Learning to Cut by Looking Ahead: Cutting Plane Selection via Imitation Learning

Max B. Paulus [1] Giulia Zarpellon [2] Andreas Krause [1] Laurent Charlin [3 4] Chris J. Maddison [5]

Abstract

Cutting planes are essential for solving mixed-integer linear problems (MILPs), because they facilitate bound improvements on the optimal solution value. For selecting cuts, modern solvers rely on manually designed heuristics that are tuned to gauge the potential effectiveness of cuts. We show that a greedy selection rule explicitly *looking ahead* to select cuts that yield the best bound improvement delivers strong decisions for cut selection—but is too expensive to be deployed in practice. In response, we propose a new neural architecture (NeuralCut) for imitation learning on the lookahead expert. Our model outperforms standard baselines for cut selection on several synthetic MILP benchmarks. Experiments with a B&C solver for neural network verification further validate our approach, and exhibit the potential of learning methods in this setting.

1. Introduction

Mixed-Integer Linear Programming (MILP) problems are optimization problems in which some decision variables represent indivisible choices and are thus required to assume integer values. MILP models are used in many industrial contexts (e.g., logistics, production planning), and can also be solved to verify robustness of neural networks (NN) (Cheng et al., 2017; Tjeng et al., 2018). We write a MILP as

$$z^{OPT} = \min\{c^T x : Ax \leq b, x_j \in \mathbb{Z}\, \forall j \in I\}, \quad (1)$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c, x \in \mathbb{R}^n$, and $I \subseteq \{1, \dots, n\}$ defines which variables are required to be integral. MILPs are in general $\mathcal{NP}$-hard, but modern solvers effectively tackle them by means of a collection of both exact and heuristic techniques. In particular, the MILP algorithmic framework relies on repeatedly solving (and improving) relaxed versions of the original problem that are computationally tractable. When integrality requirements $x_j \in \mathbb{Z} \forall j \in I$ are dropped, one obtains the *continuous relaxation* of (1)

$$z^* = \min\{c^T x : Ax \leq b, x \in \mathbb{R}^n\}, \quad (2)$$

which is a linear problem (LP). Clearly, $z^* \leq z^{OPT}$, and tightening the relaxation leads to better bounds on the optimal value of (1), which can be used, e.g., to improve the efficiency of a Branch-and-Bound (B&B) tree search procedure (Land & Doig, 1960)—the typical framework used to solve MILPs.

Tightening a MILP continuous relaxation can be done by introducing *cutting planes* (Gomory, 1960) in the formulation. Given an optimal solution x^* for the LP relaxation (2), a cut C is a linear inequality $\pi^T x \leq \pi_0$ that *separates* x^* from the convex hull of integer feasible solutions of (1). Formally, $(\pi, \pi_0) \in \mathbb{R}^{n+1}$ is such that $\pi^T x^* > \pi_0$ and $\pi^T x \leq \pi_0$ is valid for the feasible solutions of (1).

Cutting planes are an essential component of the MILP resolution process, with many families of both general-purpose and class specific cuts theoretically studied and implemented in modern solvers (Dey & Molinaro, 2018). Still, their practical management is governed by manually-designed heuristic rules that decide, e.g., how many cuts should be added to the formulation from a pool of candidates, which types should be preferred, how many times the separation procedure should be repeated, and also which cuts should be removed in later optimization stages. In particular, the process of selecting cutting planes from a pool relies on simple formulas that are tuned to gauge the potential effectiveness of cuts in solving MILPs, for example by measuring the extent by which a cut is violated by x^* or its nonzero support.

A natural way of assessing the impact of a cut is to quantify its resulting bound improvement. With this motivation, we design a greedy *lookahead* selection rule, explicitly trying

[1]Department of Computer Science, ETH Zürich, Zürich, Switzerland [2]Vector Institute, Toronto, Canada [3]Department of Decision Sciences, HEC Montréal, Montréal, Canada [4]Mila, Montréal, Canada [5]Department of Computer Science and Department of Statistical Sciences, University of Toronto, Toronto, Canada. Correspondence to: Max Paulus <max.paulus@inf.ethz.ch>, Giulia Zarpellon <gzarpellon@cs.toronto.edu>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

cuts beforehand to select those yielding the largest gain in terms of LP bound. We show that following such criterion allows us to close the integrality gap more effectively than common heuristics, delivering strong decisions for cut selection (Section 3.1). However, looking ahead comes with too high of a computational overhead to be viable in practice. In response, we use the lookahead rule as an *expert* to train NN policies via imitation learning; once trained, a single learned policy can be deployed on several MILP instances of the same family, so machine learning (ML) offers a way to amortize the cost of computing expert decisions.

Interfacing the MILP solver SCIP (Zuse Institute Berlin, 2021), we focus on the core task of selecting a single cut from a pool of available ones. We extract useful information from the solver to populate with novel hand-crafted input features a tripartite graph encoding of the cut selection system. We then develop NeuralCut, a NN architecture that combines graph convolutions with attention to train cut selection policies through imitation of the lookahead rule (Section 3.2). Experiments on four synthetic families of MILPs confirm the ability of NeuralCut policies to mimic the expert, selecting bound-improving cutting planes better than standard heuristics and SCIP's own cut selection criterion. Trained policies are also successful in closing the integer gap when rolled-out on never-seen test instances (Section 4.2). Finally, we stress-test our framework on the challenging benchmark of NN Verification models (Nair et al., 2020) and deploy NeuralCut in a realistic B&C framework (Section 4.3). Our results highlight the potential for improving solver performance with learned models for cut selection.

The present work shows that imitation learning is a viable approach to tackle and improve cutting planes selection in MILP solvers. Specifically, our contributions can be summarized in the following points.

- We propose a lookahead criterion for cut selection that chooses cuts based on explicit computations of LP bound improvement, and use it as the expert in imitation learning experiments.
- We design a tripartite encoding for the cut selection system and develop NeuralCut, a novel NN architecture able to scale to multiple families of MILPs.
- We highlight NN Verification as a challenging benchmark for cut selection, exhibiting the potential of ML methods in this setting.

2. Background

Cutting planes in MILP solvers are separated in successive *rounds*. At a basic level, each round k involves (i) solving a MILP continuous relaxation; (ii) inspecting a pool $\mathcal{C}^k$ of available cuts generated by the MILP solver to select a subset $S^k \subseteq \mathcal{C}^k$; (iii) adding S^k to the continuous relaxation, before proceeding to round $k+1$. On the one hand, incorporating all the cuts available in the pool into the formulation would maximally improve the bound and tighten the relaxation at each round. On the other hand, the addition of too many cuts would result in large models, which can become slower to solve and present numerical instabilities—so cut selection is necessary.

After k separation rounds the LP relaxation includes all the original constraints $Ax \leq b$, together with cutting planes (π, π_0) selected along the way: $z^k = \min\{c^T x : x \in P^k\}$ is solved at round $k+1$, where

$$\begin{aligned} P^k &= \{x \in \mathbb{R}^n : Ax \leq b, \pi^T x \leq \pi_0 \ \forall (\pi, \pi_0) \in \textstyle\bigcup_{i=1}^k S^i\}, \\ &= \{x \in \mathbb{R}^n : A^k x \leq b^k\}, \text{ with } A^0 = A, b^0 = b. \end{aligned} \tag{3}$$

The *integrality gap* after separation round k is given by the bound difference $g^k := z^{OPT} - z^k \geq 0$.

To solve MILPs, cutting planes are often combined with a B&B search procedure, in what is known as Branch-and-Cut (B&C) algorithm: cuts are applied at the root node of the system (i.e., on the first LP relaxation of a MILP, as in (2)), and also in subsequent nodes to strengthen subproblems before branching. Given that tightening the relaxation before starting to branch is decisive to ensure an effective tree search, we focus on separation iteratively happening only at the root node of a B&C system, i.e., on multiple cuts applied to (2).

Clearly, a pure cutting plane approach at the root node (without branching) is often not enough to optimally solve MILP instances, so only counting the number of added cuts does not give a complete picture of the cut selection performance. Instead, the *integrality gap closed* (IGC) focuses on computing the factor by which the integrality gap is closed, between the first relaxation (solved on P^0) and the end of separation rounds (see also Section 2 of Tang et al., 2020). Formally,

$$\text{IGC}^k := \frac{g^0 - g^k}{g^0} = \frac{z^k - z^0}{z^{OPT} - z^0} \in [0, 1]. \tag{4}$$

When experimenting with a more general B&C framework, performance measures typically focus on the total number of explored subproblems (nodes) and LP iterations needed to prove optimality.

Cut Selection in SCIP State-of-the-art solvers like SCIP (Zuse Institute Berlin, 2021) can generate (or *separate*) various families of cutting planes to populate a *cut pool* (see Section 3.3 of Achterberg, 2007). In SCIP, various parameters are tuned to work in concert and control which types of cutting planes are separated and at what frequency, and also which ones are added to the formulation. Specifically, given a pool of available cuts, a parametrized scoring function ranks them, and a cut attaining maximum score is selected to be part of S^k. The rest of the cutpool is filtered for parallelism and quality; remaining cuts are sorted again based on their initial score; and the next score-maximizing cut is selected, and so on, continuing until a pre-specified quota of cuts has been chosen for S^k, which is finally added to the LP relaxation. At its core, then, cut selection proceeds by choosing *one* cut at a time and is governed by a *scoring function*, which in SCIP is a weighted sum of different scores from the MILP literature (integer support, objective parallelism, efficacy, directed cutoff)[1], developed to empirically assess the potential quality of a given cut. We describe these scores and other cut selection heuristics in Appendix B (Table 4).

3. Learning to Cut by Looking Ahead

The central idea of our work is to learn a cut selection policy by imitation of a criterion that *looks ahead* to explicitly measure the effect of a cut on the LP bound. In this section, after providing a precise formulation of the cut selection task, we describe and motivate our expert, and establish our imitation learning framework.

Core Cut Selection Task Reviewing the cut selection process in SCIP, it becomes apparent that the core of cut selection resides in the ability to select a single cut from a set of available ones, via a scoring function. We focus precisely on this decision to formulate our cut selection task. Namely, given a pool of available cuts $\mathcal{C}$ for some relaxation P, the selection of a cut is made according to a *scorer* criterion, which determines

$$C_{j^*} = \operatorname*{argmax}_{C_j \in \mathcal{C}} \mathit{scorer}(C_j, P). \tag{5}$$

We consider multiple iterations of this basic decision, without introducing an additional notion of rounds: after a cut is selected as in (5), it is added to the formulation and the LP solved again, to produce a new pair $(\mathcal{C}', P')$ where $\mathcal{C}'$ is a freshly separated cutpool. In other words, we clear the previous $\mathcal{C}$ and replace it with cuts separated on P'. Plugging-in different *scorer* functions allows to obtain different cut selection criteria, some of which we describe in Appendix B.

3.1. Designing a Lookahead Cut Selector

As mentioned above, a natural way of assessing the impact of a cut is to measure the bound improvement it leads to in the relaxation—the bigger, the better. Of course, this metric cannot be evaluated without first trying out the cut itself, so

[1]See https://scipopt.org/doc/html/cuts_8c_source.php#l02509.

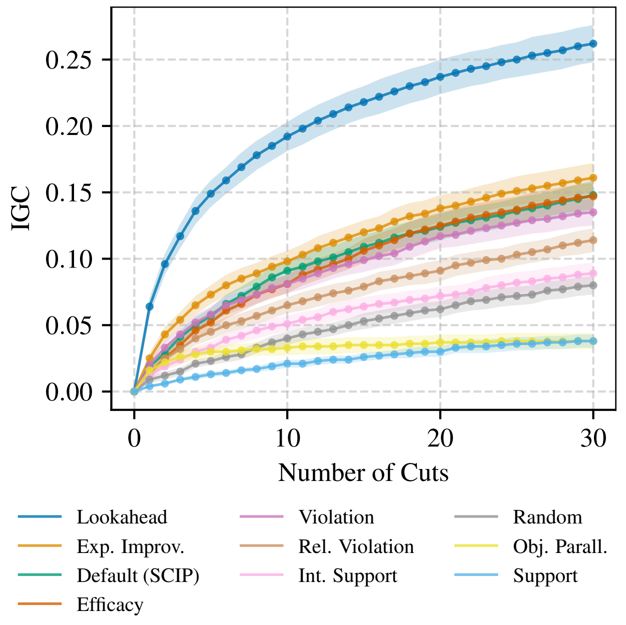


Figure 1. Lookahead clearly outperforms common heuristics for cut selection on 510 instances from the MIPLIB (easy) collection. It achieves higher mean IGC throughout when performing 30 consecutive separation rounds and adding a single cut per round.

MILP solvers make use of alternative indicators to gauge and approximate the cut effectiveness. We design a greedy criterion for cut selection performing such explicit *lookahead steps* to measure beforehand the resulting LP bound of each cut available. Given $(\mathcal{C}, P)$, for a cut $C_j \in \mathcal{C}$ defined by (π, π_0) we compare the relaxation given by P with $P^j := P \cap \{\pi^T x \leq \pi_0\}$, obtained by adding C_j. In particular, we evaluate the dual bound difference of these two relaxations, which we call the *lookahead (LA) score* of cut C_j:

$$s_{\text{LA}}(C_j, P) := z^j - z \geq 0. \tag{6}$$

Our *Lookahead* cut selection criterion is obtained by using LA scores (6) as the *scorer* function in (5). Within a cutpool, it is common to observe multiple cuts with the same LA score, i.e., establishing the same bound improvement; in case of ties, *Lookahead* breaks them randomly.

The idea of performing a lookahead step is in the spirit of the strong branching (SB) heuristic (Applegate et al., 2007) for variable selection in B&B: multiple works in the *learning to branch* literature use this explicit evaluation as the costly but valuable expert to be imitated and ultimately replaced by a learned heuristic. Although improving the dual bound is a recognized goal in cut selection (Wesselmann & Stuhl, 2012; Dey & Molinaro, 2018), to the best of our knowledge the explicit, SB-like approach has not been considered for cutting planes before.

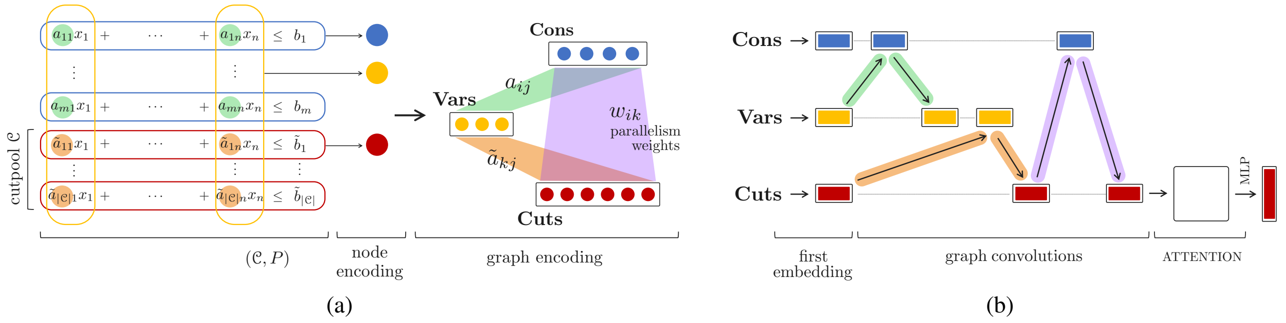


Figure 2. (a) Node and graph encoding from the state $(\mathcal{C}, P)$. Parallelism weights are used on edges between constraints and cuts. (b) NeuralCut model: after three graph convolutions, an attention module allows cuts to attend to each other within the cutpool.

A Strong (and Expensive) Criterion We assess the proposed LA cut selection rule by exposing the basic cut selection task in a custom version of SCIP, and implementing scoring functions for common heuristics as well as SCIP's own *scorer* formula (more specifications are given in Section 4). Operationally, to compute LA scores at a given relaxation P, we set SCIP in diving mode[2] to test one by one all candidate cuts in $\mathcal{C}$, and register their bound improvements.

We compare *Lookahead* to common heuristics for cut selection on 510 bounded and feasible instances from the 'easy' collection of MIPLIB 2017 (Gleixner et al., 2021). After pre-solving each instance, we perform 30 separation rounds and add a single cut per round to the LP relaxation. We average the IGC after each round over all instances and display the results in Figure 1. Lookahead clearly outperforms other rules for cut selection throughout. It achieves larger IGC with fewer cuts than all baselines. The margin over the default rule of the SCIP solver is sizable; Lookahead achieves mean 0.25 IGC (compared to $\sim$0.15 IGC of the default rule) after 30 cuts and reaches 0.15 IGC after adding only five cuts on average. Further, since MIPLIB contains diverse instances that vary in size, complexity and structure, this suggest that *Lookahead* may be a universally strong criterion for cut selection, potentially useful to improve performance in general-purpose MILP solvers.

Lookahead is thus a strong rule for selecting cuts—but an expensive one. At every iteration, running LA requires to solve $|\mathcal{C}|$ additional LPs; considering that even for small synthetic instances cutpools are typically populated by hundreds of cuts, repeating the LA iteration quickly becomes computationally intractable and makes the approach not viable in practice. In a way, *Lookahead* appears to have all the credentials needed to become an *expert* on which to train a ML policy.

[2]Diving mode allows to temporarily alter some aspects of a problem (like enforcing a new constraint or variable bound), solve it, and be able to resume the original solving process.

3.2. Lookahead Expert Imitation

In light of the cost of looking ahead, we propose to use LA scores to facilitate the training of a policy for cut selection via imitation learning. This learning approach offers a way to amortize the cost of expert decisions across entire families of MILP instances. By running several iterations of the basic cut selection loop, we can obtain a dataset of expert samples specified by $(\mathcal{C}, P, \{s_{\text{LA}}(C_j, P)\}_{C_j \in \mathcal{C}})$. We learn a cut selection policy that imitates the *Lookahead* expert defined in Section 3.1 by choosing the learnt scoring $\tilde{s}$ to minimize a soft binary entropy loss over all cuts $C \in \mathcal{C}$,

$$L(\tilde{s}) := -\frac{1}{|\mathcal{C}|} \sum_{C \in \mathcal{C}} q_C \log \tilde{s}_C + (1 - q_C) \log(1 - \tilde{s}_C) \quad (7)$$

where $q_C = \frac{s_{LA}(C)}{s_{LA}(C^*_{LA})}$ and $C^*_{LA} = \arg\max_{C \in \mathcal{C}} s_{LA}(C)$.

State Encoding and Input Features For each cut selection decision we extract information from the solver to describe the cutpool and LP relaxation pair $(\mathcal{C}, P)$ on which our *scorer* operates. Specifically, we organize the state of the system as a tripartite graph whose nodes hold vectors (features) representing variables and constraints of P (**Vars** and **Cons**), and cuts available in $\mathcal{C}$ (**Cuts**). Edges between **Vars** and **Cons** (resp. **Vars** and **Cuts**) are added when a variable appears in a constraint (resp. a cut), and carry the corresponding nonzero coefficients. We also link **Cons** and **Cuts** with a complete set of weighted edges. If (a_i, b_i) is a **Cons** and $(\tilde{a}_k, \tilde{b}_k)$ is a **Cut**, then the weight $w_{ik} \in [0, 1]$ on the edge connecting them measures their degree of parallelism, i.e., $w_{ik} = |a_i^T \tilde{a}_k| / (\|a_i\| \|\tilde{a}_k\|)$. In other words, hyperplanes are connected in the tripartite graph with a weight that measures their similarity. This facilitates a shortcut (skip connection) for communication between constraints and cuts, which could otherwise only happen via shared variables. Figure 2(a) illustrates node and graph encoding of the state $(\mathcal{C}, P)$.

For input features, we extend those defined by Gasse et al. (2019) with new ones, to emphasize the role of constraints and cutting planes in our setup. For example, we add to our **Cons** and **Cuts** features the rank and fraction of nonzeros, as well as cut scores like violation, objective parallelism, support, and SCIP weighted score. A description of our input data is given in Appendix C (Table 5), where we also discuss feature implementation.

Policy We use the graph encoding as input to NeuralCut, a new NN architecture for cut selection. The model we devise to parametrize a cut selection policy is a combination of a graph convolution neural network (GCNN) and an attention block on the cuts representations (Scarselli et al., 2009; Vaswani et al., 2017). Following the diagram in Figure 2(b), input features **Cons**, **Vars** and **Cuts** are first embedded into a hidden representation, before undergoing three passes of graph convolutions on the graph's edges. In order, the message passing happens between **Vars**$\rightarrow$**Cons**$\rightarrow$**Vars**; **Cuts**$\rightarrow$**Vars**$\rightarrow$**Cuts**; and finally **Cuts**$\rightarrow$**Cons**$\rightarrow$**Cuts**. Our rationale behind these convolution passes is to first get a sense of the current relaxation P and cutpool $\mathcal{C}$, and then establish communication between available cuts and the constraints of the problem.

A similar GCNN was proposed by Gasse et al. (2019) for B&B variable selection: we change their model to introduce **Cuts** in our inputs and convolution passes, and replace their use of layer-norm with batch normalization (Ioffe & Szegedy, 2015), which performed better in our setting.

The last part of NeuralCut focuses on the cutpool representation, which enters an attention block to enable each cut to attend to all other cuts presently in the pool. We weigh the attention module with parallelism weights—calculated as the ones between a Cons and a Cut above—$w_{k,k'}$ between each pair of cuts (k, k'). A final MLP with sigmoid activation is used to predict a score $\tilde{s} \in [0, 1]$ for each cut from its representation. Our implementation follows Shi et al. (2021), using PyTorch Geometric modules.

4. Experiments

We divide our computational experiments into two main parts. The first one focuses on evaluating learned NeuralCut policies on the core cut selection task: we measure the imitation learning performance of trained policies as well as their behavior when rolled-out on synthetic test MILP instances, in a controlled solver environment. Next, we investigates the impact that LA policies can have in a more realistic B&C solver framework, with a dataset of real-world Neural Network Verification instances.

4.1. Evaluation Protocol

Solver Interface We derive a custom version of the SCIP solver (v.7.0.2) to expose and isolate the cut selection task as defined in Section 3. Our interface with the solver covers ten different separators, namely, `aggregation`, `clique`, `disjunctive`, `flowcover`, `gomory`, `impliedbounds`, `mcf`, `oddcycle`, `strongcg`, `zerohalf`. Heuristics for the *scorer* function are implemented following Wesselmann & Stuhl (2012), and we mirror the SCIP formula (SCIPScore). As previously mentioned, we exploit SCIP's diving mode to compute LA scores. The solver parametric setting is specified in Appendix D.

Data Collection and Baselines We experiment with different types of MILP models (see next sections). For each MILP dataset, we execute the first 10 iterations in our controlled cut selection environment to collect samples $(\mathcal{C}, P, \{s_{\text{LA}}(C_j, P)\}_{C_j \in \mathcal{C}})$. Ten cuts are sometimes enough to solve the easiest instances to optimality, and in general we observe that with more iterations the quality of cuts becomes more uniform within a cutpool. To diversify the training space, during these 10 iterations we select cuts according to a *scorer* uniformly drawn from $\{$Random, SCIPScore, *Lookahead*$\}$. Note that regardless of the used *scorer*, we always gather LA scores on the resulting $(\mathcal{C}, P)$ state.

The learned policy is compared to the *Lookahead* expert and the SCIPScore baseline, as well as other *scorer* heuristics including Efficacy, ExpImprovement, ObjParallelism, RelViolation, Violation, Support, IntSupport, Random. Again, we refer to Appendix B for details.

ML Setup and Training We train NeuralCut policies with ADAM (Kingma & Ba, 2015) using the default PyTorch setting (Paszke et al., 2019). Our evaluations run on a distributed compute cluster; hardware specification are reported in Appendix E.

4.2. Core Task Evaluation

This first set of experiments is focused on assessing the quality of our imitation learning approach. We work in the controlled cut selection environment introduced above, and answer the following points: Are the proposed encoding and architecture (Figure 2) effective in modeling the task of cut selection, and do they enable the imitation of *Lookahead*? Are trained NeuralCut policies successful in closing the integer gap when rolled-out on never seen test instances? We answer both questions positively on four different MILP benchmarks.

MILP Benchmarks We consider the four classes of MILP models used in Tang et al. (2020) and reproduce their generators for instances of type Maximum Cut, Packing, Binary Packing and Planning; we refer to Tang et al. (2020) supplementary material for details on the models' mathematical formulations. A quick assessment revealed that instances of size Small and Medium were too easy for our setting, often being solved at presolve or after the addition of a handful of cuts. We thus focus on Large instances only, which exhibit sizes $n, m \in [50, 150]$, and generate a total of 3000 models for each family, divided into (2000, 500, 500) for (training, validation, test). Following the procedure outlined above, we gather just short of 20K samples for each training set, due to some models being solved in less than 10 cut selection iterations during data collection. Details about the composition of each benchmark are reported in Appendix A.

Test Evaluation Metrics To measure the imitation performance of our policies, we design a *bound fulfillment* metric assessing the quality of selected cuts. Specifically, for each collected test sample we look at its cutpool $\mathcal{C}$ and scale the attainable bound improvements within $[0, 1]$, fixing at 1 the best one, specified by the cutpool's maximum LA score. Cut selection decisions for a trained policy and other heuristics can then be compared in this range to measure their ability in selecting bound-improving cuts. Note that *Lookahead* achieves by definition a bound fulfillment of 1.

We also plug-in the learned policies in our custom solver interface and evaluate their roll-outs on never-seen MILP test instances. For this online setting we employ IGC as defined in Section 2 to track the integer gap, also measuring the area *over* the IGC curve (the smaller, the better) to summarize each policy's performance. We call this metric *reversed IGC integral*; note that in this case the minimum attainable integral is constrained by the composition of the cutpools. Runtime improvements are of marginal interest in diagnostic instances like those from Tang et al. (2020), so we do not evaluate runtimes for this first set of experiments.

Results Table 1 reports results on the evaluation of NeuralCut policies trained on each MILP family. Bound fulfillment rates (left side of the table) confirm that NeuralCut effectively learns to imitate *Lookahead*, selecting bound-improving cutting planes better than all other heuristic baselines including SCIPScore, and scoring almost perfectly on Max. Cut and Plan datasets. Even though the difficulty of the selection task may vary across benchmarks, as reflected by the average size of the cutpools in the test samples, the imitation learning performance is quite impressive and homogeneous. In terms of integer gap closed over 30 iterations roll-outs (i.e., on the addition of the first 30 cuts) (right side), on 3/4 datasets NeuralCut achieves smallest reversed IGC integrals, the nearest to *Lookahead*; overall, we observe a good correlation with the bound fulfillment metric. The Packing case is tied between many rules, and does not seem to offer much space for improvement via LA greedy selection. Figure 3 depicts mean IGC curves for NeuralCut on

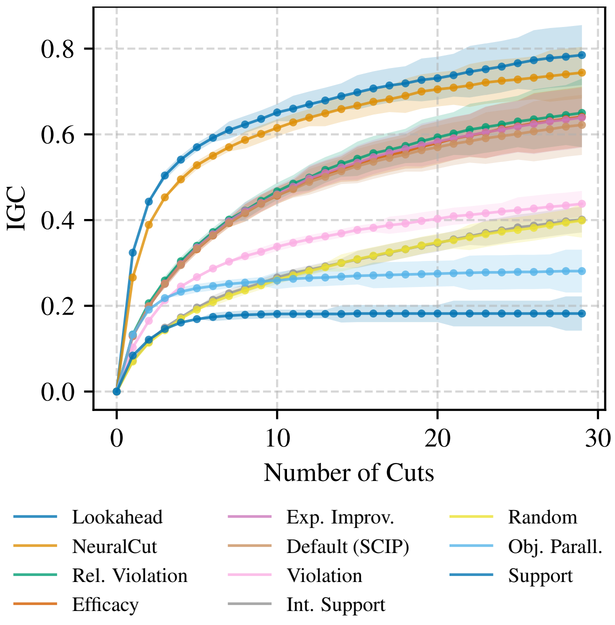


Figure 3. On binary packing test instances, NeuralCut nearly achieves the same IGC as *Lookahead* that it was trained to imitate. It outperforms other heuristics for cut selection. IGCs for the other benchmarks are in Figure F.1 in Appendix F.1.

binary packing instances, and clearly shows the advantage of the LA-trained policy over all baselines; similar plots for the other benchmarks are collected in Appendix F.

We also compare NeuralCut to the reinforcement learning (RL) approach for cut selection proposed by Tang et al. (2020). They cast cut selection as an RL problem and use improvements to the LP bound as rewards to train a model for cut selection via evolutionary strategies. Tang et al. (2020) consider only Gomory cutting planes and implement their model with Gurobi (Gurobi, 2022). To compare to them, we re-implement their method in our SCIP solver interface and expose it to all cutting planes. NeuralCut outperforms RL on all four benchmarks (Table 1). It achieves higher bound fulfillment and lower reversed IGC integral. The difference is particularly acute on the binary packing instances (0.78 *vs* 0.22 bound fulfillment and 19.96 *vs* 10.96 *vs* 16.06 reversed IGC integral)..

With respect to ML modelling choices previously used for MILP optimization, NeuralCut introduces several innovations. These include new features, the introduction of shortcut connections between constraints and cuts and the use of an attention module for the cutpool. In Table 2, we perform model ablations on the binary packing instances, to better assess the effectiveness of our modelling choices. We begin with a simple GNN model that is based on Gasse et al. (2019), but adapted to cut selection by using cuts

Table 1. NeuralCut selects cuts with high average bound fulfillment (left) and smaller average reversed IGC integral on four benchmarks. It outperforms both a competing RL approach (Tang et al., 2020) and manual heuristics for cut selection. It approximates the performance of *Lookahead* closely. Performance on test instances for 30 consecutive separation rounds and adding a single cut per round. Best model or heuristic is bold-faced, best overall is in italics.

	Bound fulfillment (↑) on test samples, mean				**Reversed IGC integral (↓) on test instances**, mean (ste)			
	MAX. CUT	PACKING	BIN. PACKING	PLANNING	MAX. CUT	PACKING	BIN. PACKING	PLANNING
Lookahead	*1.0*	*1.0*	*1.0*	*1.0*	*15.05 (0.09)*	*26.42 (0.08)*	*9.85 (0.33)*	*10.33 (0.04)*
NeuralCut	**0.96**	**0.61**	**0.78**	***1.0***	**15.55 (0.09)**	***26.30 (0.08)***	**10.96 (0.33)**	**10.42 (0.04)**
Tang et al. (2020)	0.58	0.27	0.22	0.49	19.00 (0.09)	27.59 (0.06)	16.06 (0.38)	14.94 (0.08)
Default (SCIP)	0.71	0.60	0.33	0.64	16.72 (0.09)	***26.29 (0.08)***	15.42 (0.28)	14.01 (0.06)
Exp. Improv.	0.69	0.60	0.32	0.85	19.00 (0.08)	26.27 (0.07)	15.14 (0.29)	11.15 (0.05)
Efficacy	0.65	0.60	0.32	0.46	17.01 (0.09)	***26.28 (0.08)***	15.19 (0.29)	14.52 (0.06)
Obj. Parall.	0.47	0.34	0.27	0.44	24.01 (0.08)	28.28 (0.05)	22.23 (0.26)	27.71 (0.07)
Rel. Violation	0.50	0.60	0.33	0.48	17.95 (0.08)	***26.28 (0.08)***	14.90 (0.29)	15.36 (0.06)
Violation	0.64	0.35	0.21	0.26	23.20 (0.08)	28.81 (0.03)	19.41 (0.31)	25.75 (0.09)
Support	0.57	0.18	0.13	0.29	19.31 (0.10)	28.77 (0.04)	24.79 (0.22)	18.39 (0.09)
Int. Support	0.62	0.18	0.21	0.34	21.87 (0.07)	28.78 (0.03)	21.14 (0.28)	23.31 (0.08)
Random	0.41	0.15	0.16	0.25	21.99 (0.07)	28.73 (0.04)	21.23 (0.28)	23.26 (0.08)

Table 2. Our model choices tend to improve tend to improve both bound fulfillment on test samples and reversed IGC integral on test instances for binary packing. Ablations for the other benchmarks are in Table 7 in Appendix F.

	Bound full.	**Rev. IGC integral**
Gasse et al. (2019)	0.54	16.09 (0.31)
+ our features	0.66	12.77 (0.32)
+ our architecture	0.76	**10.73 (0.33)**
NeuralCut (+ our graph)	**0.78**	**10.96 (0.33)**

in place of constraints in the original model formulation of Gasse et al. (2019) and switching the order in which half-convolutions are applied. Second, we use the features of NeuralCut (+ our features) for both cuts and variables instead of the features of Gasse et al. (2019). Third, we additionally make architectural choices, i.e. we replace layer normalization with batch normalization and add the attention module to the model. Finally, we leverage the tri-partite graph formulation to model constraints explicitly (in addition to the cuts and variables), which gives the full NeuralCut model. For binary packing, improvements are most pronounced when using our features (0.54 *vs* 0.66, 16.09 *vs* 12.77) and applying our architectural choices (0.66 *vs* 0.76, 12.77 *vs* 10.73). Ablations for the other benchmarks are in Table 7 in Appendix F.3. In addition, an ablation on the choice of loss function is in Table 8 in Appendix F.4.

4.3. NeuralCut in a B&C Solver

Up to this point we worked in a controlled environment and focused on the core task of selecting a single cut from a pool of available ones. While for the sake of evaluation clarity we isolated this decision from all the other choices related to cuts management happening in a solver, in practice these heuristics work in concert and impact each other.

After having determined the effectiveness of NeuralCut policies in the controlled environment, we now wish to investigate the impact that selecting good (bound-improving) cuts can have on a more complex and realistic solver setting. To do so, we deploy NeuralCut policies in the SCIP original B&C framework, and compare them against the default solver on a challenging MILP dataset of neural network verification instances.

NN Verification Dataset The problem of verifying the robustness of a neural network to input perturbation can be formulated as a MILP (Cheng et al., 2017; Tjeng et al., 2018). Because each input to be verified gives rise to a different instance, this application is particularly suited to get large MILP benchmarks that are not synthetic. The dataset we use to stress-test our policies was open-sourced by Nair et al. (2020).[3] Compared to synthetic instances, these MILPs are larger: Nair et al. (2020) report median sizes (n, m) at (7142, 6531) and harder to solve for SCIP. Interestingly for us, the formulation of these models is notoriously weak, i.e., a lot of work needs to be done to tighten the dual bound of the LP relaxation. From the original dataset splits we select 1807 problems (Appendix A).

[3] https://github.com/deepmind/deepmind-research/tree/master/neural_mip_solving

Table 3. NeuralCut in a B&C solver – Median metrics for NN verification test instances and different values for the threshold parameter ϵ. NeuralCut improves the dualbound at the root at a fraction of the number of cuts the default solver SCIP selects. When subsequently branching from the root node, this can reduce the remaining solving time by lowering the number of LP iterations or expanded nodes in the search tree. Best are bold-faced.

	# cuts	**Rel. bound improv.**	**Time (s)**	**# nodes**	**# LP iters.**
SCIP B&C	279	**1.00**	23.65	745	21933
NeuralCut, $\epsilon = 10^{-5}$	105	**1.00**	22.35	**671**	18846
NeuralCut, $\epsilon = 10^{-4}$	81	0.99	**20.89**	756	19310
NeuralCut, $\epsilon = 10^{-3}$	48	0.98	22.73	803	**18048**
NeuralCut, $\epsilon = 10^{-2}$	27	0.94	24.06	861	**18048**
NeuralCut, $\epsilon = 10^{-1}$	11	0.76	25.09	998	21996
NeuralCut, $\epsilon = 1$	**10**	0.53	23.35	952	21611

Solver Plug-in To deploy our policies in SCIP we implement a plug-in that gets called during separation rounds;[4] we minimally interfere with the solver's default process and only operate at the root node level. In particular, we do *not* override SCIP separation and cuts management parameters, and only force the sorting of SCIP cutpools based on our predictions. In this realistic solver environment, our models also need to make the choice of whether to add a cut: we combine NeuralCut with a simple stalling rule parametrized on a threshold parameter ϵ, stopping the selection after the bound improvement from adding cuts has not exceeded ϵ for 10 consecutive rounds. Technical details, solver parametric setting and changes to data collection are discussed in Appendix D.

Test Evaluation Metrics We assess NeuralCut policies' performance in the B&C system with (1) measures collected at the end of the root node, like total number of cuts and bound improvement relative to SCIP (1.0 = same bound as SCIP), and (2) end-of-run statistics, like solving time, number of nodes and number of LP iterations, which we collect when the MILP is subsequently solved. Wanting to highlight the potential of improving B&C solvers with ML models for cutting planes selection, for the second set of metrics we factor out the root node contributions, where NeuralCut operates. This allows us to get a clearer picture of the impact that an improved cut selection at the root node can have on the remaining solving process. Because of the presence of outliers in the dataset, we report median statistics.

Results Figure 4 shows learning curves (binary entropy loss and bound fulfillment) for a NeuralCut policy trained on NN verification for 32 epochs. The model learns to select good cuts: it achieves bound fulfillment score of over 70% from an initial 38%, confirming the ability of NeuralCut to scale to real-world instances. Table 3 summarizes the performance of NeuralCut policies compared to the default B&C solver. At the root node, NeuralCut reaches the same or very similar median bound while only adding a fraction of SCIP's cut budget (20-40%). This cut efficiency at the root node translates into improved solving behaviour after the root node relaxation. When subsequently solving the MILP, NeuralCut's additions help to improve the median remaining solving time by up to 10% (in NeuralCut with $\epsilon = 10^{-4}$), by reducing the number of nodes in the search tree and the number of LP iterations. Overall, results on the realistic solver setting validate the positive effects of selecting bound-improving cuts which were already established in Section 4.2, and exhibit the potential of ML methods in this complex setting.

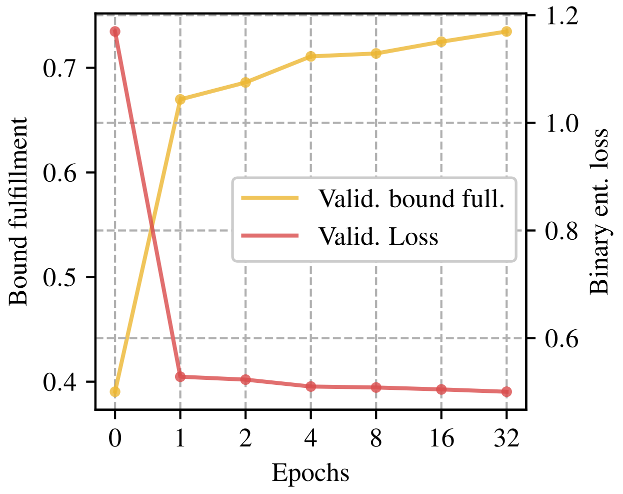


Figure 4. NeuralCut attains good validation performance after training only for one epoch. Validation performance improves as training continues.

[4]Our plug-in is inspired by https://github.com/avrech/learning2cut.

5. Related Work

Recent years have seen a rise in the application of ML approaches to the MILP algorithmic framework (Bengio et al., 2021), and imitation learning has been identified as a natural approach for tasks in the field (Marcos Alvarez et al., 2017; Hansknecht et al., 2018; Gasse et al., 2019; Zarpellon et al., 2021). On cutting planes, Radu et al. (2018) first introduced a NN estimator predicting the objective improvement of a cut in the context of semidefinite programming relaxations and quadratic optimization. Closer to our setting, the work of Tang et al. (2020) proposes a reinforcement learning framework for the Gomory cutting plane algorithm, where policies are trained via evolutionary strategy. Huang et al. (2021) instead frame cut selection as multiple instance learning, and apply it to a proprietary solver. On the theoretical side, Balcan et al. (2021) produce provable guarantees for learning cut selection policies.

The idea of performing explicit lookahead steps was introduced in the MILP literature with Strong Branching (Applegate et al., 2007), and subsequently explored by Glankwamdee & Linderoth (2011) and Schubert (2017), always in the context of variable selection. Related to explicitly optimizing cutting planes on their bound improvements, relevant works are (Amaldi et al., 2014) and (Coniglio & Tieves, 2015), the latter proposing a "bound optimal" cutting plane *separation* algorithm.

Finally, GNNs have been used before to approach combinatorial tasks (Cappart et al., 2021), and part of our model is similar to the one used by Gasse et al. (2019) for learning to branch. Attention mechanisms and Transformer architectures were first proposed in the Natural Language Processing field (Vaswani et al., 2017), and are being rapidly adopted in several other applications.

6. Conclusions

The task of cutting planes selection is essential for solving MILPs and we presented a new imitation learning framework to tackle it. We defined a *Lookahead* criterion that greedily selects cuts based on LP bound gains: we showed that this rule delivers strong decisions for cut selection and appears more effective than common heuristics, but with the drawback of being too expensive to be viable in practice. Calling ML to the rescue, we used LA scores to train a policy via imitation learning. We organized the cut selection state in a graph structure and developed a novel NN architecture to predict scores over a cutpool. Controlled experiments on four synthetic families of MILPs show how our NeuralCut policies outperform standard heuristics, proving effective in imitation and successful in closing the integrality gap when rolled-out on never-seen test instances. A stress-test on the challenging benchmark of NN Verification models further validates our approach, and demonstrates the positive effects of learning to select good (bound-improving) cuts at the root node of a realistic B&C framework. Our results highlight the potential for improving solver performance with learned models for cut selection.

The heuristic nature and complexity of MILP cuts management leaves plenty of space for ML approaches. Interesting questions include the decision of whether to trigger separation and when to stop cut addition. Exploring cutting planes at local nodes could instead allow to study the generalization ability of policies trained at the root of a B&C tree. Finally, because *Lookahead* appears to be a universal criterion for cut selection, it could potentially be used to improve performance in the general-purpose setting, too. We believe that investigating strategies to achieve broader generalization of ML for MIP (e.g., via diverse training sets, surrogate losses, model architectures) is an interesting agenda for future research.

Acknowledgements

We would like to thank Antoine Prouvost, Andrea Lodi and Avrech Ben-David for helpful discussions on the topic. MBP gratefully acknowledges support from the Max Planck ETH Center for Learning Systems. Resources used in preparing this research were provided, in part, by the Sustainable Chemical Processes through Catalysis (Suchcat) National Center of Comptence in Research (NCCR), the Province of Ontario, the Government of Canada through CIFAR, in particular the CIFAR AI Chairs and CIFAR AI Catalyst Grants programs, and companies sponsoring the Vector Institute. Finally, we thank Sam Fux from the High Performance Computing group at ETH Zürich for his help with getting things running on the ETH Euler cluster.

References

Achterberg, T. *Constraint Integer Programming*. PhD thesis, Technical University of Berlin, 2007.

Amaldi, E., Coniglio, S., and Gualandi, S. Coordinated cutting plane generation via multi-objective separation. *Mathematical Programming*, 143(1):87–110, 2014.

Applegate, D., Bixby, R., Chvátal, V., and Cook, W. *The Traveling Salesman Problem. A Computational Study*. Princeton University Press, 2007.

Balcan, M.-F., Prasad, S., Sandholm, T., and Vitercik, E. Sample complexity of tree search configuration: Cutting planes and beyond. *arXiv preprint arXiv:2106.04033*, 2021. Appeared at NeurIPS 2021.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2020.07.063. URL https://www.sciencedirect.com/science/article/pii/S0377221720306895.

Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Veličković, P. Combinatorial optimization and reasoning with graph neural networks. In Zhou, Z.-H. (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 4348–4355. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/595. URL https://doi.org/10.24963/ijcai.2021/595. Survey Track.

Cheng, C.-H., Nührenberg, G., and Ruess, H. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 251–268. Springer, 2017.

Coniglio, S. and Tieves, M. On the generation of cutting planes which maximize the bound improvement. In *International Symposium on Experimental Algorithms*, pp. 97–109. Springer, 2015.

Dey, S. S. and Molinaro, M. Theoretical challenges towards cutting-plane selection. *Math. Program.*, (170):237–266, 2018. doi: 10.1007/s10107-018-1302-4.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems 32*, 2019.

Glankwamdee, W. and Linderoth, J. Lookahead branching for mixed integer programming. In *Twelfth INFORMS Computing Society Meeting*, pp. 130–150. INFORMS, 2011.

Gleixner, A., Hendel, G., Gamrath, G., Achterberg, T., Bastubbe, M., Berthold, T., Christophel, P., Jarck, K., Koch, T., Linderoth, J., et al. Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3): 443–490, 2021.

Gomory, R. An algorithm for the mixed integer problem. Technical Report RM-2597, The Rand Corporation, 1960.

Gowal, S., Dvijotham, K., Stanforth, R., Bunel, R., Qin, C., Uesato, J., Arandjelovic, R., Mann, T., and Kohli, P. On the effectiveness of interval bound propagation for training verifiably robust models, 2019.

Gurobi. Gurobi Optimizer, 2022. URL http://www.gurobi.com.

Hansknecht, C., Joormann, I., and Stiller, S. Cuts, primal heuristics, and learning to branch for the time-dependent traveling salesman problem. *arXiv preprint 1805.01415*, 2018.

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. arXiv preprint 2105.13645, 2021.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456. PMLR, 2015.

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

Land, A. and Doig, A. An automatic method of solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

Marcos Alvarez, A., Louveaux, Q., and Wehenkel, L. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017. ISSN 1091-9856. doi: 10.1287/ijoc.2016.0723.

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.

Prouvost, A., Dumouchelle, J., Scavuzzo, L., Gasse, M., Chételat, D., and Lodi, A. 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.

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.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Schubert, C. Multi-level lookahead branching. Master's thesis, Technische Universität Berlin, 2017.

Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., and Sun, Y. Masked label prediction: Unified message passing model for semi-supervised classification, 2021.

Tang, Y., Agrawal, S., and Faenza, Y. Reinforcement learning for integer programming: Learning to cut. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9367–9376. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/tang20a.html.

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. NIPS'17, pp. 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(5):3931–3939, May 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/16512.

Zuse Institute Berlin. The SCIP Optimization Suite, 2021. https://scip.zib.de.

A. MILP Instances

A.1. MIPLIB 2017

We used 510 instances of the MIPLIB 2017 collection (Gleixner et al., 2021). We only considered instances that were considered 'easy' by (Gleixner et al., 2021) in spring 2022, bounded and feasible, such that IGC could be readily computed. For 119 out of 629 'easy', bounded and feasible instances, we were unable to perform the total 30 separation rounds (with all cutting plane generators) on standard compute hardware within 24 hours and excluded these instances from consideration.

A.2. Benchmarks

We used instances from the four integer programming domains maximum cut, packing, binary packing and planning as suggested by Tang et al. (2020) for cut selection. Their exact mathematical formulation is given in Tang et al. (2020). They consider three different sizes (*small*, *medium*, *large*) for each domain. Early experiments revealed that *small* and *medium*-sized instances were often pre-solved or easily solvable by adding a very small number of cuts. Therefore, we only considered *large* instances. For each domain, we generated 1000 instances for training, 500 instances for validation and 500 instances for testing. To generate these instances, we followed the protocol in (Tang et al., 2020). We used graphs with $|V| = 14$ vertices and $|E| = 40$ edges for maximum cut, packing problems with $m = 60$ (resource) constraints $n = 60$ variables, binary packing problems with $m = 66$ (resource) constraints, 66 binary constraints and $n = 66$ variables and a time horizon $T = 40$ for production planning.

A.3. Neural Network Verification

We also work with a neural network verification dataset from Nair et al. (2020), which is obtained from the verification of a trained convolutional NN on each image in the MNIST dataset. The corresponding MILP mathematical formulation is given by Gowal et al. (2019). We select from the original MILP collection a total of 1807 problems: we exclude infeasible instances (often trivially solved at presolve) and those models hitting a 1 hour time-limit in SCIP default mode. We considered 1260 instances for training, 271 instances for validation and 276 instances for testing.

B. Baseline Heuristics

A description of baselines heuristics used for cut selection and their SCIP implementation details are given in Table 4. In particular, note that while our *Lookahead* rule computes the bound improvement exactly, the *expected improvement* heuristic (ExpImprovement) aims at estimating the LP gain of a cut (under certain assumptions), and is derived as a product of other two scores. A useful reference is (Wesselmann & Stuhl, 2012).

C. Input Features

We interface SCIP and collect input features via custom PySCIPOpt functions, closely following the structure of Ecole's NodeBipartite observation function[5] (Prouvost et al., 2020). With respect to the features defined in (Gasse et al., 2019), we remove from **Vars** two attributes related to incumbent values (incval and avgincval), as very rarely an incumbent has been found already at the root node for our problems and parametric setting. We substantially extend the parametrization of constraints and cutting planes. Note that we represent both types of rows by the same feature set, even though parts of the feature (e.g., scores) are more meaningful for just one of the two types (e.g., cuts). Our input state for a cut selection sample on $(\mathcal{C}, P)$ is thus composed of **Vars** $\in \mathbb{R}^{n\times 17}$, **Cons** $\in \mathbb{R}^{m\times 34}$ (where n, m are the number of variables and constraints in P), and **Cuts** $\in \mathbb{R}^{|\mathcal{C}|\times 34}$. We attach to edges of the graph encoding scalar row coefficients and, for each pair of (cut, constraint) their parallelism coefficient as weight. A detailed description of the input features can be found in Table 5.

D. SCIP Interface and Parametric Settings

Core Task Evaluation (Section 4.2) Our controlled SCIP environment covers separators: `aggregation`, `clique`, `disjunctive`, `flowcover`, `gomory`, `impliedbounds`, `mcf`, `oddcycle` (not enabled in SCIP default), `strongcg`, `zerohalf`. It runs with presolve enabled and without primal heuristics; we override SCIP's separation and enforce instead our cut selection loop on the 10 custom separators, by setting the node limit to 1, disabling SB computa-

[5] `https://github.com/ds4dm/ecole/blob/master/libecole/src/observation/node-bipartite.cpp`

Table 4. Description of baselines heuristics used for cut selection.

Name	Description
Random	Random cut selection.
Violation	Infeasibility of a cut with respect to the current LP solution. Computed as −SCIProwGetLPFeasibility.
RelViolation	Relative violation, i.e., violation score scaled by $\min$ of a cut rhs (π_0) and lhs ($\pi^T x$).
Efficacy	Cut's efficacy with respect to the current LP solution. Computed as −SCIProwGetLPFeasibility scaled by the cut's euclidean norm.
ObjParallelism	Parallelism of a cut with the objective function. Computed as SCIProwGetObjParallelism.
ExpImprovement	Expected improvement, i.e., squared Euclidean norm of objective function vector multiplied by objective parallelism and efficacy. Computed as objsqrnorm · SCIPgetCutObjParallelism · SCIPgetCutEfficacy.
Support	Support score, negative proportion of a cut nonzero support. Computed as −SCIProwGetNNonz scaled by the number of variables.
IntSupport	Integer support, i.e., proportion of cut's integer entries over nonzero ones. Computed as SCIProwGetNumIntCols / SCIProwGetNNonz.
SCIPScore	SCIP default score, i.e., weighted sum of obj. parallelism, integer support and efficacy scores.

tions and fixing `minefficacy` parameters to 0, to include all separated cuts in the pool. Because in our experiments we disable all primal heuristics, we mirror SCIP's formula (SCIPScore) without including the directed cutoff metric, i.e., as it is computed in the solver when no primal solution is available.

NeuralCut in a B&C Solver (Section 4.3) The plug-in for experiments in SCIP minimally interferes with the default optimization process. With respect to the SCIP procedure described in Section 2, we only get input features, LA scores (when collecting data), and force the sorting of SCIP cutpools based on our predictions, so that SCIP automatically selects the cut identified by LA/a trained policy. We only operate at the root node separation level. We parametrize the stalling rule to operate NeuralCut policies with two parameters: a tolerance ϵ to define stalling on bound improvement, and the number K of consecutive stalling rounds. In our experiments, we fix $K = 10$, similarly to SCIP's own parameter.

For data collection in this setting, we use SCIP default rule as the "exploration agent" and collect data within SCIP rounds with 1% probability; we fix a low collection rate to keep our dataset small and diverse, as we found there is little information gained from storing samples from highly correlated rounds. In terms of parametric setting, we run experiments with presolve and primal heuristics enabled, no node limit, and a time-limit of 1h; we disable restarts and reoptimization. In particular, we do *not* override SCIP separation and cuts management parameters.

E. Hardware Specifics

We collect data for training and evaluate all methods on a distributed compute cluster which predominantly contains Intel Xeon E3-1585Lv5 CPUs. A single GPU device (NVidia GeForce GTX 1080 Ti) is only used for training the models, but not for evaluation.

F. Additional Results

We include additional experimental results in this section. We visualize the IGC trajectories of Lookahead, NeuralCut and manual heuristics on the four benchmarks in F.1. We test the transferability of NeuralCut models across different domains of integer programs in F.2. We perform model ablations in F.3. We perform ablations on the loss function of the surrogate in F.4.

Table 5. Description of features populating the input's vectors.

feature	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, 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

F.1. Integrality Gap Closed

In Figure F.1, we visualize the IGC trajectories of Lookahead, NeuralCut and common heuristics for cut selection on the four benchmarks Maximum Cut, Packing, Binary Packing and Planning. The NeuralCut models correspond to those in Table 1. All models and heuristics were evaluated on the 500 test instances of the respective domain. 30 separation rounds were performed in each a single cut was added.

F.2. Transferability

In Table 6, we test the transferability of NeuralCut models across different domains of integer programs. For this purpose, each of the four NeuralCut models in Table 1 is additionally evaluated on the test instances of the other three domains. For example, the second-row (packing) fourth-column entry of Table 6 corresponds to the NeuralCut model trained on packing instances, but evaluated on planning test instances. We find that NeuralCut models do not transfer well to other domains: On all benchmarks, the test performance of transferred models is significantly worse than the performance of the model that was trained on instances of the respective domain. This suggests that NeuralCut learns to exploit application-specific patterns to select good cuts.

F.3. Model Ablation

In Table 7, we perform model ablations to better assess the effectiveness of our modelling choices. We begin with a simple GNN model that is based on Gasse et al. (2019), but adapted to cut selection by using cuts in place of constraints in the

Table 6. Training NeuralCut on a single domain (row) does not generalize well to other domains (columns). On all four benchmarks, the model that was trained on the same domain as it is tested on (i.e., main diagonal) performs best. Models that were trained on another domain when they are tested on perform significantly worse. The performance drop tends to be less severe when domains are more similar, e.g., packing and binary packing.

	Reversed IGC integral (↓) on test instances, mean (ste)			
	MAX. CUT	PACKING	BIN. PACKING	PLANNING
MAX. CUT	**15.55 (0.09)**	28.96 (0.03)	26.33 (0.16)	17.07 (0.07)
PACKING	25.03 (0.06)	**26.30 (0.08)**	16.29 (0.27)	29.67 (0.01)
BIN. PACKING	21.75 (0.08)	27.66 (0.06)	**10.96 (0.33)**	25.01 (0.12)
PLANNING	19.13 (0.08)	28.98 (0.03)	22.83 (0.27)	**10.42 (0.04)**

Table 7. Our model choices tend to improve both bound fulfillment (left) and test reversed IGC integral (right) on all four benchmarks. The improvements tend to be clearer for bound fulfillment and binary packing. (Gasse et al., 2019) is the model and features described in (Gasse et al., 2019) and adapted to cut selection by replacing constraints with cuts in the bi-partite graph. For the purpose of this ablation, this base model is first augmented with the features of NeuralCut. Second, structural changes are made to the model by adding the attention module and using batch normalization. Finally, constraints are explicitly modelled in the tripartite graph giving the full NeuralCut model.

	Bound fulfillment (↑) on test samples, mean				**Reversed IGC integral (↓) on test instances**, mean (ste)			
	MAX. CUT	PACKING	BIN. PACKING	PLANNING	MAX. CUT	PACKING	BIN. PACKING	PLANNING
Gasse et al. (2019)	0.90	0.42	0.54	0.95	16.68 (0.10)	27.30 (0.07)	16.09 (0.31)	11.26 (0.06)
+ our features	0.93	0.26	0.66	**0.99**	**15.54 (0.09)**	28.54 (0.04)	12.77 (0.32)	10.66 (0.04)
+ our architecture	0.93	**0.62**	0.76	**1.00**	**15.72 (0.09)**	**26.44 (0.07)**	**10.73 (0.33)**	10.55 (0.04)
NeuralCut (+ our graph)	**0.96**	**0.61**	**0.78**	**1.00**	**15.55 (0.09)**	**26.30 (0.08)**	**10.96 (0.33)**	**10.42 (0.04)**

original model formulation of Gasse et al. (2019) and switching the order in which half-convolutions are applied. Second, we use the features of NeuralCut (+ our features) for both cuts and variables instead of the features of Gasse et al. (2019). Third, we additionally make architectural choices, i.e. we replace layer normalization with batch normalization and add the attention module to the model. Finally, we leverage the tri-partite graph formulation to model constraints explicitly in addition to the cuts and variables, which gives the full NeuralCut model.

F.4. Loss Ablation

In Table 8, we experimented with different loss functions on the bound fulfillment surrogate. In addition to the binary entropy loss (which was used to train all NeuralCut models), we explored the effectiveness of mean squared error (MSE), both with linear and sigmoid activations as well as a cross entropy loss function.

G. Q&A Discussion

Would NeuralCut generalize to different test problems without the need of re-training? What about heterogeneous instances such as those in MIPLIB? The MIPLIB dataset has been developed to benchmark general-purpose solvers, and thus contains diverse instances that vary in size, complexity and structure. ML for MILP may be most promising when used to learn application-specific models that can exploit structural patterns between problem instances arising from a common distribution (Bengio et al., 2021). Indeed, this is the setting on which many recent works in ML for MILP operate[6] and highly relevant in practice. Therefore, even though we demonstrate the effectiveness of *Lookahead* across heterogenous problems (Figure 1), we did not focus on learning models for MIPLIB but trained application-specific ones instead. Regarding generalization, we observed that training on a single domain is unlikely to generalize to other domains. In additional evaluations, we assess how the models trained on the datasets from Tang et al. (2020) perform when used on families different from the one on which they were trained on, and found that performance dropped significantly. This suggests that NeuralCut learns to exploit application-specific patterns to select good cuts.

[6]See also the recent NeurIPS competition `https://www.ecole.ai/2021/ml4co-competition/#datasets`.

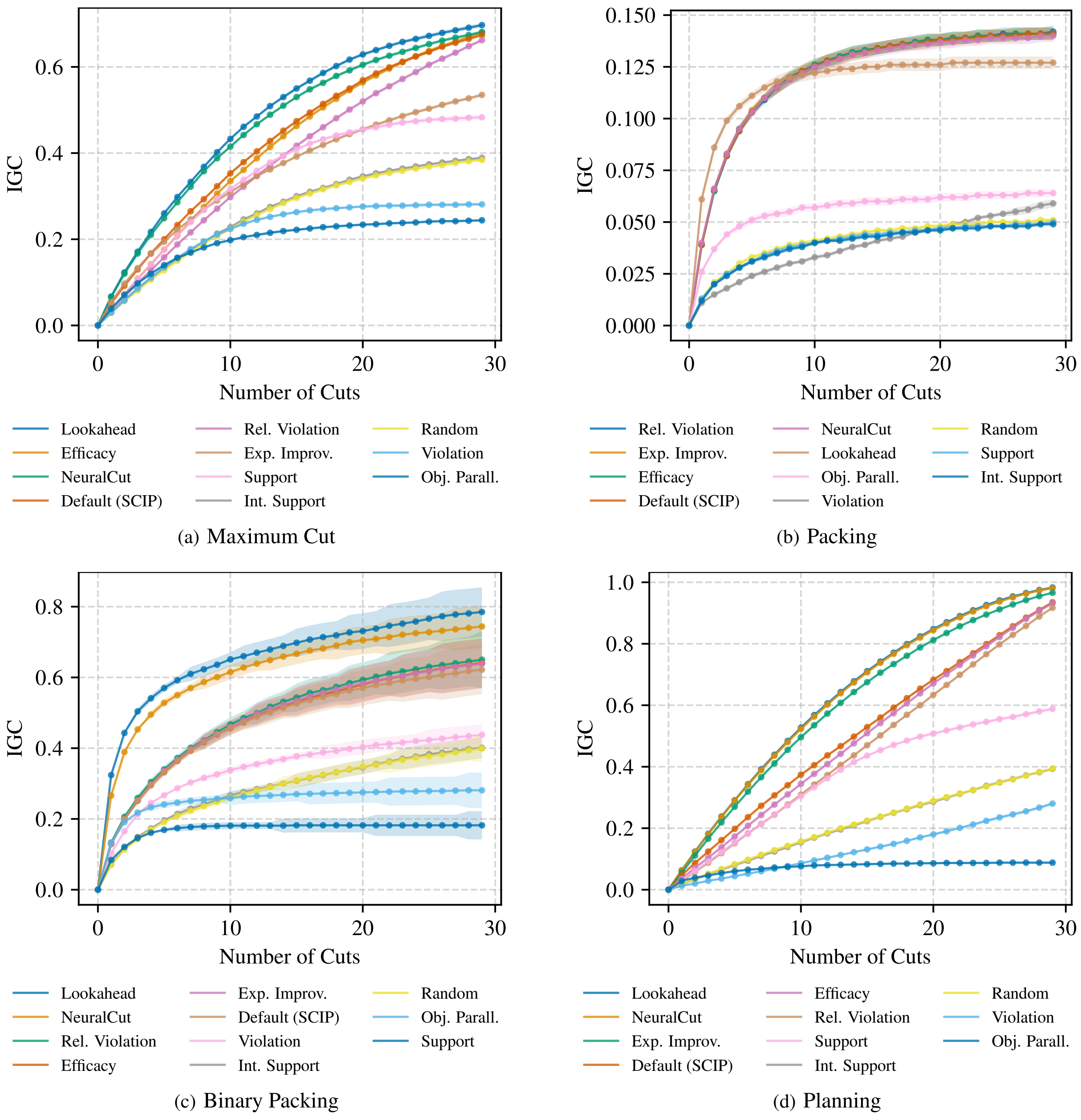
Table 8. Different choices for the training objective (where bound fulfillment is used as surrogate) tend to produce models with comparable test performance. Notable exception are binary packing and the choice of cross entropy for packing. Binary entropy (as was used to train NeuralCut) tends to be a good choice across all four benchmarks. Best are bold-faced.

	Bound fulfillment (↑) on test samples, mean				**Reversed IGC integral (↓) on test instances**, mean (ste)			
	Max. Cut	Packing	Bin. Packing	Planning	Max. Cut	Packing	Bin. Packing	Planning
Binary Ent. (NeuralCut)	**0.96**	**0.61**	**0.78**	**1.00**	**15.55 (0.09)**	**26.30 (0.08)**	**10.96 (0.33)**	**10.42 (0.04)**
MSE (linear act.)	**0.96**	**0.60**	0.70	**0.99**	**15.59 (0.09)**	**26.37 (0.08)**	11.66 (0.33)	10.54 (0.05)
MSE (sigmoid act.)	**0.97**	**0.61**	0.69	**1.00**	**15.63 (0.09)**	**26.30 (0.08)**	12.65 (0.32)	**10.47 (0.04)**
Cross Entropy	**0.96**	0.30	0.73	**1.00**	**15.71 (0.09)**	28.73 (0.04)	**11.36 (0.34)**	10.59 (0.04)

What are the costs of deploying NeuralCut in a realistic MILP solver? While previous work evaluated their models within a naive stripped-down B&C procedure on synthetic instances only, we directly embed our models into SCIP and subject them to the complexity of operating with all the other components of the solver, demonstrating improvements on real-world instances from NN verification. We believe our work is a push forward and that our results are encouraging, as they highlight the potential of NeuralCut to improve MILP solver performance in specific applications. At the same time, it is true that any ML method may suffer from "burn-in cost" for training and potential overhead in deployment. In our setting, the cost of training mostly comes down to the computation of the LA expert decisions. Once trained, though, a single learned policy can be applied to several MILP instances of the same family, effectively amortizing this labeling cost. At deployment, the computational overhead comes from extracting features and performing a forward-pass through the model on CPU. Finally, note that for some difficult problems spending extra time and computational budget at early stages of B&C might payoff in later optimization phases.

Why selecting only one cut at a time? Could you look ahead further then one step? Directly inspired by cutting plane selection in the solver, our model selects a single cut at a time, but it is used to select multiple cuts from the same cutpool by iterative application. Indeed, one could define a lookahead score for k-subsets of cuts (to select multiple cuts at once) or based on a k-step lookahead window (i.e., computing explicit bound improvements after k cut additions). We choose not to, because the cost of computing such a score grows exponentially in k; for this reason, it is also intractable to compute guarantees. While the idea of simultaneously selecting multiple cuts relates to techniques like ranking and multiple instance learning (see, e.g., Huang et al., 2021), the theme of multi-step lookahead has been examined for the task of variable selection in B&B, e.g., by Glankwamdee & Linderoth (2011); Schubert (2017).

Figure 5. Mean test IGC curves for NeuralCut models trained on Maximum Cut, Packing, Binary Packing and Planning instances.


IGC
Number of Cuts
Lookahead
Efficacy
NeuralCut
Default (SCIP)
Rel. Violation
Exp. Improv.
Support
Int. Support
Random
Violation
Obj. Parall.
(a) Maximum Cut
Rel. Violation
Exp. Improv.
Efficacy
Default (SCIP)
NeuralCut
Lookahead
Obj. Parall.
Violation
Random
Support
Int. Support
(b) Packing
Lookahead
NeuralCut
Rel. Violation
Efficacy
Exp. Improv.
Default (SCIP)
Violation
Int. Support
Random
Obj. Parall.
Support
(c) Binary Packing
Lookahead
NeuralCut
Exp. Improv.
Default (SCIP)
Efficacy
Rel. Violation
Support
Int. Support
Random
Violation
Obj. Parall.
(d) Planning