





Machine Learning for Optimization-Based Separation of Mixed-Integer Rounding Cuts

Oscar Guaje ^{1*}, Arnaud Deza ¹, Aleksandr M. Kazachkov ^{2*},
Elias B. Khalil ¹

¹Department of Mechanical & Industrial Engineering, University of Toronto, Toronto, ON, Canada.

²Department of Industrial & Systems Engineering, University of Florida, Gainesville, FL, USA.

*Corresponding author(s). E-mail(s): o.guaje@mail.utoronto.ca;

akazachkov@ufl.edu;

Contributing authors: arnaud.deza@mail.utoronto.ca;

khalil@mie.utoronto.ca;

Abstract

Mixed-integer rounding (MIR) cutting planes (cuts) are effective at improving the strength of a linear relaxation for mixed-integer linear programming (MIP) problems. The cuts in this family are derived by aggregating constraints then rounding coefficients, but finding the strongest MIR cuts requires optimizing a costly MIP for the aggregation step, so in practice, heuristic strategies for separating fractional points are employed. We propose to improve MIR cut generation in the context of a common scenario in applications, where constraints remain fixed but costs are varied. We present a hybrid cut generation framework in which we train a machine learning (ML) model to classify which constraints are involved in useful MIR cuts based on fractional points from relaxations of the problem. At test time, the predictions of the ML model create a reduced MIP-based generator of MIR cuts. In our experiments, we create an instance family from each of three benchmark MIP instances by performing a careful and costly perturbation of objective coefficients to build a dataset of 1,000 fractional points to be separated over the same constraint set. The results indicate that the reduced separator better strengthens the bound in each round of cut generation, particularly for instances in which the full separator failed to find strong cuts.

Keywords: integer programming, machine learning, cutting planes, aggregations, mixed-integer rounding inequalities

1 Introduction

Cutting planes, or cuts, are used by mixed-integer linear programming (MIP) solvers to strengthen formulations and improve running time. Some cut families contain a large number of possible inequalities, leading to heuristic separation strategies and prompting research into optimization-based generation to find the best cuts [2, 6, 9, 13]. In this work, we investigate a learning-based approach to reduce the computational cost of MIP subproblems for generation of *mixed-integer rounding* (MIR) inequalities [19, 21].

An MIR cut is derived via a weighted aggregation of a subset of constraints, followed by appropriate rounding of the resulting base inequality’s coefficients. Selecting which constraints to aggregate is, in practice, based on computationally-inexpensive heuristics [1, 18], which may miss strong MIR cuts obtainable by optimization. However, applications often involve repeatedly solving related optimization problems, which provides an opportunity for inferring helpful structural properties. Specifically, we consider a practitioner that solves a set of MIP instances with the same constraints but varying cost vectors.

We ask the following question: assuming access to samples of a distribution of similar MIP instances, can one *learn* useful constraint selection models from an expensive optimization-based generator for MIR cuts? Our work tests two hypotheses: (i) information about effective cutting planes for a distribution of MIP instances can inform the separation of cutting planes for similar but previously unseen MIP instances; and (ii) an optimization-based generator can be accelerated by carefully fixing to zero some of its decision variables, i.e., eliminating some constraints from consideration, at little sacrifice to the dual bound improvement obtained by the original separator. Our approach can be seen as a hybrid of machine learning (ML) and MIP: an ML model accelerates a MIP-based generator.

We frame the ML problem as supervised binary classification of constraints. The MIP-based separator of Dash, Günlük, and Lodi [9] is executed for a number of rounds on each training instance to produce a portfolio of cuts, tracking which constraints are aggregated to produce each cut. Binary “labels” are assigned to each constraint depending on whether it was selected to generate a good MIR cut or not. Given a MIP instance, a fractional vertex to be separated, and a particular constraint, we devise and compute a set of 54 features that contextualize the relationship between that constraint and the vertex, as well as that constraint and other constraints of the problem. The supervised learning problem is then one of finding an accurate mapping from features to labels. Given an unseen test instance in which a different fractional point is to be cut, the trained ML classifier selects a subset of promising constraints, after which MIP-based MIR separation is restricted only to those constraints.

We test our framework on datasets derived from three benchmark instances from the *2017 Mixed Integer Programming Library* (MIPLIB 2017) [14] that have nontrivial *integrality gaps*, i.e., the objective value of an integer-optimal solution is significantly different than the optimal value after relaxing integrality constraints. We create a large dataset from each base MIPLIB 2017 instance by perturbing the objective function coefficients to find 1,000 different initial fractional solutions that need to be cut off.

Our results indicate that learning a useful classifier is indeed possible and can result in favorable performance compared to running the separator with all constraints.

2 Related Work

In recent years, several papers have explored ML techniques to enhance optimization solvers. In the context of general-purpose MIP solvers, ML has been used to design branching strategies [16] and to decide when and which heuristics to run [7]. As for cutting planes, most work has been on selecting which cuts to add from a pool of cuts [15, 23, 24], while a more recent paper has tackled the problem of deciding whether to generate cuts only at the root node of the branch-and-bound tree [5].

One stream of literature has studied Chvátal-Gomory (CG) cuts, which also involve aggregation of constraints followed by a different rounding step [8]. Balcan et al. [3] study the sample complexity of learning CG constraint aggregation coefficients. They establish lower bounds on the number of samples (instances) needed to accurately estimate expected search tree size resulting from adding CG cuts with specific coefficients. Deza et al. [11] present a framework to predict a set of useful CG coefficients for aggregation using graph neural networks. Becu et al. [4] use the aggregation weights of known instance variations to Gomory mixed integer cuts for other variations of the same instance. Dragotto et al. [12] generate more general split cuts with the help of ML, identifying a split disjunction by a neural network.

A more comprehensive survey of ML for cuts literature was recently completed by Deza and Khalil [10]. We note that ML-based approaches have also been used to exploit problem structures and accelerate the solving process for specific classes of problems (see, e.g., Larsen et al. [17], Xavier et al. [27]).

3 Methodology

In this section, we describe how to optimize over MIR cuts, add in a step to produce a pool of cuts, introduce our data generation procedure, set up our classification model, and explain how this is then given to a reduced MIR generator.

We consider a MIP problem over integer variables $x \in \mathbb{Z}_{\geq 0}^n$ and continuous variables $v \in \mathbb{R}_{\geq 0}^p$ associated to rational objective coefficients f and g , subject to m constraints with rational data A , C , b :

$$\begin{aligned} \min_{x,v} \quad & f^\top x + g^\top v \\ & Ax + Cv = b, \\ & x, v \geq 0, \\ & x \in \mathbb{Z}^n, \\ & v \in \mathbb{R}^p. \end{aligned} \tag{1}$$

Let (x^*, v^*) be a fractional feasible solution to the linear relaxation of (1), obtained by relaxing the integrality restrictions. Our target is to find cuts that are violated by (x^*, v^*) but valid for (1), i.e., satisfied by all integer-feasible points.

3.1 Generation of MIR Cuts

We first provide an overview of MIR inequalities, following the presentation by Dash, Günlük, and Lodi [9], based on [20] and [26]. We then review the optimization-based approach by Dash, Günlük, and Lodi [9] to identifying strong cuts in the family.

3.1.1 Relaxed MIR Inequalities

Let $\lambda \in \mathbb{R}^m$ be a set of multipliers that we treat as a row vector to aggregate the constraints of the MIP (1) into a single constraint

$$\lambda Ax + \lambda Cv = \lambda b. \quad (2)$$

Suppose $\hat{\alpha} \in [0, 1]^n$ and $\bar{\alpha} \in \mathbb{Z}^n$ satisfy $\hat{\alpha} + \bar{\alpha} \geq \lambda A$, where $\hat{\alpha}$ represents the fractional component. Further, let $c^+ \in \mathbb{R}_{\geq 0}^p$ with $c^+ \geq \max\{0, \lambda C\}$. Finally, let $\hat{\beta} \in [0, 1]$ and $\bar{\beta} \in \mathbb{Z}$ such that $\hat{\beta} + \bar{\beta} \leq \lambda b$, in which $\hat{\beta}$ is again the fractional component. Then, for any (x, v) feasible to the linear relaxation of (1), it holds that

$$(\hat{\alpha} + \bar{\alpha})x + c^+v \geq \lambda Ax + \lambda Cv = \lambda b \geq \hat{\beta} + \bar{\beta}.$$

We can then deduce the validity of the following “relaxed” MIR inequality [9, Eq. (11)]:

$$\hat{\alpha}x + \hat{\beta}\bar{\alpha}x + c^+v \geq \hat{\beta}(\bar{\beta} + 1).$$

3.1.2 Separating from the Family of MIR Inequalities

MIR cuts are effective in practice at strengthening a MIP’s linear relaxation [1, 19]. However, as there are infinitely many aggregation vectors λ , each of which yields a valid cut, finding the “best” cut to add is a nontrivial task. One approach to computing a “good” MIR cut is to formulate a MIP to find an aggregation vector that maximizes some measure of violation of a cut by a fractional solution. Given a MIP instance and a fractional solution (x^*, v^*) , Dash, Günlük, and Lodi [9, Eqs. (12)–(18)] propose the following MIP, referred to as Appx-MIR-Sep, to (approximately) search for the MIR cut most violated by (x^*, v^*) :

$$\max_{\lambda, \hat{\alpha}, \bar{\alpha}, c^+, \hat{\beta}, \bar{\beta}, \{\Delta_k, \pi_k\}_{k \in K}, \Delta} \sum_{k \in K} \varepsilon_k \Delta_k - (c^+v^* + \hat{\alpha}x^*) \quad (3a)$$

$$\hat{\alpha} + \bar{\alpha} \geq \lambda A \quad (3b)$$

$$c^+ \geq \lambda C \quad (3c)$$

$$\hat{\beta} + \bar{\beta} \leq \lambda b \quad (3d)$$

$$\hat{\beta} \geq \sum_{k \in K} \varepsilon_k \pi_k \quad (3e)$$

$$\Delta = (\bar{\beta} + 1) - \bar{\alpha}x^* \quad (3f)$$

$$\Delta_k \leq \Delta \quad \forall k \in K \quad (3g)$$

$$\Delta_k \leq \pi_k \quad \forall k \in K \quad (3h)$$

$$c^+ \geq 0 \quad (3i)$$

$$\hat{\alpha}, \hat{\beta} \in [0, 1] \quad (3j)$$

$$\pi \in \{0, 1\}^{|K|} \quad (3k)$$

$$\bar{\alpha}, \bar{\beta} \in \mathbb{Z}. \quad (3l)$$

Since computing the violation of an MIR cut would yield a nonlinear model, Appx-MIR-Sep approximates the violation ε of a cut by assuming it is representable over a set $\mathcal{E} = \{\varepsilon_k = 2^{-k} : k \in K\}$. Any feasible solution to Appx-MIR-Sep can be used to compute an MIR cut to add to the linear relaxation of the problem. The reader is referred to [9] for a more complete account of this model.

What is of interest to us here is that MIR cuts empirically close large gaps and can be separated via a MIP. As cut separation is most useful at the root node of a branch-and-bound tree, the *de facto* procedure for MIP solving, it is rather impractical to solve another complex separation MIP to facilitate solving the original MIP instance. However, since the number of variables in Appx-MIR-Sep depends on the number of constraints in the original MIP, a straightforward way to reduce Appx-MIR-Sep’s computational cost is to reduce the number of constraints to aggregate (i.e., fix some entries of the vector λ to zero *a priori*). This is what we will attempt to do by training an ML classifier that identifies “useful” versus “unimportant” constraints.

3.2 Populating a Pool of Cuts

Given a fractional point, Appx-MIR-Sep seeks the most violated MIR cut. However, violation is one of many cut quality metrics [25]. An arguably more relevant metric, for which violation serves as a tractable but imperfect substitute, is *percent (integrality) gap closed*, defined as the percent of the “gap” between the optimal objective value of the MIP (z_I) and of its LP relaxation (z_{LP}) that is closed by adding cuts (leading to objective value z): $\text{GapClosed} := 100 \cdot (z - z_{LP}) / (z_I - z_{LP})$.

To increase the chance of obtaining MIR cuts with large gap closed via Appx-MIR-Sep, we use the fact that any feasible solution to Appx-MIR-Sep gives a valid cut, and populate a “pool” of cuts using the off-the-shelf capability of modern solvers to collect *multiple* feasible solutions to a MIP. These alternative (and potentially suboptimal for Appx-MIR-Sep) cuts may result in a larger gap closed than the most violated cut. Similarly to the approach by Bonami, Cornuéjols, Dash, Fischetti, and Lodi [6] and Dash, Günlük, and Lodi [9], we populate the cut pool with all the incumbent solutions that the solver finds while solving Appx-MIR-Sep within a given time limit, add all the cuts in the pool to the linear relaxation, get a new fractional solution to separate, and repeat this procedure iteratively until Appx-MIR-Sep is unable to find a separating cut. Algorithm 1 describes the complete cutting loop; lines 3 and 4 are skipped when the classification model is not used, resulting in the benchmark “**full**” **separator** that operates on all constraints.

Algorithm 1: Cutting loop (optionally with ML)

Input: MIP instance P in the form (1); ML model ζ (optional)

```
1 Initialize:  $LP \leftarrow$  LP relaxation of  $P$ ;  $(x^*, v^*) \leftarrow \text{Solve}(LP)$ 
2 while  $x^* \notin \mathbb{Z}^n$  do
3   features $_C \leftarrow \text{get\_features}(P, x^*, v^*)$ 
4    $\Lambda \leftarrow \zeta(\text{features}_C)$ ; /*  $\Lambda$  are constraints  $\zeta$  predicts to be useful */
5    $C \leftarrow \text{Solve Appx-MIR-Sep}(x^*, v^*, \Lambda)$ ; /*  $C$  is a set of cuts */
6   if  $C = \emptyset$  then
7     break
8   else
9      $LP \leftarrow LP \cup C$ ; /* Add cuts to LP */
10     $(\hat{x}, \hat{v}) \leftarrow \text{Solve}(LP)$ 
11    if  $(\hat{x}, \hat{v}) = (x^*, v^*)$  then
12      break
13    else
14       $(x^*, v^*) \leftarrow (\hat{x}, \hat{v})$ 
```

3.3 Learning for MIR Cuts

3.3.1 Data Generation

To conduct supervised learning, one must assume access to a sample over a distribution of similar MIP instances. While these may be available in a variety of application domains or through synthetic generators for specific families of combinatorial problems, we have opted to generate perturbations of instances from the MIPLIB 2017 library of benchmark MIP instances [14]. We will now detail that generation process.

Consider a “base” MIP instance P in the form (1). If the base MIP instance contains inequality constraints, we explicitly add continuous slack variables, and we add variable upper bounds as rows of the constraint matrix. We generate a new instance $\tilde{P} := \min\{\tilde{f}^\top x + \tilde{g}^\top v : Ax + Cv = b, x \in \mathbb{Z}^n, v \in \mathbb{R}^p, x, v \geq 0\}$ by replacing the objective coefficients with random vectors \tilde{f} and \tilde{g} drawn from a truncated normal distribution, then using rejection sampling to ensure that \tilde{P} has a different optimal solution to its linear relaxation compared to P . Specifically, every positive component of (f, g) is replaced by $\max\{0, u^+\}$, where u^+ is normally distributed with mean and variance computed based on all positive components of (f, g) , while every negative component of (f, g) is replaced by $\min\{0, u^-\}$ where u^- is normally distributed with mean and variance computed based on the negative components of (f, g) .

We denote by \mathcal{P} the *instance family* generated from the base instance P , where each member of the family is identified by a different objective vector. For any pair of instances from the same family, Appx-MIR-Sep only differs in the objective function (3a) and in constraint (3f). Following Algorithm 1, for every instance $P \in \mathcal{P}$, we find a solution $(x^*, v^*)^0$ to its linear relaxation. If this solution is not integer-feasible, we solve Appx-MIR-Sep to obtain a set of MIR cuts that we add to the linear relaxation, for which we then compute a new solution $(x^*, v^*)^1$. We repeat these *rounds of cuts* until we find a feasible solution to the original MIP instance, or Appx-MIR-Sep

cannot find a cut that separates the last fractional solution. We generate only *rank-1* cuts, i.e., previously-computed cuts are not used in the derivation of new cuts, so the only difference between Appx-MIR-Sep in different rounds is the objective function (3a) and constraint (3f). Finally, after each run of Appx-MIR-Sep, we record the multiplier vector λ for each cut in the solution pool.

3.3.2 Classification Models

To test both our hypotheses, we train an ML model that uses information about a MIP instance and a fractional solution and predicts a subset of constraints that will be useful in generating MIR cuts. The prediction is made for each constraint of the instance independently.

Each observation in our dataset corresponds to a constraint of an instance \tilde{P} and a round of separation. For example, for an instance family \mathcal{P} with $|\mathcal{P}| = 10$, constructed from a base instance P with $m = 5$ constraints, for which the cutting loop did 2 rounds, the dataset would have $10 \times 5 \times 2 = 100$ observations. We construct a set of features based on previous work on ML for MIP [16], and on measures that are traditionally used to score cuts [25], which we describe in Table 1. These features describe the instance (e.g., through statistics of the cost coefficients), the constraint (e.g., the constraint’s constant side b_i , $1 \leq i \leq m$), and the relationship between the fractional point of interest and the constraint (e.g., the value of the corresponding slack variable, the distance between the constraint’s hyperplane and the point). This results in a set of 54 features that will serve as input to the binary classification model. In Algorithm 1, the features are computed in line 3.

As for the labels, the observation corresponding to constraint i on any round is assigned a positive label if $|\lambda_i| > \epsilon$ for any cut in the cut pool of that round of separation, and a negative label otherwise. This gives us a dataset that can be used for the traditional binary classification task in ML.

3.4 Solving a Reduced Separator

We can use the output of the ML model to predict which constraints will be useful to generate MIR cuts. Then, at each iteration of the cutting loop in Algorithm 1, we compute the features for the current fractional solution on line 3, and use the ML model to classify each constraint on line 4. We use the predicted class for each constraint to update Appx-MIR-Sep and fix $\lambda_i = 0$ for those constraints that the ML model classified in the negative class. We call this the “**reduced**” separator.

4 Computational Experiments

In our experiments, we compare the “reduced” and “full” separators to evaluate how removing constraints that are predicted to be unimportant affects Appx-MIR-Sep.

4.1 Evaluation and Computational Setup

To evaluate the impact of a set of cuts, we use the percent integrality gap closed as defined in Section 3.2. We implement the pipeline described above and test it on

Table 1 Features for the i^{th} constraint $A_i.x + C_i.v = b_i$ and a fractional solution (x^*, v^*) of a MIP.

Feature	Description	Count
Right-hand side (RHS)	The raw value of b_i and a categorical feature if it is nonzero	2
Slack	The raw value of the slack variable and a categorical feature if it is nonzero	2
Dual	The value of the dual variable in the optimal LP solution	1
Degree	The number of variables with nonzero coefficients in the constraint: considering all the variables, only the variables that are nonzero at (x^*, v^*) , only the variables that are zero at (x^*, v^*) , and only the variables that are at their upper bound at (x^*, v^*)	3
Sense	One-hot encoding of the sense of the constraint	2
Stats of the coefficients	Mean, standard deviation, minimum, and maximum of the coefficients: considering all the variables, only the variables that are nonzero at (x^*, v^*) , only the variables that are zero at (x^*, v^*) , and only the variables that are at their upper bound at (x^*, v^*)	16
Stats of the ratios	Mean, standard deviation, minimum, and maximum of the ratios between the coefficients and the RHS: considering all the variables, only the variables that are nonzero at (x^*, v^*) , only the variables that are zero at (x^*, v^*) , and only the variables that are at their upper bound at (x^*, v^*)	16
Euclidean distance to (x^*, v^*)	From [25]	1
Relative violation	From [25]	1
Adjusted distance to (x^*, v^*)	From [25]	1
Objective function parallelism	From [25]	1
Stats of the cost vector	Mean, standard deviation, minimum, and maximum of the cost coefficients of the variables with nonzero coefficients in the constraint	4
	We normalize the absolute value of the cost vector, and compute the number of variables on the top 1, 5, 10, and 20% of the costs that appear with non zero coefficient in the constraint	4
Total count		54

instance families derived from three base instances in the benchmark set of MIPLIB 2017 [14]: **binkar10_1**, **gen-ip054**, and **neos5**.

These three base instances are selected from 37 MIPLIB 2017 “Benchmark” instances for which two conditions are satisfied: (1) random perturbations of the objective function coefficients lead to 1,000 distinct LP relaxation optima; and (2) [Algorithm 1](#) closes at least 5% gap on average across the random variations. We limit our experiments to only three of these 37 potential instances due to the high computational costs of running our experiments and limitations on our shared computational resources: for each of the 1,000 instances in an instance family, the cut generation loop of [Algorithm 1](#) runs for up to three hours. [Table 2](#) provides characteristics on the three chosen instances, which sample some of the diversity of the broader set: **binkar10_1** and **neos5** are mixed integer, whereas **gen-ip054** is pure integer; **binkar10_1** has more than 2,000 variables and 1,000 constraints, whereas the other two are smaller.

We run [Algorithm 1](#) on each of the 1,000 variations from each instance family. We set the time limit to 10 minutes for each individual run of Appx-MIR-Sep, and to three hours for the entire cutting loop per instance. The instances are then split into three disjoint parts: “**training**”, “**test**”, and “**unseen**” sets. The training and test sets are an 80/20 split among instances in which MIR cuts perform well: we discard perturbed instances from each family that have less than 5% gap closed at the end of the cutting loop. [Table 2](#) shows the number of instances in each of the sets.

Table 2 Descriptive statistics for the three instance families.

Family	Constraints	Continuous vars.	Integer vars.	Training	Test	Unseen
<code>binkar10_1</code>	1026	2128	170	478	120	402
<code>gen-ip054</code>	27	0	30	416	104	480
<code>neos5</code>	63	10	53	611	153	236

Training set: For each instance family, we use the corresponding training data to fit a gradient-boosted tree ensemble using `sklearn`’s [22] implementation with default hyperparameters except for the maximum tree depth, which we switch from 3 to 5. Table 3 shows that the trained models perform fairly well as measured by their accuracy, precision (fraction of constraints predicted to be useful that are labeled as useful), and recall (fraction of constraints labeled as useful that are predicted to be useful). For example, for instance `binkar10_1`, 80% of the constraints that the classifier selects to keep have a label of “useful”, and 93% of the constraints with a label of “useful” are correctly classified.

Test set: We next use the trained ML models to reduce the Appx-MIR-Sep problem in each round of separation. Since we know *a priori* that the reduced separator closes more than 5% gap in these variations, this comparison favors the full separator.

Table 3 Trained classifier performance on the training and test sets.

Family	Training			Test		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall
<code>binkar10_1</code>	0.819	0.786	0.928	0.823	0.794	0.926
<code>gen-ip054</code>	0.878	0.839	0.778	0.820	0.674	0.846
<code>neos5</code>	0.940	0.776	0.820	0.936	0.755	0.833

Unseen set: The last set is of instances from each family not considered in the training and test sets, i.e., objective functions that lead to an initial fractional optimal solution from which the eventual gap closed is less than 5% at the end of the cutting loop. Effectively, we investigate if this negative performance is inherent to these instances, i.e., we cannot find good MIR cuts from these starting points, or if the learned “useful” constraint set produces better MIR cuts from Appx-MIR-Sep.

4.2 Computational Results

Figures 1 to 3 show the effect of the reduced separator for each instance family. The three panels correspond to the training set on the left, test set in the middle, and unseen set on the right. The horizontal axis is the round of separation. The bottom subplots show the number of instances for which the cutting loop does not terminate by that round. In the top subplots, the vertical axis is percent integrality gap closed. The solid lines show the average percent gap closed calculated over only the instances

Table 4 Average of number of rounds, number of generated cuts, and seconds to solve Appx-MIR-Sep for each dataset.

Family	Set	Rounds (#)		Cuts (#)		Time per round (s)	
		Full	Reduced	Full	Reduced	Full	Reduced
binkar10_1	Training	24.01	10.51	16.02	16.15	307.24	496.79
	Test	24.48	11.13	16.40	16.31	296.66	501.85
	Unseen	11.87	10.65	15.42	16.43	231.24	492.73
gen-ip054	Training	4.92	4.47	16.14	15.83	279.06	270.32
	Test	4.98	4.30	15.79	15.55	279.62	289.08
	Unseen	2.96	3.48	14.40	14.66	172.85	194.79
neos5	Training	2.79	1.93	5.27	4.61	107.15	226.92
	Test	2.59	1.89	5.25	4.90	103.11	255.45
	Unseen	1.07	1.42	4.93	4.61	0.02	235.40

that require at least that many rounds of the cutting plane loop. This set of surviving instances may differ across the methods, and the behavior is nonmonotone since an instance with a high gap closed no longer contributes to the average after its last round of cuts. The error bars show the standard deviation of the percent gap closed over the surviving instances. The dashed lines show the average percent gap closed across all instances in the family, where the gap closed for an instance that terminates at an earlier round remains constant for all subsequent rounds.

We will also refer to [Table 4](#) for averaged statistics for each instance family, dataset, and method for (1) number of cutting plane rounds, (2) number of generated cuts, and (3) time for cut generation. Generally, the reduced separator uses fewer rounds, generates a comparable number of cuts, and requires more time. We do not focus on cut generation time, as it can be easily skewed: an instance that does no rounds of cuts at all would have a lowest-possible value of zero on that metric.

4.2.1 Instance Family **binkar10_1**

[Figure 1](#) shows the results for instance family **binkar10_1**. We first discuss the training and test sets. The reduced separator tends to close a larger percent of the integrality gap for the instances that survive to a given round (these may not be the same subset across methods). However, as seen in the bottom subplots and in [Table 4](#), the cutting loop with the reduced separator tends to terminate earlier. This means that the reduced separator terminates for many instances at an early round with a small gap closed. The reduced separator may close more gap by that round for those instances, but the full generator continues for more rounds and eventually tends to provide better bounds. This is seen with the dashed lines in the top subplots.

The reduced separator is about 30% smaller than the full separator for this family, for both training and test sets. However, from [Table 4](#), we see that cut generation time is larger on average. Hence, the smaller Appx-MIR-Sep requires more time to solve to optimality. The reduced separator generates around the same number of cuts in total as the full separator, but it obtains these over fewer rounds. Thus, the reduced separator finds more cuts in each round, on average, which may explain the observed larger gap closed (for the surviving instances per round).

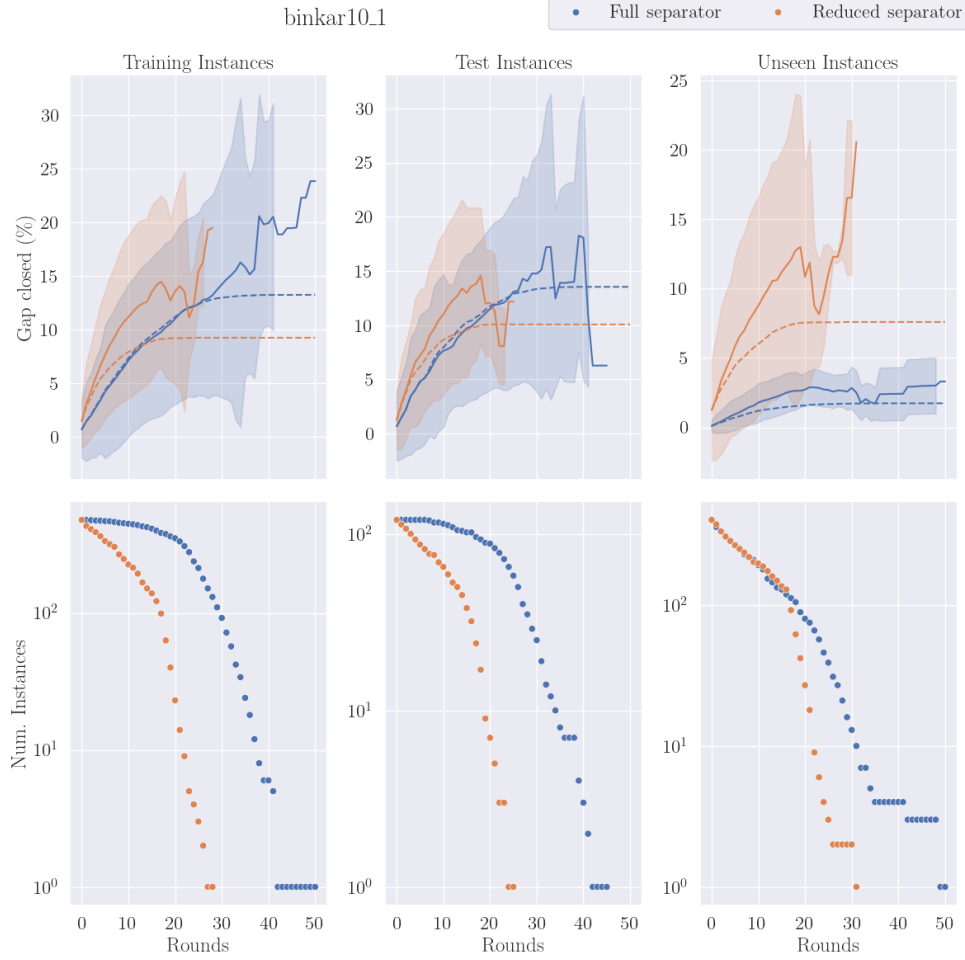


Fig. 1 Effect of reduced separator on instance family `binkar10.1`.

[Table 4](#) does not state the cumulative time spent on cut generation per method. With respect to that metric, for the training and test sets, since the reduced generator requires significantly fewer rounds, it also takes 25–30% less time on average, even though each round is more expensive.

For the unseen set, the reduced separator, using 71% of the constraints, closes more gap than the full separator for all variations, with (slightly) fewer rounds of the cutting loop, but on average Appx-MIR-Sep running time (both per round and, for this set, cumulative) is higher. This additional cost is related to (and justified by) the higher “success” (in terms of cut quality) solving Appx-MIR-Sep for these instances.

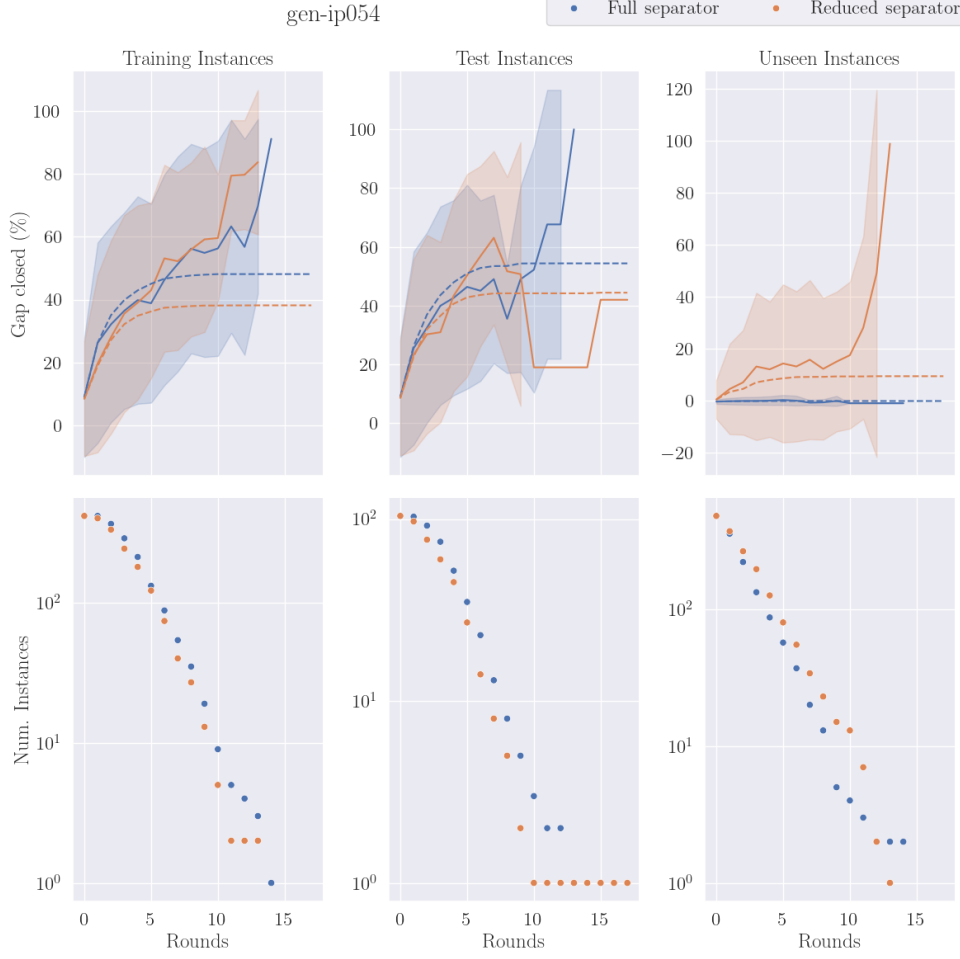


Fig. 2 Effect of reduced separator on instance family **gen-ip054**.

4.2.2 Instance Family **gen-ip054**

Figure 2 shows the results for instance family **gen-ip054**. We again first analyze the training and test sets. Both generators have a similar initial progression of average percent gap closed for surviving instances in these sets. The cutting loop for the reduced separator terminates after slightly fewer iterations in most cases. However, on the instances that do not terminate early, the full separator closes more gap. Around 30% of the constraints are labeled “unimportant” by the classifier for both training and test sets. In contrast to the situation with **binkar10_1**, for this family we observe in Table 4 that there is a slight decrease in time to solve Appx-MIR-Sep for the training set, and a slight increase for the test set, but the values are comparable.

For the unseen set, the reduced separator, using 75% of the constraints, again outperforms the full separator in terms of gap closed (both on average and per round)

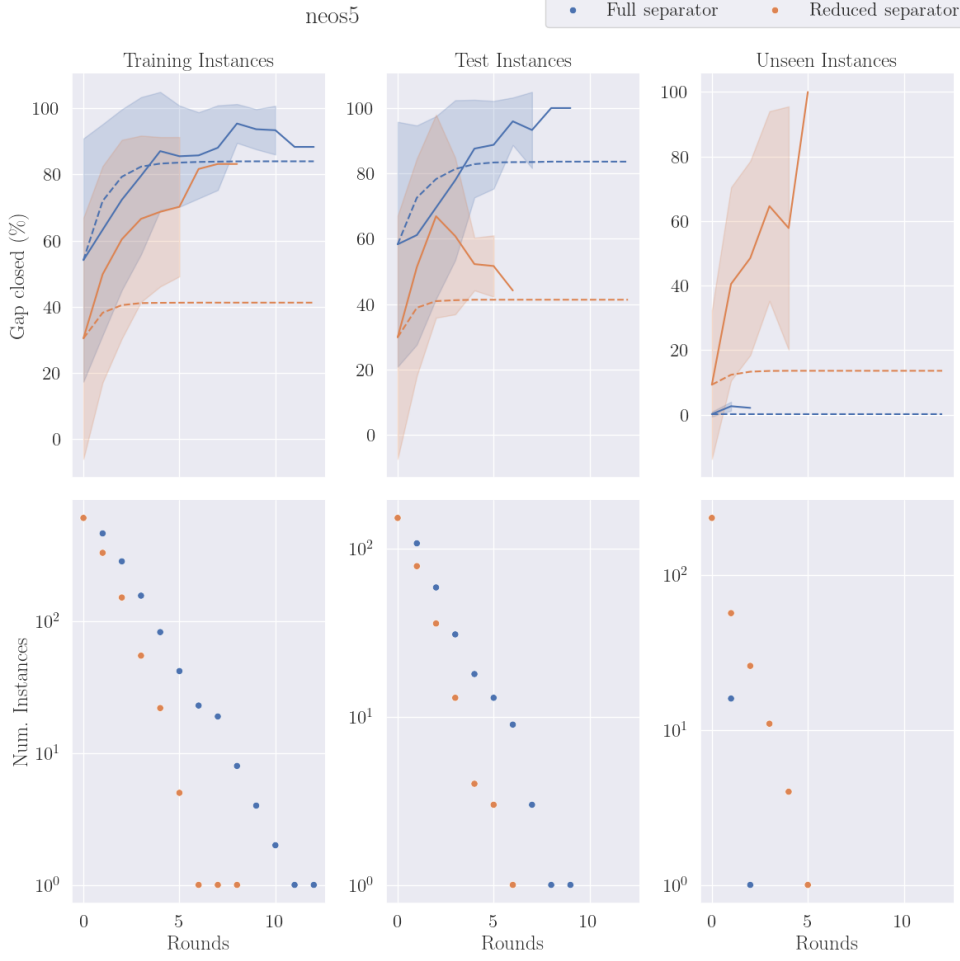


Fig. 3 Effect of reduced separator on instance family **neos5**.

and notably closes 100% of the gap for one instance. The reduced separator also performs more cutting plane rounds on average than the full generator, and each round requires more time, leading to a 30% increase in cumulative time on cut generation.

4.2.3 Instance Family **neos5**

Finally, [Figure 3](#) shows our results for instance family **neos5**. We observe that the full separator closes more gap than the reduced separator. On average, for the training and test sets, only 15% of the constraints are labeled “useful”, but, from [Table 4](#), the average cut generation time (per round), i.e., the average time to prove optimality of Appx-MIR-Sep, is still longer for the reduced separator. For the unseen set, the reduced separator uses 16% of the constraints and consistently outperforms the full separator, closing 100% of the gap for one instance.

5 Conclusion

Takeaway 1: Figures 1 to 3 show that the reduced separator tracks the full separator reasonably well on both training and test instances. Importantly, these instances are ones for which we know that the full separator closes at least 5% of the gap. Since the ML model is trained to imitate the cuts produced by the full separator on the training instances, it is not surprising that the performance of the reduced separator is typically upper bounded by that of the full separator.

Takeaway 2: The results shown for the unseen set in Figures 1 to 3 are especially promising, as they show the potential for incorporating ML into optimization-based separators. When optimizing purely for violation, the total gap closed by the full separator is negligible, but when using the information learned by the ML models, the (reduced) separator generates cuts that close much more gap, the true goal in cut generation.

We have explored the potential of a rather simple binary classification formulation for the problem of reducing the set of constraints that are considered by an optimization-based cut separator. While more sophisticated ML models such as graph neural networks or transformers could be used instead of a gradient-boosted tree ensemble, we have chosen the latter for its simplicity in terms of fitting and amount of training data it requires. We are yet to achieve a significant improvement in the running time of the reduced MIR separator as compared to the full separator. We believe this to be an important step towards operationalizing this hybrid approach. Although we have considered only three MIPLIB 2017-derived instance datasets, we note that generating the training data and evaluating both the full and reduced separators on the training and test instances required hundreds of CPU days in computation. Expanding the experiments to more instance families both from MIPLIB 2017 and other more structured problem classes is of immediate interest.

Acknowledgments Aleksandr M. Kazachkov acknowledges the following funding support: This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-23-1-0340. Elias Khalil acknowledges funding from the Natural Sciences and Engineering Research Council of Canada - Discover Grant Program and from the SCALE AI Research Chair Program. This research was enabled in part by support provided by Compute Ontario (<https://www.computeontario.ca>) and the Digital Research Alliance of Canada (<https://alliancecan.ca>).

Competing Interests The authors have no relevant financial or non-financial interests to disclose.

Data Availability Statement All data supporting the findings of this study are available via the University of Toronto Dataverse at <https://doi.org/10.5683/SP3/KBVV6C>. Code for this paper is at <https://github.com/khalil-research/LearnMIR/>.

References

- [1] Tobias Achterberg. *Constraint Integer Programming*. Doctoral thesis, Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften, Berlin, 2007. URL <http://dx.doi.org/10.14279/depositonce-1634>.
- [2] Egon Balas and Anureet Saxena. Optimizing over the split closure. *Math. Program.*, 113, 2008. URL <http://dx.doi.org/10.1007/s10107-006-0049-5>.
- [3] Maria-Florina Balcan, Siddharth Prasad, Tuomas Sandholm, and Ellen Vitercik. Sample complexity of tree search configuration: Cutting planes and beyond. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 4015–4027. Curran Associates, Inc., 2021. URL <https://dl.acm.org/doi/10.5555/3540261.3540568>.
- [4] Berkay Becu, Santanu S. Dey, Feng Qiu, and Alinson S. Xavier. Approximating the Gomory mixed-integer cut closure using historical data, 2024. URL <https://arxiv.org/abs/2411.15090>.
- [5] Timo Berthold, Matteo Francobaldi, and Gregor Hendel. Learning to use local cuts, 2022. URL <https://arxiv.org/abs/2206.11618>.
- [6] Pierre Bonami, Gérard Cornuéjols, Sanjeeb Dash, Matteo Fischetti, and Andrea Lodi. Projected Chvátal–Gomory cuts for mixed integer linear programs. *Math. Program.*, 113(2):241–257, June 2008. URL <https://doi.org/10.1007/s10107-006-0051-y>.
- [7] Antonia Chmiela, Elias B. Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. *Advances in Neural Information Processing Systems*, 34:24235–24246, 2021. URL <https://dl.acm.org/doi/10.5555/3540261.3542116>.
- [8] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 306(10):886–904, 2006. URL <https://doi.org/10.1016/j.disc.2006.03.009>. 35th Special Anniversary Issue.
- [9] Sanjeeb Dash, Oktay Günlük, and Andrea Lodi. MIR closures of polyhedral sets. *Math. Program.*, 121(1, Ser. A):33–60, January 2010. URL <http://dx.doi.org/10.1007/s10107-008-0225-x>.
- [10] Arnaud Deza and Elias B. Khalil. Machine learning for cutting planes in integer programming: a survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI ’23*, pages 6592–6600, 2023. URL <https://doi.org/10.24963/ijcai.2023/739>.
- [11] Arnaud Deza, Elias B. Khalil, Zhenan Fan, Zirui Zhou, and Yong Zhang. Learn2aggregate: Supervised generation of Chvátal–Gomory cuts using graph neural networks, 2024. URL <https://arxiv.org/abs/2409.06559>.
- [12] Gabriele Dragotto, Stefan Clarke, Jaime Fernández Fisac, and Bartolomeo Stellato. Differentiable cutting-plane layers for mixed-integer linear optimization, 2023. URL <https://arxiv.org/abs/2311.03350>.
- [13] Matteo Fischetti and Andrea Lodi. Optimizing over the first Chvátal closure. *Math. Program.*, 110(1):3–20, 2007. URL <https://doi.org/10.1007/s10107-006-0054-8>.

- [14] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelman, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Math. Program. Comput.*, 2021. URL <https://doi.org/10.1007/s12532-020-00194-3>.
- [15] Zeren Huang, Kerong Wang, Furui Liu, Hui-Ling Zhen, Weinan Zhang, Mingxuan Yuan, Jianye Hao, Yong Yu, and Jun Wang. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognit.*, 123:108353, 2022. URL <https://doi.org/10.1016/j.patcog.2021.108353>.
- [16] Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016. URL <https://ojs.aaai.org/index.php/AAAI/article/view/10080>.
- [17] Eric Larsen, Sébastien Lachapelle, Yoshua Bengio, Emma Frejinger, Simon Lacoste-Julien, and Andrea Lodi. Predicting tactical solutions to operational planning problems under imperfect information. *INFORMS J. Comput.*, 34(1): 227–242, 2022. URL <https://doi.org/10.1287/ijoc.2021.1091>.
- [18] Andrea Lodi. The heuristic (dark) side of MIP solvers. In *Hybrid Metaheuristics*, pages 273–284. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. URL https://doi.org/10.1007/978-3-642-30671-6_10.
- [19] Hugues Marchand and Laurence A. Wolsey. Aggregation and mixed integer rounding to solve MIPs. *Oper. Res.*, 49(3):363–371, 2001. URL <https://doi.org/10.1287/opre.49.3.363.11211>.
- [20] George L. Nemhauser and Laurence A. Wolsey. A recursive procedure to generate all cuts for 0-1 mixed integer programs. *Math. Program.*, 46(3):379–390, 1990. URL <https://doi.org/10.1007/BF01585752>.
- [21] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*, volume 55. John Wiley & Sons, New York, 1999.
- [22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011. URL <https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html>.
- [23] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 9367–9376. PMLR, 2020. URL <http://proceedings.mlr.press/v119/tang20a.html>.
- [24] Mark Turner, Thorsten Koch, Felipe Serrano, and Michael Winkler. Adaptive Cut Selection in Mixed-Integer Linear Programming. *Open J. Math. Optim.*, 4: 5, 2023. URL <https://doi.org/10.5802/ojmo.25>.
- [25] Franz Wesselmann and Uwe H. Suhl. Implementing cutting plane management and selection techniques. Technical report, University of Paderborn, 2012.

- [26] Laurence A. Wolsey. *Integer programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., New York, 1998.
- [27] Álinson S. Xavier, Feng Qiu, and Shabbir Ahmed. Learning to solve large-scale security-constrained unit commitment problems. *INFORMS J. Comput.*, 33(2): 739–756, 2021. URL <https://doi.org/10.1287/ijoc.2020.0976>.