Learning to Remove Cuts in Integer Linear Programming

Pol Puigdemont [*1,2] **Stratis Skoulakis** [*1] **Grigorios G Chrysos** [3] **Volkan Cevher** [1]

Abstract

Cutting plane methods are a fundamental approach for solving integer linear programs (ILPs). In each iteration of such methods, additional linear constraints (cuts) are introduced to the constraint set with the aim of excluding the previous fractional optimal solution while not affecting the optimal integer solution. In this work, we explore a novel approach within cutting plane methods: instead of *only* adding new cuts, we also consider the *removal* of previous cuts introduced at any of the preceding iterations of the method under a learnable parametric criteria. We demonstrate that in fundamental combinatorial optimization settings such cut removal policies can lead to significant improvements over both human-based and machine learning-guided cut addition policies even when implemented with simple models.

1. Introduction

Integer linear programming (ILP) has numerous applications in engineering (Miller et al., 1960), operational research (Eiselt & Sandblom, 2000), and finance (Konno & Yamamoto, 2008). In fact, any combinatorial optimization problem can be formulated as an ILP (Conforti et al., 2014). ILP is a constrained optimization formulation in which the variables need to take integer values while satisfying some linear constraints. More precisely, an ILP with n variables and m linear constraints admits the form,

$$z^\star_{\text{int}} := \min_{x\in\mathbb{R}^n} \{c^\top x \,:\, Ax \leq b,\;\; x_j \in \mathbb{Z}\}, \tag{1}$$

where $c \in \mathbb{Z}^n$, $A \in \mathbb{Z}^{m\times n}$ and $b \in \mathbb{Z}^m$.

Despite the fact that ILPs are $\mathcal{NP}$-hard (Bixby et al., 2004), heuristics, such as the *cutting plane method* (Gomory, 1960) or branch and bound (Land & Doig, 1960a; Zarpellon et al., 2021) have proven to be extremely valuable.

The *cutting plane method*, proposed by Gomory (Gomory, 1960), is one of the most fundamental approaches for solving ILPs. The idea is to iteratively solve relaxed versions of the original problem (1) by dropping the integrality requirement, thus obtaining a linear program (LP) that is computationally tractable to solve. More precisely, by dropping the integrality constraints, we obtain the following LP:

$$z^\star_{\text{frac}} := \min_{x\in\mathbb{R}^n} \{c^\top x \,:\, Ax \leq b,\; x \in \mathbb{R}^n\}. \tag{2}$$

It is clear that $z^\star_{\text{frac}} \leq z^\star_{\text{int}}$. The cornerstone idea of the cutting plane methods is to tighten the bound $z^\star_{\text{frac}}$ by adding cutting planes (Gomory, 1960) that increase $z^\star_{\text{frac}}$ but leave $z^\star_{\text{int}}$ unchanged.

Modern solvers (Bestuzheva et al., 2023; Gurobi, 2021) use cutting planes to tighten the bounds on the linear programs (LPs) that are solved iteratively in the cutting plane method (Gomory, 1960) and the branch-and-bound method (Land & Doig, 1960b). There are multiple types of cutting planes that can increase the value of the respective LP relaxation. Namely, Gomory showed that for ILPs with m constraints, there are as much as m possible Gomory cuts (Gomory, 1960) (Apendix A.2 contains details on how to obtain Gomory cuts). Although adding all possible cutting planes would yield stronger tightening of the LP relaxation and thus faster convergence, the number of constraints would grow exponentially over the iterations, making the problem infeasible (Wesselmann & Stuhl, 2012). Consequently, all cutting plane methods select a small number of possible cuts to add at each iteration (Balas et al., 1993; Amaldi et al., 2014; Andreello et al., 2007; Tang et al., 2020; Paulus et al., 2022).

In practice, the *cut addition policy*—determining which cut to add at each iteration—plays a crucial role in the convergence properties of the method and over several handcrafted heuristics have been proposed (Balas et al., 1993; Andreello et al., 2007; Amaldi et al., 2014; Coniglio & Tieves, 2015). For instance, the SCIP solver (Bestuzheva et al., 2023) employs a weighted sum of features such as integer support and parallelism to rank the cuts.

[*]Equal contribution [1]LIONS, École Polytechnique Fédérale de Lausanne, Switzerland [2]Work developed during an exchange coming from Universitat Politècnica de Catalunya (UPC), Spain [3]Department of Electrical and Computer Engineering, University of Wisconsin-Madison, USA. Correspondence to: Pol Puigdemont <polpuigdemont@gmail.com>.

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

A more recent line of research focuses on machine learning-based cut addition policies that, after training, can adjust to the problem distributions of interest (see (Deza & Khalil, 2023) for a survey). These approaches leverage different machine learning techniques, including imitation learning (Paulus et al., 2022), reinforcement learning (Wang et al., 2023; Tang et al., 2020), and multiple instance learning (Huang et al., 2021). It is worth noting that irrespective of the specific training strategy, all previous approaches concentrate on cut addition policies that, at each iteration, include a small set of new cuts in the constraints.

1.1. Our Approach and Results

In this work, we explore cutting plane methods with learning-guided policies that do not only add cuts but also possess the capability to remove them. To the best of knowledge, this is the first work examining such cut removal policies in the context of cutting plane methods.

As already mentioned, adding multiple cutting planes at each iteration of the cut selection method is very beneficial with respect to convergence since larger sizes of the fractional polytope are removed. However, the latter leads to an exponential increase in the number of linear constraints (Wesselmann & Stuhl, 2012).

We consider the concept of *cut removals* as a strategy to mitigate the exponential growth in the number of constraints, while still capitalizing on the rapid convergence rates associated with incorporating multiple cuts (Wesselmann & Stuhl, 2012). Specifically, in each iteration of the cutting plane method, we opt to include all potential Gomory cuts in the set of linear constraints. As previously noted, this approach provides the advantage of excluding larger regions from the feasibility polytope. Subsequently, we proceed with the cut removal step, where previously introduced cuts are eliminated from the set of linear constraints. This ensures that the overall number of linear constraints increases by just one cut (or a small constant number) from iteration to iteration.

Cut Addition vs Cut Removal As mentioned earlier, our approach differs significantly from previous *cut addition* policies (Tang et al., 2020; Paulus et al., 2022). Unlike these policies, our method introduces multiple cuts at each iteration but also removes multiple cuts to achieve a balanced increase. In contrast, cut addition policies introduce a small number of new cuts in the constraints, leading to a gradual increment in the number of cuts with each iteration.

A fundamental distinction between the two approaches lies in the permanent addition of cuts. In cut addition, once a cut is incorporated into the linear constraints, it persists in all subsequent iterations of the method. However, a cut that may have been effective in the early iterations may lose its relevance as more cuts are introduced. Our cutting removal approach addresses this limitation by continually assessing the effectiveness of each cut. This allows us to replace multiple outdated cuts from early iterations with fresh cuts.

Learning to Remove Cuts As in the context of cut addition, the strategy for removing cuts plays a crucial role in the convergence properties of our method. An intuitive measure of cut quality is the difference in the LP bound with and without the cut (Coniglio & Tieves, 2015; Paulus et al., 2022). Therefore, a straightforward cut removal strategy is to eliminate cuts with the small difference in the LP bound difference. However computing the LP difference for all candidate cuts requires solving numerous LPs which leads to a significant computational overhead.

To address this challenge, we adopt an imitation learning approach, similar to that of (Paulus et al., 2022) in the case of cut addition. Specifically, we train a simple model that uses hand-crafted features of the cuts (Achterberg, 2007; Wesselmann & Stuhl, 2012) and an additional MLP layer to predict the LP bound difference for each candidate cut.

Interfacing with an implementation of the Cutting Plane method with Gomory Cuts (Gomory, 1960) we experiment on five families of MILPs by training the neural networks to predict the quality of various cuts and remove them accordingly. Our experimental evaluations indicate that our algorithm outperforms cut addition strategies.

1.2. Related Work

There is a recent line of works examining the applications of machine learning techniques in cutting plane methods. Radu et al. (2018) were the first to employ machine learning models for predicting the bound improvement of cuts in the context of semi-definite programming. In the realm of Integer Linear Programming, Tang et al. (2020) utilize reinforcement learning to devise cut addition strategies for Gomory cuts. Huang et al. (2021) develop alternative cut addition strategies through multiple instance learning. Paulus et al. (2022) employ imitation learning to create efficient cut selection policies, approximating the computationally expensive *look-ahead policy* that calculates the LP bound improvement for each candidate cut. In a related but slightly orthogonal task, Wang et al. (2023) leverage the RL framework to determine the order of cuts presented in the LP solver, minimizing the solving time of the LP. Deza & Khalil (2023) provides a great survey in recent work for learning to add cuts in the cutting plane method.

At the same time machine learning techniques have been used in the context of other approaches of ILPs such as *Column Generation* (Chi et al., 2022) and *Branch&Bound* (Chmiela et al., 2021; Gupta et al., 2020; Khalil et al., 2016; Khalil, 2016; Zarpellon et al., 2021). On the theoretical front (Balcan et al., 2022a;b; 2021) study

sample complexity and generalization bounds for cutting plane methods.

2. Background and Preliminaries

We denote with $\mathbb{R}$ the set of real numbers and with $\mathbb{Z}$ the set of integer numbers. To simplify notation, we describe the linear constraints $Ax \leq b$ with a set of hyperplanes $\mathcal{H}$. More precisely, each hyperplane $(\alpha, \beta) \in \mathcal{H}$ (where $\alpha \in \mathbb{Z}^n$ and $\beta \in \mathbb{Z}$) denotes the hyperplane $\alpha^\top x \leq \beta$.

With a slight abuse of notation, we will write that an n-dimensional vector $x \in \mathcal{H}$ if and only if $\alpha^\top x \leq \beta$ for all $(\alpha, \beta) \in \mathcal{H}$. We also denote with $(\mathcal{H}, c)$ an instance of Integer Linear Program, $\min_{x \in \mathbb{Z}^n} \{c^\top x \,:\, x \in \mathcal{H}\}$.

We denote with $x^\star_{\text{frac}} := \operatorname{argmin}_{x \in \mathbb{R}^n} \{c^\top x \,:\, x \in \mathcal{H}\}$ the optimal fractional solution of $(\mathcal{H}, c)$ and with $x^\star_{\text{int}} := \operatorname{argmin}_{x \in \mathbb{Z}^n} \{c^\top x \,:\, x \in \mathcal{H}\}$ the optimal integral solution. We remark that $x^\star_{\text{frac}}$ can be computed in polynomial time (Eiselt & Sandblom, 2007). On the contrary computing $x^\star_{\text{int}}$ is an $\mathcal{NP}$-hard problem.

Notice that $c^\top x^\star_{\text{frac}} \leq c^\top x^\star_{\text{int}}$ and thus solving the fractional problem provides a lower bound on the cost of optimal integral solution. In Definition 2.1 we introduce the notion of cutting plane (α, β) that will be crucial throughout the paper.

Definition 2.1. Let the instance $(\mathcal{H}, c)$. A hyperplane (α, β) is called *cutting plane* if and only if

$$\alpha^\top x^\star_{\text{frac}} > \beta \quad \text{and} \quad \alpha^\top x^\star_{\text{int}} \leq \beta$$

In case (α, β) is a cutting plane, then the new instance $(\mathcal{H} \cup (\alpha, \beta), c)$ admits a higher optimal fractional value but the same integral optimal value as $(\mathcal{H}, c)$. More precisely,

$$c^\top x^\star_{\text{frac}} \leq c^\top \hat{x}^\star_{\text{frac}} \leq c^\top x^\star_{\text{int}}$$

where $\hat{x}^\star_{\text{frac}} := \operatorname{argmin}_{x \in \mathbb{R}^n} \{c^\top x : x \in \mathcal{H} \cup (\alpha, \beta)\}$.

In his seminal work, Gomory (1960) showed that if $(\mathcal{H}, c)$ admits a strictly fractional solution $x^\star_{\text{frac}}$ then there exists a set of separating hyperplanes $\mathcal{C}$, i.e. each hyperplane $(\alpha, \beta) \in \mathcal{C}$ is a separating hyperplane.

Theorem 2.2. *[Gomory (1960)] Let an instance $(\mathcal{H}, c)$ with a strictly fractional solution $x^\star_{\text{frac}}$. Then there exists a set of hyperplanes $\mathcal{C}$ with $|\mathcal{C}| = |\mathcal{H}|$ such that any $(\alpha, \beta) \in \mathcal{C}$ is a cutting plane. $\mathcal{C}$ is also referred as cutpool.*

Gomory (1960) proposed the seminal *cutting plane method* (Algorithm 1) that solves ILPs by iteratively adding new cutting planes to the constraints.

The cornerstone idea of the cutting plane method is that in case the solution $x^\star_k$ at iteration k is not integral then the cutting plane (α_k, β_k) added at Step 6 will render $x^\star_k$ infeasible while keeping the optimal integer solution $x^\star_{\text{int}}$ of $(\mathcal{H}, c)$ intact. As a result, Algorithm 1 guarantees that $c^\top x^\star_{k+1} \geq c^\top x^\star_k$ until a final integral solution is reached.

Cut Selection We remark that regardless the way (α_k, β_k) is selected at Step 6, Algorithm 1 is always guaranteed to converge to the optimal integral solution. However in practice the cut selection policy plays a major role on the convergence properties of the cutting plane method. As a result, over the years various handcrafted heuristics have been proposed (Balas et al., 1993; Andreello et al., 2007; Amaldi et al., 2014; Coniglio & Tieves, 2015).

One of the most natural heurstics to select cutting planes is the one that directly *looks ahead* on the improvement of the fractional LP bound (Amaldi et al., 2014; Coniglio & Tieves, 2015; Paulus et al., 2022). As in Paulus et al. (2022) we will refer to this as the *look-ahead policy*. More precisely,

$$(\alpha_k, \beta_k) := \max_{(\alpha,\beta) \in \mathcal{C}_k} \left[\min_x \{c^\top x : x \in \mathcal{H}_k \cup \mathcal{P}_k \cup (\alpha, \beta)} \right].$$

On the positive side, *look-ahead policy* admits very favorable convergence properties to the optimal integral solution (Paulus et al., 2022). On the negative side, implementing the above cut selection policy is very time-consuming since it requires the solution of $|C_k|$ linear programs at each iteration k which creates a significant computational overhead.

Evaluating Cut Selection Policies We can utilize the number of iterations until convergence to the optimal integer solution as a measure of performance of a cutting plane method. However cutting plane methods can take a lot of iterations before converging to the the optimal integral solution, rendering the latter metric not a very practical (Tang et al., 2020; Paulus et al., 2022).

An alternative performance metric is to measure how far is the solution of method at iteration k is from the optimal integer solution. The latter is referred as *integrality gap* (Tang et al., 2020; Paulus et al., 2022) which at iteration k is defined as $g_k := c^\top x^\star_{\text{int}} - c^\top x^\star_k \geq 0$.

Algorithm 1 Cutting plane method

1: **Input:** An integer linear program $(\mathcal{H}, c)$
2: Set $\mathcal{P}_1 := \varnothing$
3: **for** $k = 1, \ldots,$ **do**
4: Compute the *fractional solution* of $(\mathcal{H} \cup \mathcal{P}_k, c)$

$$x^\star_k := \operatorname{argmin}_{x \in \mathbb{R}^n} \{c^\top x \,:\, x \in \mathcal{H} \cup \mathcal{P}_k\}$$

5: **if** $x^\star_k$ is integral **return** $x^\star_k$.
6: Compute cutpool $\mathcal{C}_k$ and pick $(\alpha_k, \beta_k) \in \mathcal{C}_k$.
7: Set $\mathcal{P}_{k+1} := \mathcal{P}_k \cup (\alpha_k, \beta_k)$ *# insert new constraint*
8: **end for**

Different MILP problems, even from the same distribution, will have different magnitudes for g_k. To compare different instances we can measure the factor by which the integrality gap is closed between the first relaxation and the current round. At iteration k of the CP method, the *integrality gap closure* (IGC) (Tang et al., 2020; Paulus et al., 2022) is defined as

$$IGC_k := \frac{g_1 - g_k}{g_1} = \frac{c^\top x_k^\star - c^\top x_1^\star}{c^\top x_{\text{int}}^\star - c^\top x_1^\star} \in [0, 1]. \quad (3)$$

3. Cut Removal Policies

The starting point of this work concerns the design of novel cutting plane methods that, at each iteration, not only add cuts but are also capable of removing cuts. As we shall see shortly, the latter offers significant advantages compared to cutting plane methods that only add cuts at each iteration.

Adding Multiple Cuts Notice that at each iteration of Algorithm 1, only one new cutting plane (α_k, β_k) is added. However, it would make perfect sense at iteration k to include the entire cutpool $\mathcal{C}_k$ of Theorem 2.2. The latter would result in an even greater increase in the objective function and, consequently, better convergence properties. However since $|\mathcal{C}_k|$ can have size up to $|\mathcal{H}| + |\mathcal{P}_k|$, the latter would lead to an exponential increase in the number of constraints, rendering the solution of the linear program at Step 4 impossible. This the reason that previous cutting plane methods have focused on adding just one or small constant number of cutting planes at each iteration (Tang et al., 2020; Huang et al., 2021; Paulus et al., 2022).

Cut Removal Our approach revolves around addressing the exponential surge resulting from the inclusion of multiple cuts in each iteration. We incorporate an additional cut-removal procedure to prevent the number of constraints from escalating exponentially. In Algorithm 2, we outline the fundamental pipeline of our approach.

Let us explain how the Algorithm 2 is able to combine both the advantages of selecting multiple cuts at each iteration while at the same time avoiding the exponential increase in the number of linear constraints.

In Step 4, Algorithm 2 computes the cutpool $\mathcal{C}_k$ (see Definition 2.1) for the problem $(H \cup \mathcal{P}_k, c)$ that includes the original constraints $\mathcal{H}$ plus the additional cuts $\mathcal{P}_k$ that have been added so far.

In Step 5, Algorithm 2 calculates the optimal solution $\hat{x}_k$ for the instance $(H \cup \mathcal{P}_k \cup \mathcal{C}_k, c)$ meaning that the algorithm completely incorporates all the cutpool $\mathcal{C}_k$. To this end we remark that due to Definition 2.1, the optimal integral solution of $(\mathcal{H}, c)$ is the same with the optimal integral solution of $(\mathcal{H} \cup \mathcal{P}_k \cup \mathcal{C}_k, c)$. Namely,

$$\operatorname*{argmin}_{x \in \mathbb{Z}^n} \{c^\top x : x \in \mathcal{H}\} = \operatorname*{argmin}_{x \in \mathbb{Z}^n} \{c^\top x : x \in \mathcal{H} \cup \mathcal{P}_k \cup \mathcal{C}_k\}.$$

Algorithm 2 Cutting Plane method with Cut Removal

1: **Input:** An integer linear program $(\mathcal{H}, c)$
2: Set $\mathcal{P}_1 := \varnothing$
3: **for** $k = 1, \ldots,$ **do**
4: Compute the cutpool $\mathcal{C}_k$ for $(\mathcal{H} \cup \mathcal{P}_k, c)$.
5: Compute the *fractional solution* of $(\mathcal{H} \cup \mathcal{P}_k \cup \mathcal{C}_k, c)$
$x_k^\star := \operatorname{argmin}_{x \in \mathbb{R}^n} \{c^\top x : x \in \mathcal{H} \cup \mathcal{P}_k \cup \mathcal{C}_k\}$
 # include all possible cuts
6: **if** $x_k^\star$ is integral **return** $x_k^\star$.
7: Select a subset $\hat{P}_{k+1} \subseteq \mathcal{P}_k \cup \mathcal{C}_k$ with $|\hat{P}_{k+1}| = k+1$.
 # removal of cuts
8: $\mathcal{P}_{k+1} := \hat{P}_{k+1} \cup \{c^\top x \geq \lceil c^\top x_k^\star \rceil\}$
 # new constraints for the next iteration
9: **end for**

The idea behind this step is to achieve a substantial improvement in the objective function compared to the strategy of introducing just one cut $(\alpha, \beta) \in \mathcal{C}_k$. In other words,

$$\min_{x \in \mathbb{R}^n} \{c^\top x : x \in \mathcal{H} \cup \mathcal{P}_k \cup \mathcal{C}_k\} \geq \min_{x \in \mathbb{R}^n} \{c^\top x : x \in \mathcal{H} \cup \mathcal{P}_k \cup (\alpha, \beta)\}$$

where the last inequality comes from the fact that the left problems admit a superset of constraints with respect to the right one.

In Step 6 of Algorithm 2, the cut removal process is executed to prevent an exponential rise in the number of constraints. To be more specific, only $k+1$ cuts from the set of previous cuts, $\mathcal{P}_k \cup \mathcal{C}_k$, are kept. This ensures that only one additional cut is introduced in each iteration, preserving the tractability of the solution obtained in Step 4 of the linear programming formulation.

Step 7 plays a crucial role in ensuring the algorithm's monotonic improvement. Note that $\mathcal{P}_k$ may not necessarily be a subset of $\hat{\mathcal{P}}_k$, implying that a cut introduced in a previous iteration can be removed in a subsequent iteration if it becomes uninformative. The positive aspect of this approach lies in the ability to replace inactive cuts from earlier iterations with more valuable ones. On the downside, as $\mathcal{P}_k$ is not guaranteed to be a subset of $\hat{\mathcal{P}}_{k+1}$, there is a potential for the objective function to decrease. To mitigate this, a special constraint $c^\top x \geq \lceil c^\top x_k^\star \rceil$ is included. This constraint ensures that the fractional value of the problem $(\mathcal{H} \cup \mathcal{P}_{k+1}, c)$ can only increase with respect to the fractional value of the problem $(\mathcal{H} \cup \mathcal{P}_k, c)$.

Remark 3.1. The additional special constraint $c^\top x \geq \lceil c^\top x_k^\star \rceil$ does not affect the optimal integral solution. The latter is formally stated and proven in Corollary 3.2.

Corollary 3.2. *It holds that* $\operatorname{argmin}_{x \in \mathbb{Z}^n} \{c^\top x : x \in \mathcal{H} \cup \mathcal{P}_{k+1}\} = \operatorname{argmin}_{x \in \mathbb{Z}^n} \{c^\top x : x \in \mathcal{H}\}$

Proof. The reason is that the following holds

$$\begin{aligned}\min_{x\in\mathbb{Z}^n}\{c^\top x : x\in\mathcal{H}\} &= \min_{x\in\mathbb{Z}^n}\{c^\top x : x\in\mathcal{H}\cup\mathcal{P}_k\}\\ &= \min_{x\in\mathbb{Z}^n}\{c^\top x : x\in\mathcal{H}\cup\hat{\mathcal{P}}_{k+1}\}\\ &\geq \underbrace{\min_{x\in\mathbb{R}^n}\{c^\top x : x\in\mathcal{H}\cup\hat{\mathcal{P}}_{k+1}\}}_{c^\top x_k^\star}\end{aligned}$$

The first equality holds inductively and the second comes from the fact that $\hat{\mathcal{P}}_{k+1}\subseteq\mathcal{H}\cup\mathcal{P}_k$. Notice that since $c\in\mathbb{Z}^n$, $\min_{x\in\mathbb{Z}^n}\{c^\top x : x\in\mathcal{H}\}\geq\lceil c^\top x_k^\star\rceil$. Thus $\min_{x\in\mathbb{Z}^n}\{c^\top x : x\in\mathcal{H}\cup\hat{\mathcal{P}}_{k+1}\}\geq\lceil c^\top x_k^\star\rceil$. As a result, adding the additional constraint $c^\top x\geq\lceil c^\top x_k^\star\rceil$ does not affect the optimal integral solution. □

3.1. Learning to Remove Cuts

In this section, we will examine what constitutes a reasonable cut removal criterion for Step 6 of Algorithm 2 that can contribute to fast convergence properties. Motivated by the *look-ahead* policy presented in Section 2, the most intuitive cut-removal criterion is the one that maximizes the objective function. Namely:

$$\hat{P}_{k+1} := \text{argmax}_{\mathcal{P}\subseteq\mathcal{P}_k\cup\mathcal{C}_k}\left[c^\top x : x\in\mathcal{H}\cup\mathcal{P}\right].$$

Unfortunately, there is no efficient algorithm for solving the problem above. Consequently, the most natural approach is to consider the *greedy look-ahead criterion* where we keep the $k+1$ cuts of $\mathcal{P}_k\cup\mathcal{C}_k$ that lead to highest bounds in the objective function and remove the rest. More precisely, we associate each cutting plane $(\alpha,\beta)\in\mathcal{P}_k\cup\mathcal{C}_k$ with the following score,

$$\begin{aligned}\text{SC}(\alpha,\beta) &= \min_{x\in\mathbb{R}^n}\{c^\top x \ : \ x\in\mathcal{H}\cup\mathcal{P}_k\cup\mathcal{C}_k\} \qquad (4)\\ &- \min_{x\in\mathbb{R}^n}\{c^\top x \ : \ x\in\mathcal{H}\cup\mathcal{P}_k\cup\mathcal{C}_k/(\alpha,\beta)\},\end{aligned}$$

which measures the difference of the fractional LP value once $(\alpha,\beta)\in\mathcal{P}_k\cup\mathcal{C}_k$ is removed. Then $\hat{\mathcal{P}}_{k+1}$ is defined as the $k+1$ cuts of $\mathcal{P}_k\cup\mathcal{C}_k$ with the highest score.

Parametrizing Cut Removal Unfortunately, similar to the case of single-cut addition, the cut removal policy of Eq. (4) is highly computationally demanding to implement since it requires solving numerous linear programs. To circumvent this additional computational overhead, we will approximate the scores of Eq. (4) through an adequately parametrized model $\pi_\theta(\cdot)$. More precisely, for each cut $(\alpha,\beta)\in\mathcal{P}_k\cup\mathcal{C}_k$ we compute the model $\pi_\theta(\cdot)$ to compute the scores

$$\text{SC}(\alpha,\beta)=\pi_\theta(\mathcal{H},\mathcal{P}_k,\mathcal{C}_k,(\alpha,\beta)).$$

More precisely, we first convert each cut (α,β) into a 14-dimensional feature vector. Due to the fact that the coordinate of the feature vectors involve also the optimal fractional solution $x_k^\star:=\min\{c^\top x : x\in H\cup\mathcal{P}_k\cup\mathcal{C}_k\}$ these feature embeddings naturally encode the overall structure of the linear constraints. The details of feature vectors can be found in Appendix B.3. The encoded state is then fed to a Multi Layer Perceptron (MLP) with a sigmoid (σ) activation.

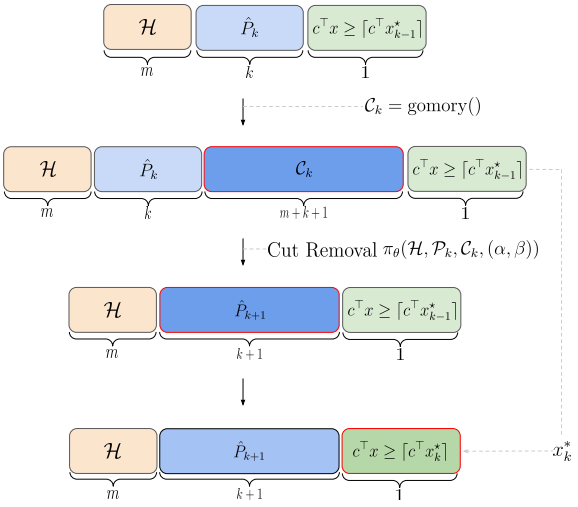

Figure 1. Visual representation of Algorithm 2.

A summary of our approach is illustrated in Figure 1.

Remark 3.3. More complicated models such as graph neural networks over a graph encoding of the state as in Paulus et al. (2022); Gasse et al. (2019) could also be used. Although graph architectures draw an interesting line of improvements they are outside the scope of this study.

3.2. Training the Model

The goal of training process is to approximately select the parameters θ so as to predict the scores of Eq. (4). In other words select θ such that

$$\begin{aligned}&\pi_\theta(\mathcal{H},\mathcal{P}_k,\mathcal{C}_k,(\alpha,\beta))\simeq\min_x\{c^\top x \ : \ x\in\mathcal{H}\cup\mathcal{P}_k\cup\mathcal{C}_k\}\\ &-\min_x\{c^\top x \ : \ x\in\mathcal{H}\cup\mathcal{P}_k\cup\mathcal{C}_k/\{\alpha^\top x\leq\beta\}\}.\end{aligned}$$

We note that Paulus et al. (2022) adopt a similar imitation learning approach to approximate the scores of the *look-ahead policy* presented in Section 2.

Dataset Generation In order to generate our training data we use trajectories produced by the cutting plane method with the *look-ahead policy* for various classes of ILPs of such as Packing, Bin Packing, Set Cover, Max Cut and Production Planning.

For each intermediate ILP produced at iteration k, denoted as $(\mathcal{H}\cup\mathcal{P}_k,c)$, we utilize the corresponding cut pool $\mathcal{C}_k$ and the previously added cuts $\mathcal{P}_k$ to generate the following data set. For every cutting plane $(\alpha,\beta)\in\mathcal{C}_k\cup\mathcal{P}_k$, a new

data point is created with a value equal to the *normalized LP improvement*,

$$\frac{c^\top x^\star_{k\cup(\alpha,\beta)} - c^\top x^\star_k}{c^\top x^\star_k} > 0$$

where $x^\star_k := \text{argmin}_x\{c^\top x : x \in \mathcal{H}\cup\mathcal{P}_k\}$ and $x^\star_{k\cup(\alpha,\beta)} := \text{argmin}_x\{c^\top x : x \in \mathcal{P}\cup\mathcal{C}_k\cup(\alpha,\beta)\}$. The reason that we use the normalized LP improvement over the actual LP improvement $c^\top x^\star_{k\cup(\alpha,\beta)} - c^\top x^\star_k$ is to get target values that are not as dependent on the specific instance of the problem family.

Once all training data are generated, we solve a regression problem with quadratic loss to choose the parameters θ.

4. Experiments

We divide our experiments in two main parts. The first one focuses on evaluating the performance of cut removal acting against multiple benchmark policies by rolling them out on synthetic test MILP instances for each of the problem families in a controlled environment. Next, we investigate how well do our trained models generalize to larger instances.

4.1. Evaluation Setup

As in Tang et al. (2020); Paulus et al. (2022), we assess the performance of the various cutting plane methods by considering the improvement over the Integral Gap Closure value (IGC) (see Section 2) with respect to the number of iterations. This evaluation metric offers a robust estimate of the actual running time of the cutting plane methods, as the primary bottleneck in running time arises from solving linear programs (LPs). Simultaneously, this metric provides a cleaner benchmark since running time significantly depends on factors such as implementation, the choice of backbone solver, and available compute resources.

Paulus et al. (2022) employ imitation learning to train a cut addition policy, mimicking the behavior of the *look-ahead policy* discussed in Section 2. Their study demonstrates that the imitation learning approach outperforms the reinforcement learning method proposed by Tang et al. (2020). However, both approaches are surpassed by the *look-ahead policy*, which involves solving numerous LPs at each iteration. Unfortunately, neither Paulus et al. (2022) nor Tang et al. (2020) provide a public implementation of their code. To ensure a fair comparison, we consider the *look-ahead policy*, which outperforms both previous approaches, and an in-house implementation of the *Neural Cut* method proposed by Paulus et al. (2022) using the same feature encoding as in our model (see Appendix B.3). We also include several human-crafted heuristics such as the min similar, max normalized violation, max violation, lexicographical and random (see Appendix B.2 for the definitions).

Remark 4.1. Paulus et al. (2022) utilize the SCIP solver, which incorporates instance pre-solving and various types of cuts (Achterberg, 2007). To ensure a fair benchmarking environment focused solely on cut decisions, we benchmark on an implementation of the Cutting Plane method from scratch. The implementation considers only Gomory Cuts without any pre-solving modes.

In order to stress-test our environment and compute the optimal solution required to obtain the IGC metrics we use the SCIP solver (Bestuzheva et al., 2023).

Our models are trained with (Robbins & Monro, 1951) using Pytorch (Paszke et al., 2019) for the implementation.

4.2. Datasets

We experiment with five families of MILPs: packing, bin packing, max cut, production planning (as in Paulus et al. (2022); Tang et al. (2020)) and set cover. For each family, instances are generated randomly. Packing, bin packing, max cut and production planning are generated under the random formulations used in Tang et al. (2020); Paulus et al. (2022). For set cover we suggest our own probabilistic formulation. Details on the generation of the instances can be found in the Appendix B.1.

4.3. In Distribution Evaluation

The core experiments in this work aim to respond the following question: Can our cut removal acting algorithm (after training) surpass the *look-ahead*, Paulus et al. (2022) and other cut addition baselines on unseen instances? We answer the previous question positively on the five different ILP benchmarks.

ILP Benchmarks We generate a total of 3000 instances for each of the five problem families. Regarding the dimensions, the small and medium instances suggested in Tang et al. (2020) were too small for our comparisons and they were being solved after too few iterations to extract differences. For this reason we only consider large instances ($\sim$100 variables, 100 constraints) as done in Paulus et al. (2022).

For each problem family, we use 2000 instances for training, 500 instances for validation and 500 instances for testing as done in Paulus et al. (2022). We collect trajectories of the *look-ahead* policy for the train and validation instances. We train these parametric models π_θ with these trajectories to predict the bound improvement as previously specified in Section 3. Finally, the various cutting plane methods are compared in terms of their performance on the test instances.

The dimensions (variables, constraints) for training, validation and testing are the same for each problem. The

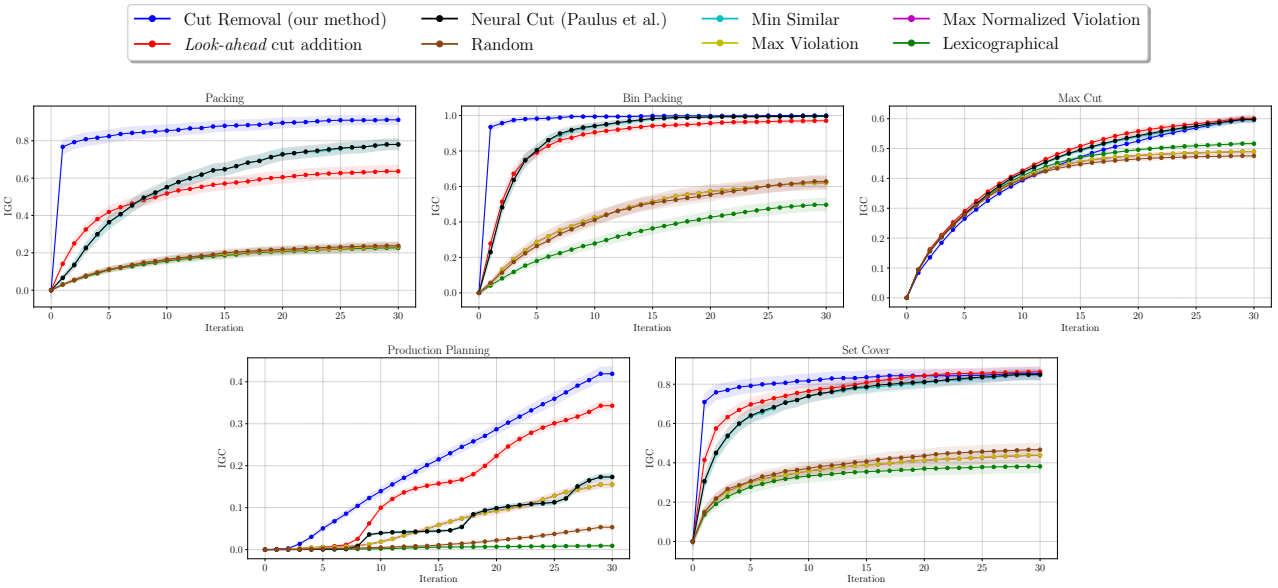

Figure 2. **Mean IGC for the benchmark instances:** We report the mean Integral Gap Closed (IGC) that measures quality of the solution with respect to the optimal integer solution (see Section 2 for details), a policy achieving larger IGC values with fewer iterations is better. The highlighted region represents the variance. Our cut removal algorithm outperforms or matches all cut addition methods in all benchmarks. For cut addition policies the *look-ahead* outperforms all of the others except in Packing where Neural Cut, this behavior matches the results showcased in the equivalent benchmark of Paulus et al. (2022).

specifications for each problem can be found in Appendix B.1.

Test Evaluation Metrics In order to assess which method is best we compare the mean IGC after each iteration across all instances as done in Paulus et al. (2022); Tang et al. (2020). Larger IGC with fewer iterations denotes better performance. The IGC is 0 at the start and 1 in case of convergence to the optimal intergral solution.

We don not compare execution times because the focus of our experiments relies in evaluating how large is the improvement at each round as in Paulus et al. (2022); Tang et al. (2020). A time detailed evaluation would depend on the LP solver, pre-solving of the instances, ordering of the constraints (Wang et al., 2023) and other factors which are outside the scope of this evaluation. Our experimental findings are presented in Figure 2.

Our Experimental Results After training on the test instances, we select the best π_θ according to the validation loss. Our cut removal algorithm outperforms all cut addition policies in packing, bin-packing, production planning and set cover families. For packing and production planning the mean IGC of our method outpeforms significantly the comparing methods. For bin-packing and set cover the mean IGC is significantly larger in the first half of the iterations. We observe that this match in performance occurs in IGC values close to optimal pointing that in mean the algorithms are able to reach the optimal integral solution before iteration 30.

For max cut cut removal acting does not outperform the expert and neural cut addition methods. Nevertheless, the differences in performance are very small, for instance, a random policy performs as well as the best policies in the first third of the iterations. This behaviour is consistent with the findings in Paulus et al. (2022).

4.4. Generalization on Larger Instances

After having studied the performance that cut removal acting yields we aim to evaluate how well do our trained policies generalize to larger instances. We aim to answer the following questions: Can our cut removal acting algorithm with a model trained on smaller instances generalize for bigger instances? How does it compare with the previous benchmarks?

We remark that generalization to instances of lager size is a very desirable property since the annotation of the data

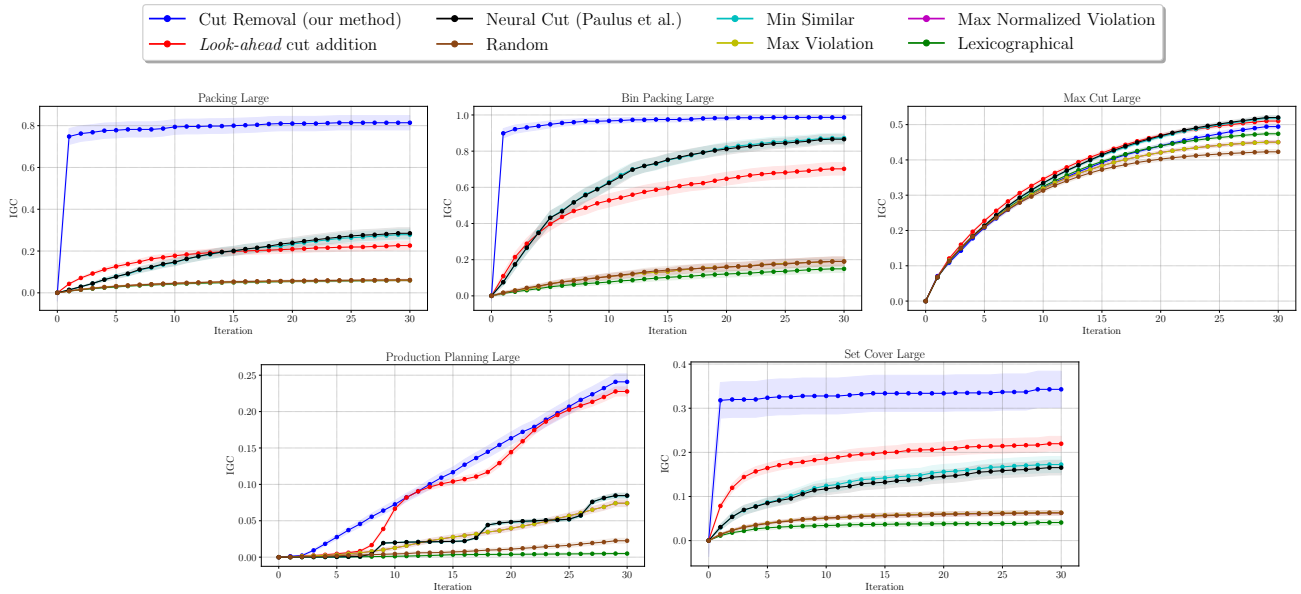

Figure 3. **Mean IGC for out-of-distribution instances** Our cut removal algorithm outperforms or matches all cut addition methods in all benchmarks. Our method shows a much stronger generalization ability onto the larger instances. The scaling improvement is especially accentuated in packing, bin packing and set cover.

becomes harder as the size increases.

ILP Benchmarks We use the same instances as in the first experiment (see subsection 4.3) to train the models. We run the evaluation with 500 fresh instances of larger problems $\sim 50\%$ larger. Details on the specific dimensions are contained in Appendix B.1.

Test Evaluation Metrics In order to asses which method is best we use the IGC as in the previous experiment.

Our Experimental Results The model π_θ is pretrained according to Section 4.3. Figure 3 shows the test IGC for cut removal acting against the benchmark policies for packing, bin-packing, max cut, production planning and set cover.

Even after training π_θ on smaller instances than the ones used for testing, our cut removal algorithm outperforms all cut addition policies in packing, bin-packing, production planning and set cover families. For packing, bin-packing and set cover the margin betweeen cut removal acting and the *look-ahead* policy significantly grows when compared to the margins in training and testing in the medium instances. This shows that the cut removal algorithm scales better than the cut addition benchmarks for this families of instances. For production planning the margin is slightly decreased when compared to the results in the previous experiment as the *look-ahead* policy matches the performance of the cut removal method in the last third of the iterations.

For the max cut family the performance is the same as in the previous experiment. Again, the cut removal acting does not outperform all cut addition methods but the differences in performance are very small and the random policy performs almost as well as the best policies in the first third of the iterations.

5. Conclusion

Cutting plane methods play a crucial role in solving Integer Linear Programs. Recent works have employed machine learning techniques to design cut addition methods that, after training, are able to ensure fast convergence to the optimal integral solution (Tang et al., 2020; Paulus et al., 2022). In this work, we propose a novel cutting plane method that, at each iteration, introduces multiple cutting planes which are subsequently removed based on the output of a well-trained machine learning model. Our experimental evaluations demonstrate that our method outperforms both human-based heuristics and more recent machine learning-based approaches for cut addition.

Several intriguing research directions emerge for future work. The first involves the use of models capable of cap-

turing the joint combinatorial structure of sets of cuts. The second focuses on designing cutting plane methods that do not maintain a constant increase in the number of linear constraints from iteration to iteration, but rather select the number of linear constraints to add based on an appropriate model. We are confident these approaches have the potential to significantly enhance the convergence properties of modern cutting plane methods.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

Authors acknowledge the constructive feedback of reviewers and the work of ICML'24 program and area chairs. ARO - Research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-24-1-0048. Hasler AI - This work was supported by Hasler Foundation Program: Hasler Responsible AI (project number 21043). SNF project – Deep Optimisation - This work was supported by the Swiss National Science Foundation (SNSF) under grant number 200021_205011. Stratis Skoulakis is supported by Innosuisse through an Innovation project (contract agreement 100.960 IP-ICT)

References

Achterberg, T. *Constraint Integer Programming*. PhD thesis, 2007.

Amaldi, E., Coniglio, S., and Gualandi, S. Coordinated cutting plane generation via multi-objective separation. *Mathematical Programming*, 143, 02 2014. doi: 10.1007/s10107-012-0596-x.

Andreello, G., Caprara, A., and Fischetti, M. Embedding 0,1/2-cuts in a braneh-and-cut framework: A computational study. *INFORMS Journal on Computing*, 19: 229–238, 03 2007. doi: 10.1287/ijoc.l050.0162.

Balas, E., Ceria, S., and Cornuéjols, G. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Math. Program.*, 58(3):295–324, feb 1993. ISSN 0025-5610.

Balcan, M., Prasad, S., Sandholm, T., and Vitercik, E. Sample complexity of tree search configuration: Cutting planes and beyond. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 4015–4027, 2021.

Balcan, M., Prasad, S., Sandholm, T., and Vitercik, E. Structural analysis of branch-and-cut and the learnability of gomory mixed integer cuts. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022a.

Balcan, M., Prasad, S., Sandholm, T., and Vitercik, E. Structural analysis of branch-and-cut and the learnability of gomory mixed integer cuts. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022b.

Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S. J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M. E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D., and Witzig, J. Enabling research through the scip optimization suite 8.0. *ACM Trans. Math. Softw.*, 49(2), jun 2023. ISSN 0098-3500. doi: 10.1145/3585516. URL https://doi.org/10.1145/3585516.

Bixby, R. E., Fenelon, M., Gu, Z., Rothberg, E., and Wunderling, R. Mixed-integer programming: A progress report. In *The sharpest cut: the impact of Manfred Padberg and his work*, pp. 309–325. SIAM, 2004.

Chi, C., Aboussalah, A. M., Khalil, E. B., Wang, J., and Sherkat-Masoumi, Z. A deep reinforcement learning framework for column generation. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

Chmiela, A., Khalil, E. B., Gleixner, A. M., Lodi, A., and Pokutta, S. Learning to schedule heuristics in branch and bound. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 24235–24246, 2021.

Conforti, M., Cornuéjols, G., and Zambelli, G. *Integer programming*, volume 271. Springer, 2014.

Coniglio, S. and Tieves, M. On the generation of cutting planes which maximize the bound improvement. volume 9125, pp. 97–109, 06 2015. ISBN 978-3-319-20085-9. doi: 10.1007/978-3-319-20086-6_8.

Dey, S. S. and Molinaro, M. Theoretical challenges towards cutting-plane selection. *Mathematical Programming*, 170(1):237–266, 2018. doi: 10.1007/s10107-018-1302-4. URL https://doi.org/10.1007/s10107-018-1302-4.

Deza, A. and Khalil, E. B. Machine learning for cutting planes in integer programming: A survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI-2023. International Joint Conferences on Artificial Intelligence Organization, August 2023. doi: 10.24963/ijcai.2023/739. URL http://dx.doi.org/10.24963/IJCAI.2023/739.

Eiselt, H. and Sandblom, C.-L. *Integer Programming and Network Models*, pp. 457–477. 01 2000. ISBN 978-3-642-08651-9. doi: 10.1007/978-3-662-04197-0_20.

Eiselt, H. A. and Sandblom, C. L. *Linear Programming and its Applications*. Number 978-3-540-73671-4 in Springer Books. Springer, June 2007. ISBN ARRAY(0x5513f500). doi: 10.1007/978-3-540-73671-4. URL https://ideas.repec.org/b/spr/sprbok/978-3-540-73671-4.html.

Gasse, M., Chetelat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/d14c2267d848abeb81fd590f371d39bd-Paper.pdf.

Gomory, R. An algorithm for the mixed integer problem. Technical report, RAND CORP SANTA MONICA CA, 1960.

Gupta, P., Gasse, M., Khalil, E. B., Mudigonda, P. K., Lodi, A., and Bengio, Y. Hybrid models for learning to branch. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Gurobi. Gurobi solver, https://www.gurobi.com/, 2021.

Huang, Z., Wang, K., Liu, F., ling Zhen, H., Zhang, W., Yuan, M., Hao, J., Yu, Y., and Wang, J. Learning to select cuts for efficient mixed-integer programming, 2021.

Khalil, E. B. Machine learning for integer programming. In Kambhampati, S. (ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 4004–4005. IJCAI/AAAI Press, 2016. URL http://www.ijcai.org/Abstract/16/576.

Khalil, E. B., Bodic, P. L., Song, L., Nemhauser, G. L., and Dilkina, B. Learning to branch in mixed integer programming. In Schuurmans, D. and Wellman, M. P. (eds.), *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pp. 724–731. AAAI Press, 2016. doi: 10.1609/AAAI.V30I1.10080. URL https://doi.org/10.1609/aaai.v30i1.10080.

Konno, H. and Yamamoto, R. *Applications of Integer Programming to Financial Optimization*, volume 18, pp. 25–48. 01 2008. doi: 10.1007/978-0-387-76682-9_2.

Land, A. H. and Doig, A. G. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960a. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1910129.

Land, A. H. and Doig, A. G. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960b. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1910129.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, oct 1960. ISSN 0004-5411.

Nair, V., Bartunov, S., Gimeno, F., von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., Addanki, R., Hapuarachchi, T., Keck, T., Keeling, J., Kohli, P., Ktena, I., Li, Y., Vinyals, O., and Zwols, Y. Solving mixed integer programs using neural networks, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Paulus, M. B., Zarpellon, G., Krause, A., Charlin, L., and Maddison, C. J. Learning to cut by looking ahead: Cutting plane selection via imitation learning, 2022.

Radu, B.-L., Bonami, P., Misener, R., and Tramontani, A. Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks., 2018. http://www.optimization-online.org/DB_HTML/2018/11/6943.html.

Robbins, H. and Monro, S. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951. doi: 10.1214/aoms/1177729586. URL https://doi.org/10.1214/aoms/1177729586.

Tang, Y., Agrawal, S., and Faenza, Y. Reinforcement learning for integer programming: Learning to cut, 2020.

Wang, Z., Li, X., Wang, J., Kuang, Y., Yuan, M., Zeng, J., Zhang, Y., and Wu, F. Learning cut selection for mixed-integer linear programming via hierarchical sequence model, 2023.

Wesselmann, F. and Stuhl, U. Implementing cutting plane management and selection techniques. Technical report, Technical report, University of Paderborn, 2012.

Zarpellon, G., Jo, J., Lodi, A., and Bengio, Y. Parameterizing branch-and-bound search trees to learn branching policies. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 3931–3939. AAAI Press, 2021.

Contents of the Appendix

We describe the contents of the supplementary material below:

- In Appendix A, we provide additional details on how the randomized ILP instances are generated and the description on how Gomory Cuts are obtained.
- In Appendix B, we present additional implementation details including the dataset dimensions, definitions for both the baselines and the feature extraction step and the specification for the training hyperparameters.
- In Appendix C, we evaluate the performance of our method in interaction with an ILP solver. We include performance results for both end-to-end and isolated cutting plane stage solving.
- In Appendix D, we discuss runtime considerations and provide a side-by-side comparison on that axis.
- In Appendix E, we explore how the cut quality is distributed in cutpools for each of the different families. We gain some insights on the experimental results in Section 4

A. ILP generation and primer on Gomory Cuts

A.1. Integer Programming Domains

We used instances from five integer programming domains. The first four: (i) packing, (ii) bin packing, (iii) maximum cut and (iv) production planning are the ones first suggested by Tang et al. (2020) and also used by Paulus et al. (2022). The exact mathematical formulation for the four first families is given in Tang et al. (2020). We extend the benchmarks with instances from the set cover family.

For set cover, we generate those instances probabilistically. Starting with m elements and n subsets, we add each element to a subset with probability p. After the full iteration, we achieve feasibility by ensuring: (i) that no subset is empty by adding a random element to empty subsets, (ii) that all elements are included in at least one subset by adding non-included elements to a random set. Let $E = \{e_1, e_2, \ldots, e_n\}$ be a set of n elements. Let $S_1, S_2, \ldots, S_m$ be subsets of E with associated costs $c_1, c_2, \ldots, c_m$. Let X_i be a random variable associated with each subset S_i. $X_i = 1$ if S_i is in the solution, and 0 otherwise. The ILP formulation is as follows:

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{m} c_i X_i \\ \text{subject to} \quad & \forall e \in E, \sum_{i:e \in S_i} X_i \geq 1, \\ & \forall X_i, X_i \in \{0, 1\}. \end{aligned}$$

The constraints in (1) ensure that every element is present in at least one of the chosen subsets. The constraints in (2) indicate that every subset is either chosen or not. The objective function chooses a feasible solution with the minimum cost. We use $p = 0.2$ and $\boldsymbol{c} = \mathbf{1}$ for our experiments.

A.2. Generating Gomory Cuts

When an LP is tackled using a simplex algorithm, the primary step involves converting the original LP into standard form. This entails transforming all inequalities into equalities through the introduction of slack variables:

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax + Is = b, \\ \text{and} \quad & x \geq 0, \quad s \geq 0, \end{aligned}$$

where I denotes an identity matrix, and s represents the set of slack variables. The simplex method iterates on the tableau formed by $[A, I]$, b, and c. Upon convergence, the simplex method furnishes a final optimal tableau composed by a constraint

matrix L with a constraint vector v. A Gomory cut in the standard form space is generated by utilizing the row of the tableau corresponding to a fractional variable in the optimal solution $x^\star$. For each fractional element $x_i^\star$ of $x^\star$ we can generate a Gomory cut

$$(-L_i + \lfloor L_i \rfloor)^T x \leq -v_i + \lfloor v_i \rfloor, \tag{5}$$

where L_i is the i-th row of the matrix L and $\lfloor \cdot \rfloor$ means component-wise rounding down. We can decompose the generated cuts cutting plane of the following form:

$$e^T x + r^T s \leq d \tag{6}$$

where $e, x \in \mathbb{R}^n$, $r, s \in \mathbb{R}^m$, and $d \in \mathbb{R}$. Despite the presence of slack variables, they can be eliminated by multiplying both sides of the linear constraints in (6) by r:

$$r^T A x + r^T s = r^T b \tag{7}$$

and subtracting the new cutting plane (7) from the equation above. This results in an equivalent $\alpha \leq \beta$ cutting plane:

$$(e^T - r^T A)x \leq d - r^T b \tag{8}$$

This cutting plane exclusively involves variables within the original variable space. Slack variables do not provide additional information about the polytope and operations for the encoding described in B.3 are defined in the original space.

B. Implementation Details

B.1. Dataset Dimensions

Tang et al. (2020) consider three different sizes (small, medium large) for each domain. In Paulus et al. (2022) the authors discard small and medium sizes because they are solved at presolving time or after adding a few number of cuts. Although we do not use pre-solving in our study out method has also shown to converge too fast for small and medium instances in (Tang et al., 2020).

For the in-distribution experiment we generate 2000 train, 500 validation, 500 test instances of the following dimensions:

- Packing: 50 variables, 50 (resource) constraints.
- Bin Packing: 50 variables, 50 (resource) constraints + 50 binary constraints.
- Max Cut: $|V| = 9, |E| = 25$.
- Production Planning: $T = 10$.
- Set Cover: $|E| = 35, |S| = 35$.

Note that for each of the training and validation instances a trajectory of the *look-ahead* generates up to 30 cutpools of size around $\frac{m+30}{2}$ where m is the number of constraints which leads to approximately $2000 \cdot 30 \cdot \frac{m+30}{2}$ training datapoints per family. For example, on packing $m = 50$ this is approximately $24 \cdot 10^5$ datapoints.

For the generalization into larger instances experiment we generate 500 test instances of the following dimensions:

- Packing: 100 variables, 100 (resource) constraints.
- Bin Packing: 100 variables, 100 (resource) constraints + 100 binary constraints.
- Max Cut: $|V| = 14, |E| = 40$.
- Production Planning: $T = 15$.
- Set Cover: $|E| = 50, |S| = 50$.

Table 1. Designed cut features for a generated cut (α_k, β_k). c is objective coefficient vector and $x^\star$ is the latest LP solution.

Feature	Description	Number
Cut Coefficients	Mean, Max, Min, Std of (α_k, β_k)	4
Objective Coefficients	Mean, Max, Min, Std of c	4
Parallelism	Parallelism between the objective and the cut $\frac{(\alpha_k,\beta_k)^T c}{\lvert c\rvert\lvert(\alpha_k,\beta_k)\rvert}$	1
Efficacy	Euclidean distance of the cut hyperplane to $x^\star$	1
Support	Proportion of non-zero coefficients of (α_k, β_k)	1
Integral Support	Proportion of non-zero coefficients with respect to integer variables of (α_k, β_k)	1
Normalized Violation	Violation of the cut to the current LP solution $\max\{0, \frac{\alpha_k^T x^\star - \beta_k}{\lvert\beta_k\rvert}\}$	1
Latest Cutpool	Wheter $(\alpha_k, \beta_k) \in \mathcal{C}_k$ or not	1

B.2. Baselines

Consider $\mathcal{C}$ to be a cutpool. The baseline heuristics that we use are defined as follows:

- Random: Choose $(\alpha_k, \beta_k) \in \mathcal{C}$ uniformly at random.
- Max Violation (MV): Let $x^\star$ be the basic feasible solution of the current LP. MV selects the cut corresponding to the maximum fractional component, this is the cut corresponding to the index $i_s = \text{argmax}_i\{|x_i^\star - \text{round}(x_i^\star)|\}$.
- Max Normalized Violation (MNV). Recall that L denotes the optimal tableau returned by the simplex algorithm. Let L_i be the ith row of L. Then, MNV selects the cut corresponding to index $i_s = \text{argmax}_i\{|x_i^\star - \text{round}(x_i^\star)|/\|L_i\|\}$.
- Lexicographic: Add the first cut with fractional index $i_s = \text{argmin}_{x_i^* \text{is fractional}}\{i\}$.
- Min Similar: Takes the cut $\text{argmin}_{(\alpha_k,\beta_k)\in\mathcal{C}}\{(\alpha_k, \beta_k)^T c\}$ where c is the objective coefficient vector.

Wesselmann & Stuhl (2012) is a useful resource for a more detailed description of heuristic cut selection rules.

B.3. Feature Encoding

We design 14 cut features to represent the state for the cut selection task. The first 13 follow Wang et al. (2023); Huang et al. (2021); Wesselmann & Stuhl (2012); Achterberg (2007); Dey & Molinaro (2018). The 14-th is a binary variable indicating if a cut belongs to the latest cutpool. Table 1 provides a description of such features.

B.4. Training Hyperparameters

We trained our models with SGD with a lr of $5 \cdot 10^{-3}$ for 50 epochs using a batch size of 10^4 with a patience parameter of 5.

C. Interfacing with an ILP Solver

As remarked in the experiments section 4.1 our method is tested in a clean and isolated environment using an implementation of the cutting plane method from scratch. Nevertheless, we acknowledge the interest of evaluating our approach perform inside of an ILP Solver. As a result, we have incorporated our cut-removal approach in the SCIP (Achterberg, 2007) solver which also enables solving larger instances as well as using other types of cuts implemented natively (Mixed Integer Gomory cuts, Strong Chvátal-Gomory cuts, Complemented Mixed Integer cuts and Implied Bound cuts).

In particular, we have developed an implementation of our method in the SCIP solver through the PySCIPOpt python interface and used it on the "Neural Network Verification" (Nair et al., 2020) dataset instances. More precisely, we evaluated the performance of the Branch-and-Cut mode of SCIP with the vanilla cut-addition policy (B&C-Cut_Addition) with the performance of Branch-and-Cut mode with our cut-removal approach (B&C-Cut_Removal).

C.1. End-to-end Performance Comparison

In Table 2 we present the percentage improvement of the solution found by B&C-Cut_Removal with respect to the solution found by B&C-Cut_Addition in 26 randomly selected instances with the node limit set to 100. Our experiments reveal that B&C-Cut_Removal finds on average a 35% better solution than B&C-Cut_Addition. We also observe that B&C-Cut_Removal is able to find a better solution on 88.46% of instances. In the remainder 11.54% of instances both methods reach the same solution.

Table 2. Improvement (%) for Each Instance (id)

Instance	1317	1891	1941	1987	2229	2891	2959	321	3736	3853	3964	4173	4329
Imp. (%)	27.87	4.69	0.00	125.80	46.38	76.00	22.55	17.32	22.75	74.23	20.69	0.00	3.52
Instance	4743	495	5119	5424	5463	5757	5833	6392	6481	7064	8509	8627	8630
Imp. (%)	0.00	47.85	25.24	33.36	21.32	52.88	195.50	34.01	19.20	12.12	11.36	12.87	12.91

C.2. Isolated Performance Comparison

Branching algorithms can be interpreted by the tree they describe. The starting problem formulation relies on the top node of a tree. After making a branching decision, children problem formulations with more restrictive constraints appear. At each of this sub-problem formulations (nodes) the Cutting Plane procedure is invoked, this is where our Cut Removal Algorithm is executed.

We present a second experiment in order to evaluate the improvement that our method yields specifically at a single-node level in the Cutting-Plane stage of the SCIP solver by isolating all parts of the SCIP workflow except the cutting plane step. This setting is analogous to our Cutting Plane implementation but in SCIP. Interfacing with SCIP allows for extended capabilities such as having different kinds of cuts (besides vanilla Gomory Cuts) and having the problem modified through pre-solving at the starting node. At each Cutting Plane method call we measure the improvement of the LP bound of B&C-Cut_Removal against B&C-Cut_Addition. The results are contained in Table 3

Table 3. Mean Improvement (%) for Each Instance (id)

Instance	8630	2891	3853	5424	4329	8627	6481	1941	3964	495	2959	4173	8509
MImp. (%)	137.83	29.69	0.09	0.69	109.31	276.44	3.09	0.01	47.71	0.37	22.20	0.00	31.97
Instance	7064	1317	5463	2229	5757	1891	4743	3736	1987	321	5833	5119	6392
MImp. (%)	91.77	117.55	322.77	59.97	38.47	0.51	2.44	0.00	8.21	1.78	9.61	102.77	35.61

D. Runtime Considerations

Wall-clock time highly depends on external factors such as implementation and programming language (e.g., C++ vs Python/PyTorch). This variability is why the number of cutting planes (number of iterations) is considered a more robust evaluation metric and has also been adopted by Paulus et al. (2022); Tang et al. (2020). This being said, we acknowledge the interest in providing results for this metric. In Table 4, we present the speedup that our method achieves when compared to the baselines in terms of wall-clock time. We measure the total time elapsed from start to finish when solving the instance. We consider instances where at least one of the policies converged. Max-Cut and Planning are not included in this table as none of the methods were able to achieve an ICG value of 1 (see Figure 2 in the paper). Thus, these problems do not permit a fair comparison. We notice our method attains an improvement over all baselines.

We emphasize that our current implementation on cut removal could be further optimized with data structures. For this reason, we believe that the reported speedups could be improved further.

Table 4. Performance Comparison

Problem/Method	Paulus et al.	Look-ahead Expert	Other Non-neural Baselines
Bin Packing	1.14x	24.59x	2.71x
Packing	1.61x	24.30x	2.15x
Set Cover	5.12x	47.26x	5.41x

E. Cut Pool Distributions

In this section we present results on the quality distribution for Gomory Cuts. We aim to answer the following questions: How is the quality in cuts distributed for different ILP families? How does this distribution evolve across the iterations? How do the cutpool quality distributions vary if acting with a random policy versus the *lookahead* rule?

MILP Benchmarks We collect trajectories of the random policy and *lookahead* policy for a total of 500 instances per problem family. At each iteration k we save each of the generated cuts (α, β), the previous LP value $x_k^\star$ and we calculate the LP value after adding the cut $x^\star_{k\cup(\alpha,\beta)}$.

Test Evaluation Metrics In section 3 we presented the bound improvement as a criteria to measure the quality of a cut (α, β).

Recall the formulation for the *normalized LP improvement* by the previous LP value:

$$\frac{c^T x^\star_{k\cup(\alpha,\beta)} - c^T x^\star_k}{c^T x^\star_k}.$$

This metric (M1) serves as an indicator on how valuable is a cut for the solution of the LP.

Consider also the similar construction but normalizing by the largest bound improvement instead.

$$\frac{c^T x^\star_{k\cup(\alpha,\beta)} - c^T x^\star_k}{\max_{(\tilde\alpha,\tilde\beta)\in\mathcal{C}_k}\{c^T x^\star_{k\cup(\tilde\alpha,\tilde\beta)} - c^T x^\star_k\}} \in [0,1].$$

This metric (M2) serves as an indicator on how valuable is a cut with respect to its cutpool.

Results For each problem family (packing, bin packing, maximum cut, production planning, set cover), each metric (M1, M2) and each policy generating the trajectory (random, *look-ahead*) we compute distribution matrix D. Consider M$(\cdot)$ to be a function that calculates metric M for each element of an array and sortd$(\cdot)$ to be a function that sorts an array in decreasing order. Let $\mathcal{C}_k^{(l)}$ to be the cutpool generated at iteration k for instance l. Then, the i-th row of D is calculated by aggregating sortd$(M(C_i^{(l)}))$ and scaling. For the scaling note that for some problems the cutpool dimensions may vary across different instances and iterations. For this reason, we divide each component by the number of existing components in the cutpools in the aggregation. Figures 4, 5, 6, 7, 8 show the heatmaps for the different distribution matrices.

The top plots (a), (b) for each figure explain how the "cut quality with respect to fellow cutpool cuts" distribution evolves across iterations for trajectories of the random and *look-ahead* policies respectively. The starting cutpools are uneven for all problems and that picking the best cuts in the first iterations (*look-ahead*) leads to more uniform cutpools compared with random picking in all problems except production planning where the uneven distribution holds.

The bottom plots (c), (d) show how the "cut quality with respect to previous LP value" distribution evolves across iterations for trajectories of the random and *look-ahead* policies respectively. For trajectories of both policies most of the bound improvement with respect to the previous solution is yielded in the first iterations except for production planning where the magnitudes hold. This explains the results observed in Figures 2, 3 where the IGC plateaus after the first iterations for all problems except for production planning where it shows a linear trend. We also observe that for the max cut the number of high quality cuts in the first iterations is significantly larger than in other families. This justifies the behaviour observed in the maxcut plot in Figures 2, 3 where for the first iterations many policies perform as well as the best one.

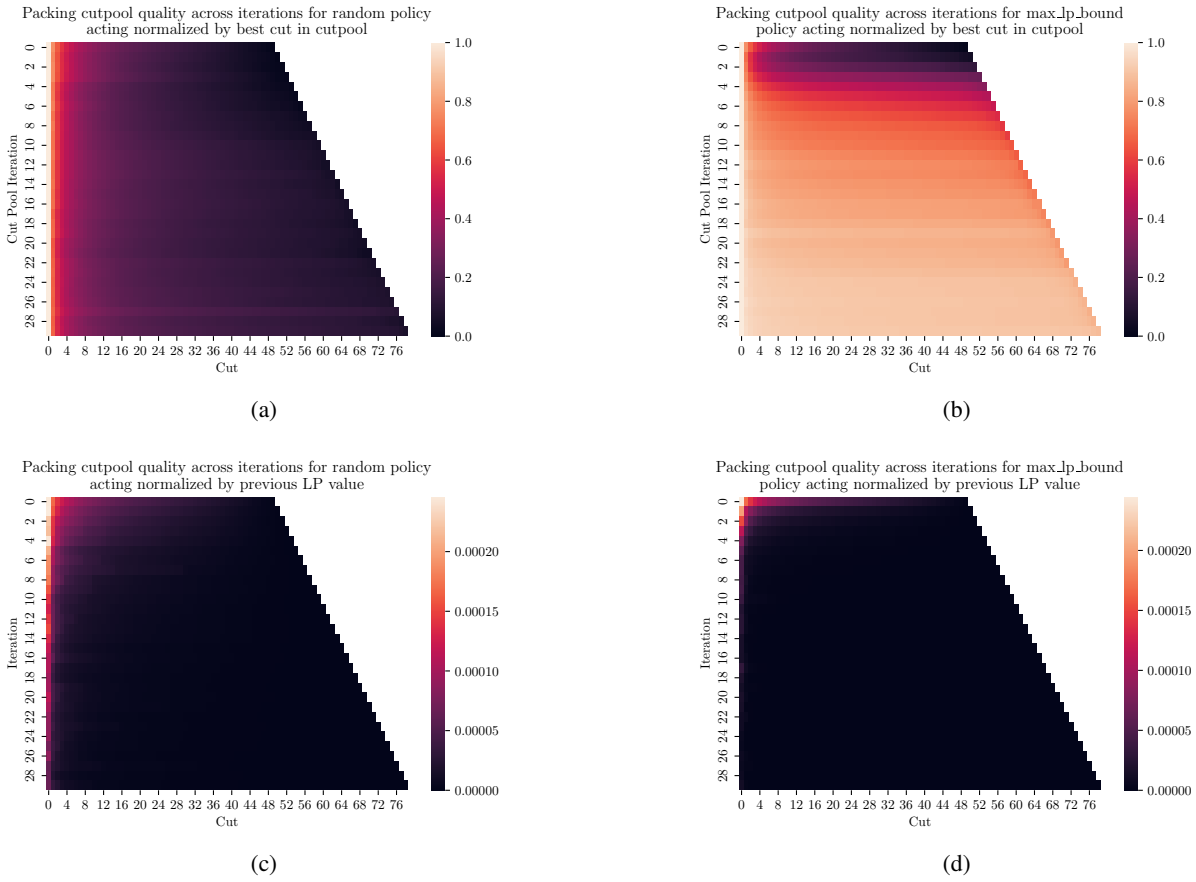

Figure 4. Packing Cutpool Distributions

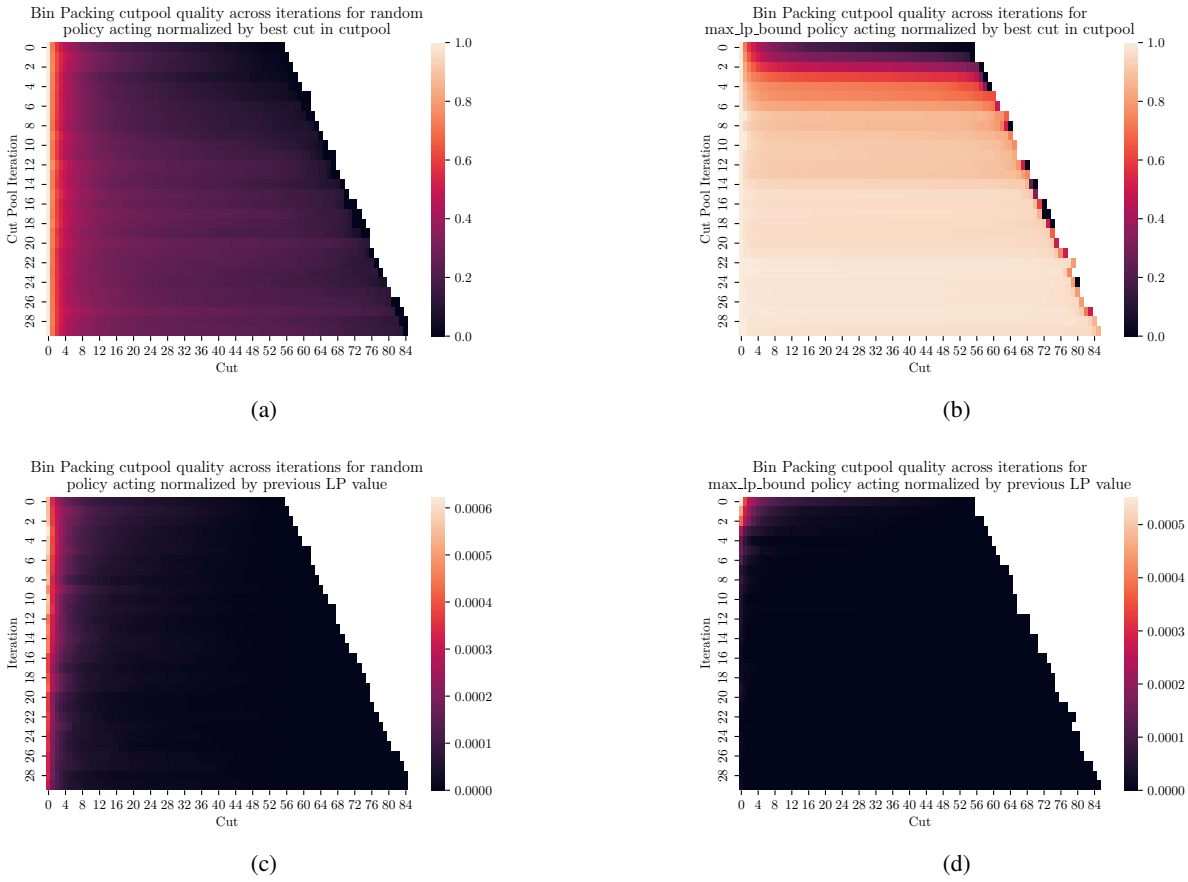

Figure 5. Bin Packing Cutpool Distributions

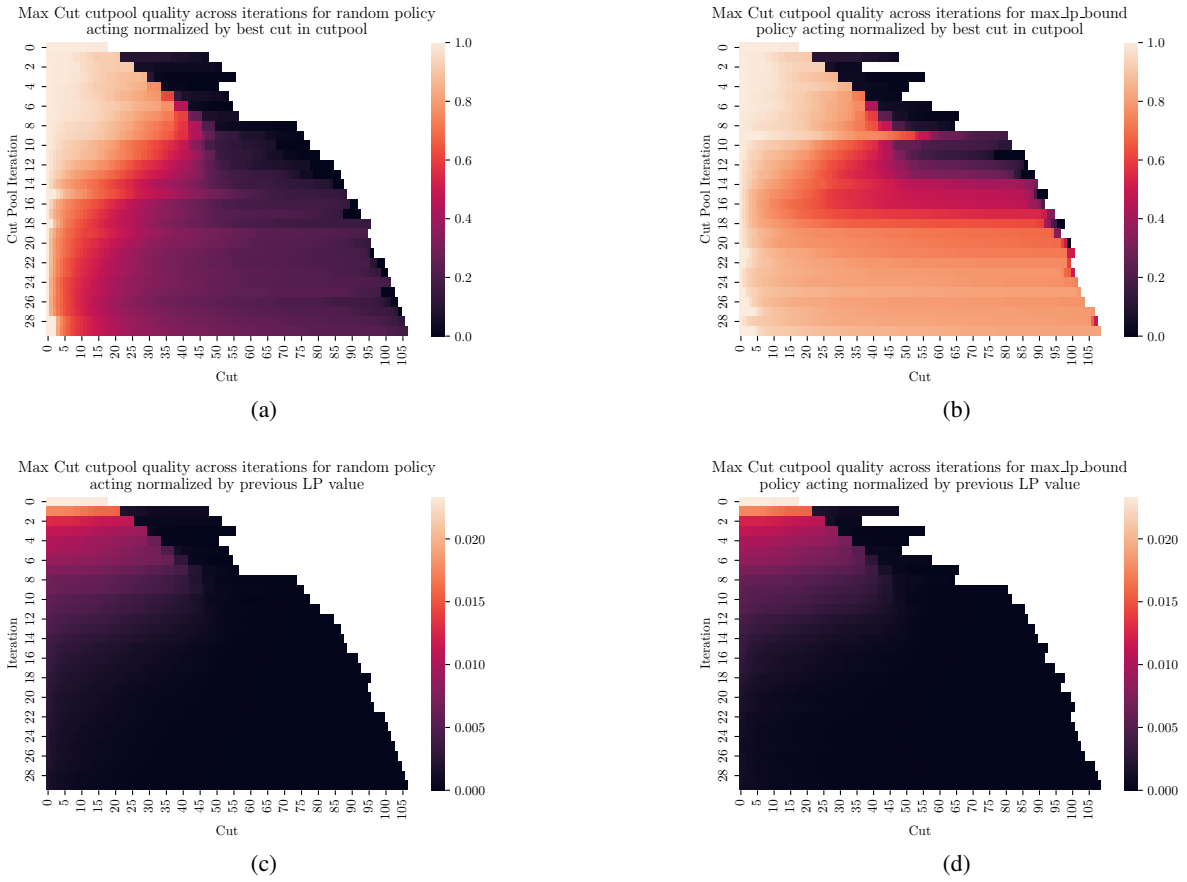

Figure 6. Max Cut Cutpool Distributions

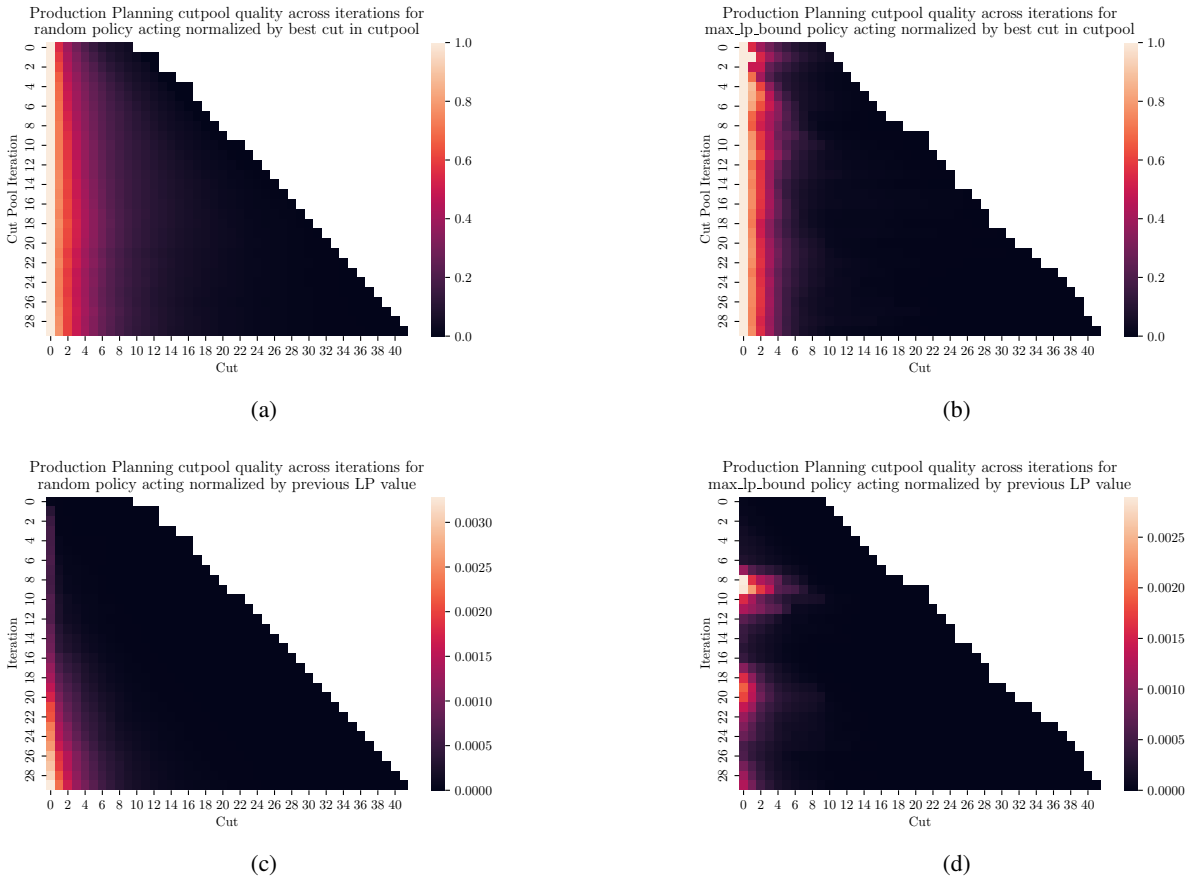

Figure 7. Production Planning Cutpool Distributions

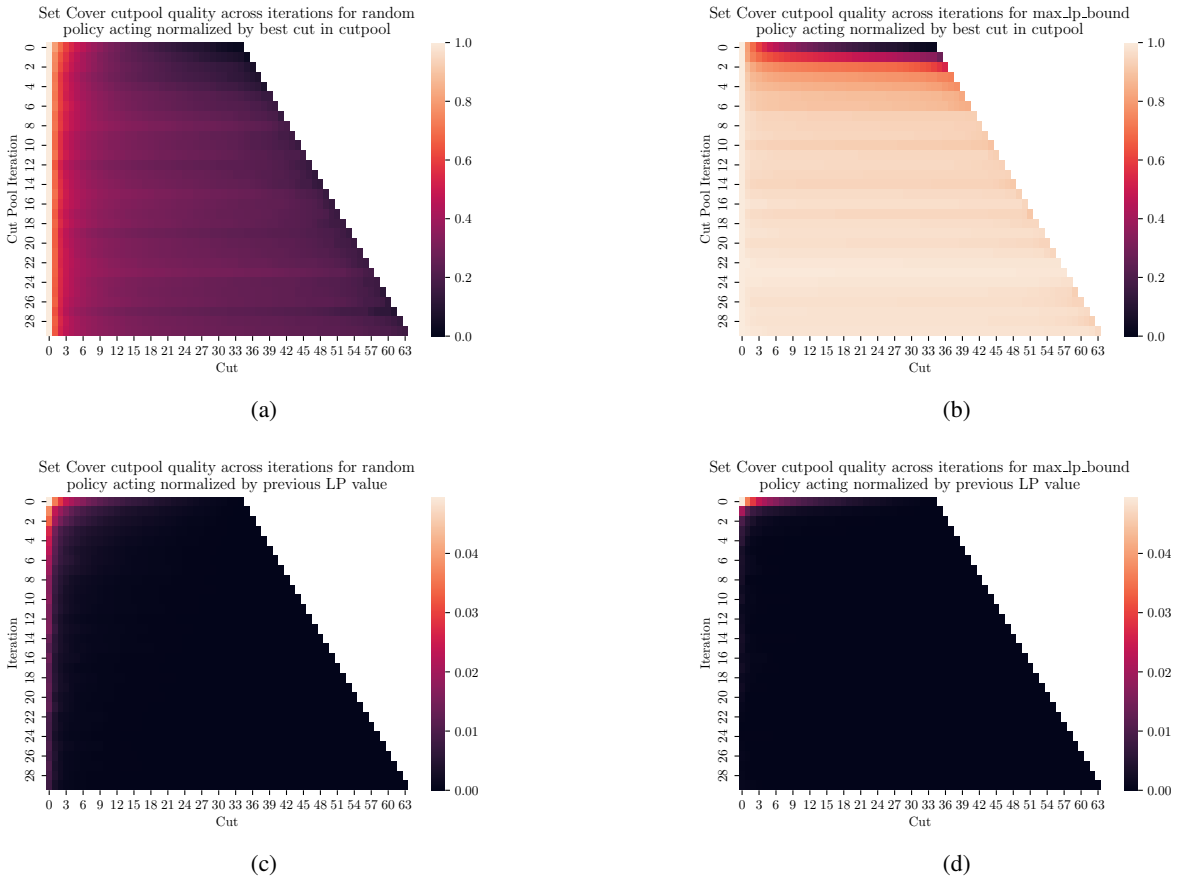

Figure 8. Set Cover Cutpool Distributions