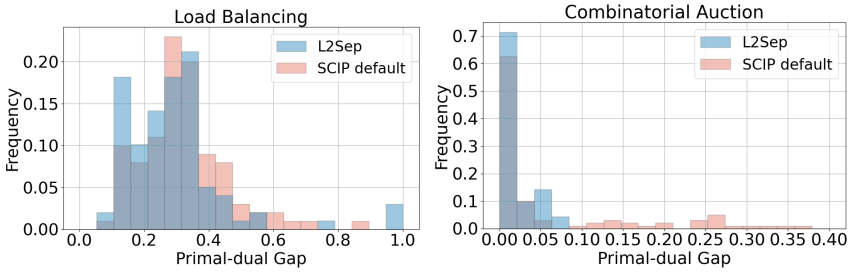


Figure 13: Distributions of absolute primal-dual gaps of L2Sep and SCIP default on Load Balancing and Combinatorial Auction. L2Sep is effective under an alternative objective (relative gap improvement).

A.9 Limitation.

While SCIP default does not require training as it sets pre-defined priorities and frequencies of separators (the same for all MILP instances), our learning method requires collecting data and fitting models for each MILP class, which introduces a time overhead. However, such a limitation is inherent in all learning for MILP methods Tang et al. [2020], Paulus et al. [2022], Wang et al. [2023], Gasse et al. [2019], as learning methods rely on training models to generalize to unseen test instances. Following these previous works, we train separate models for different MILP classes, although our L2Sep method exhibits a significant time improvement on the heterogeneous MIPLIB dataset, which suggests the possibility of learning an aggregated model for multiple MILP classes to potentially reduce training time. We leave this as a future work.

In the ablation Sec. 6.2 of the main paper, we observe that learning $k = 3$ configuration updates offers limited improvement from our L2Sep method with $k = 2$ updates. While it is beneficial to achieve a significant time improvement with a small number of updates as it simplifies the learning task and reduces training time, the optimal policy for finer-grained control should theoretically yield better performance (smaller approximation error to the optimal policy that allows updates at all separation rounds). Potential future research would involve exploring the learning of more frequent configuration updates, possibly by considering advanced reinforcement or imitation learning algorithms.

Another potential limitation of our learning method (as well as all baseline methods mentioned in the main paper) is that the mean performance tends to be lower than the interquartile mean or median due to the long solve time on a small set of outlier instances which skew the mean. A potential future research direction would involve developing techniques to identify these outlier instances, on which we use SCIP default instead of configuring with learning or heuristics methods.

A.10 Negative Social Impact.

Application of deep learning in discrete optimization may contribute to increased use of computation for training the models, which would have energy consumption and carbon emissions implications. The characterization and mitigation of these impacts remain an important area of study.