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

Oscar Guaje[1*], Arnaud Deza[1], Aleksandr M. Kazachkov[2], Elias B. Khalil[1]

[1]Department of Mechanical & Industrial Engineering, University of Toronto, Toronto, ON, Canada.
[2]Department of Industrial & Systems Engineering, University of Florida, Gainesville, FL, USA.

*Corresponding author(s). E-mail(s): o.guaje@mail.utoronto.ca;
Contributing authors: arnaud.deza@mail.utoronto.ca;
akazachkov@ufl.edu; khalil@mie.utoron.ca;

Abstract

Mixed-Integer Rounding (MIR) cuts are effective at improving the dual bound in Mixed-Integer Linear Programming (MIP). However, in practice, MIR cuts are separated heuristically rather than using optimization as the latter is prohibitively expensive. We present a hybrid cut generation framework in which we train a Machine Learning (ML) model to inform cut generation for a family of similar instances. Our framework solves a MIP-based separation problem to generate high-quality MIR cuts, then learns to identify useful constraints that led to these effective cuts. At test time, the predictions of the ML model allow us to solve a reduced MIP-based separation problem. We present computational results for this approach on datasets of randomly perturbed MIPLIB2017 instances.

Keywords: Integer Programming, Machine Learning, Cutting Planes

1 Introduction

Cutting planes are a critical component of Mixed-Integer Linear Programming (MIP) solvers. There are many families of cutting planes whose mathematical properties

have been studied extensively and their effectiveness analyzed both in theory and practice. One stream of research has explored the use of optimization for the cut separation problem [1–4]. For example, for Mixed-Integer Rounding (MIR) cuts [5, 6], the family of interest in this work, a cut is obtained by a weighted aggregation of a subset of constraints, followed by appropriate rounding of the resulting base inequality's coefficients. While finding cuts that maximally separate the vertex of interest seems like a sensible choice, optimization-based separators are seldom used in practice due to their prohibitive computational cost. Instead, MIP solvers often rely on computationally inexpensive heuristics to generate cuts [7, 8].

In this work, 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 separator for MIR cuts? Our work aims to test 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; (ii) an optimization-based separator 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 separator.

To that end, we frame the ML problem as one of supervised binary classification of constraints. We opt for an experimental design that uses a few MIPLIB2017 instances that have non-trivial integrality gaps. Each of these MIPLIB2017 instances is the basis for an entire dataset obtained by randomizing the objective function coefficients to find a large number of different initial fractional solutions that need to be cut off. The MIP-based separator of Dash et al. [4] is executed for a number of rounds on each training instance from the randomized set of instances, and 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 (i.e., a different point to cut off), the trained ML classifier selects a subset of promising constraints and MIP-based MIR separation is restricted only to those constraints. On three MIPLIB2017-derived instance datasets, we show that learning such a classifier is indeed possible and can result in favorable performance compared to running the separator with all constraints.

2 Related Work

In recent years, ML techniques have been used to enhance optimization solvers to great success. In the context of general-purpose MIP solvers, ML has been used to design branching strategies [9], and to decide when and which heuristics to run [10]. As for cutting planes, most work has been on selecting which cuts to add from a pool of cuts [11–13], while a more recent paper has tackled the problem of deciding

whether to generate cuts only at the root node of the B&B tree. Balcan et al. [14] study the sample complexity of learning Chvátal-Gomory (CG) constraint aggregation coefficients. Lower bounds are derived for the number of instances on the same set of variables and constraints with randomly varying constraint matrices and right-hand side vectors that must be observed to accurately estimate the expected search tree size resulting from adding cuts that use a specific set of CG coefficients. As for ML for cut generation, Dragotto et al. [15] present a cutting plane algorithm that generates split cuts based on a disjunction given by the output of a neural network. A more comprehensive survey of ML for cuts literature was recently completed by Deza and Khalil [16]. We note that ML-based approaches have also been used to exploit problem structures and accelerate the solving process for specific classes of problems [17, 18].

3 Methodology

3.1 Separation of MIR cuts

Consider a MIP problem over integer variables x and continuous variables v:

$$\min_{x,v}\{f^\intercal x + g^\intercal v : Cv + Ax = b,\, x, v \geq 0,\, x \in \mathbb{Z}^n,\, v \in \mathbb{R}^p\}. \tag{1}$$

A Mixed-Integer Rounding (MIR) cut is an inequality of the form

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

where $c^+ \geq \lambda C$, $c^+ \geq 0$, $\hat{\alpha} + \bar{\alpha} \geq \lambda A$, $\hat{\beta} + \bar{\beta} \leq \lambda b$, $\bar{\alpha} \in \mathbb{Z}^n$, $\bar{\beta} \in \mathbb{Z}$, $0 \leq \hat{\alpha}, \hat{\beta} \leq 1$ for some vector λ [4]. The vector λ can be interpreted as a set of multipliers that aggregate the constraints of the MIP problem into a single inequality. MIR cuts have been shown to be effective in strengthening the linear relaxation of MIP [6, 7]. However, as there are infinitely many aggregation vectors λ, each of which would result in a valid cut, finding the "best" cut to add is a non-trivial 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 $(x^\star, v^\star)$, Dash et al. [4] propose the following MIP, referred to as MIR-Sep, to obtain the MIR cut most violated by $(x^\star, v^\star)$:

$$\max \quad \sum_{k \in K} \varepsilon_k \Delta_k - \left(c^+ v^\star + \hat{\alpha} x^\star\right) \tag{2a}$$

$$\text{s.t.} \quad c^+ \geq \lambda C \tag{2b}$$

$$\hat{\alpha} + \bar{\alpha} \geq \lambda A \tag{2c}$$

$$\hat{\beta} + \bar{\beta} \leq \lambda b \tag{2d}$$

$$c^+ \geq 0 \tag{2e}$$

$$0 \leq \hat{\alpha} \leq 1 \tag{2f}$$

$$0 \leq \hat{\beta} \leq 1 \tag{2g}$$

$$\hat{\beta} \geq \sum_{k \in K} \varepsilon_k \pi_k \tag{2h}$$

$$\Delta = \left(\bar{\beta} + 1\right) - \bar{\alpha} x^\star \tag{2i}$$

$$\Delta_k \leq \Delta \qquad \forall\ k \in K \tag{2j}$$

$$\Delta_k \leq \pi_k \qquad \forall\ k \in K \tag{2k}$$

$$\pi \in \{0,1\}^{|K|} \tag{2l}$$

$$\bar{\alpha} \in \mathbb{Z}^n \tag{2m}$$

$$\bar{\beta} \in \mathbb{Z}. \tag{2n}$$

Since computing the violation of an MIR cut would yield a nonlinear model, this model is an approximation that underestimates the violation ε of a cut, by assuming it to be a number representable over a set $\mathcal{E} = \{\varepsilon_k = 2^{-k} : k \in K\}$. Any feasible solution to MIR-Sep can be used to compute an MIR cut to add to the linear relaxation of the problem. The reader is referred to [4] 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 exactly 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 MIR-Sep depends on the number of constraints in the original MIP, a straightforward way to reduce 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 disposable constraints.

3.2 Populating a Pool of Cuts

Given a fractional point, MIR-Sep finds the most violated MIR cut. However, violation is only one cut quality metric (see [19] for more), and the most violated cut does not necessarily close the most gap. The gap closed is defined in [1, 2] as $\text{GapClosed} = 100\left(\frac{z - z_{LP}}{z_I - z_{LP}}\right)$, where z_I, z_{LP}, and z represent the optimal objective values of the original MIP instance, its LP relaxation, and the LP relaxation with added cuts, respectively. Then, to increase the chance of obtaining MIR cuts with large gap closed via MIR-Sep, we use the fact that any feasible solution to 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 suboptimal (for MIR-Sep) cuts may result in a larger gap closed than the most violated cut. Similarly to the approach in [2] and [4], we populate the cut pool with all the incumbent solutions that the solver finds while solving MIR-Sep, add all the cuts in the pool to the linear relaxation, get a new fractional solution to separate, and repeat this procedure iteratively until MIR-Sep is unable to find a separating cut. Algorithm 1 describes the complete cutting loop; lines 8 and 9 are skipped when the classification model is not used, resulting in the "full" separator that operates on all constraints.

Algorithm 1: Cutting loop (optionally with ML)

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

$LP \leftarrow$ The LP relaxation of the input instance
$(x^\star, v^\star) \leftarrow \text{Solve}(LP)$
$cutting \leftarrow$ `true`
while *cutting* **do**
  if $x^\star \in \mathbb{Z}^n$ **then**
    $cutting \leftarrow$ `false`
  else
    $\texttt{features}_{\mathcal{C}} \leftarrow \texttt{get\_features}(P, x^\star, v^\star)$
    $\Lambda \leftarrow \zeta(\texttt{features}_{\mathcal{C}})$; `/* Λ are constraints ζ predicts to be useful */`
    $\mathcal{C} \leftarrow$ Solve MIR-Sep$(x^\star, v^\star, \Lambda)$; `/* C is a set of cuts */`
    if $\mathcal{C} = \emptyset$ **then**
      $cutting \leftarrow$ `false`
    else
      $LP \leftarrow LP \cup \mathcal{C}$; `/* Add cuts to LP */`
      $(\hat{x}, \hat{v}) \leftarrow \text{Solve}(LP)$
      if $(\hat{x}, \hat{v}) = (x^\star, v^\star)$ **then**
        $cutting \leftarrow$ `false`
      else
        $(x^\star, v^\star) \leftarrow (\hat{x}, \hat{v})$
      end
    end
  end
end

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 MIPLIB2017 library of benchmark MIP instances [20]. We will now detail that generation process.

Consider a MIP instance P in the form (1) and let $d = [f; g]$ be the vector containing all the cost coefficients. If the original 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 $\tilde{P} : \min\{\tilde{f}^\intercal x + \tilde{g}^\intercal v : Cv + Ax = b, x \in \mathbb{Z}^n, v \in \mathbb{R}^p, x, v \geq 0\}$ such that the optimal solution to the linear relaxation of P and $\tilde{P}$ are different by generating random vectors $\tilde{f}$ and $\tilde{g}$. Let $d^{(+)}$ and $d^{(-)}$ be the positive and negative components of d, respectively; then $\tilde{d}^{(+)} = \min\{0, u\}$, where u is a random number drawn from a normal distribution with

mean $\mu_{d^{(+)}}$ and standard deviation $\sigma_{d^{(+)}}$; and $\tilde{d}^{(-)} = \max\{0, u\}$, where u is a random number drawn from a normal distribution with mean $\mu_{d^{(-)}}$ and standard deviation $\sigma_{d^{(-)}}$.

We can repeat this procedure an arbitrary number of times until we populate a set $\mathcal{P}$ of instances. We denote as $\mathcal{P}$ the *instance family* generated from P. Note that MIR-Sep for any pair of variations $P_i, P_j \in \mathcal{P}$ only differs in the objective function (2a) and in constraint (2i). Now, as per Algorithm 1, for every instance $P \in \mathcal{P}$, we solve its linear relaxation to obtain a fractional solution $(x^\star, v^\star)^0$. We then solve MIR-Sep to obtain a set of MIR cuts that we add to the linear relaxation to obtain a new solution $(x^\star, v^\star)^1$. If this solution is fractional, we update MIR-Sep and solve it to obtain a new set of cuts. We repeat this procedure until we find a feasible solution to the original MIP instance, or MIR-Sep cannot find a cut that separates the last fractional solution. For simplicity, we consider only rank-1 cuts, so the only difference between MIR-Sep for rounds i and $i+1$ is the objective function (2a) and constraint (2i). Finally, after each run of 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. Each observation in our dataset corresponds to a constraint of a variation $\tilde{P}$ and a round of separation, e.g., for a set of variations $\mathcal{P}$ with $|\mathcal{P}| = 10$, constructed from a base instance P with 5 constraints, for which the cutting loop did 2 rounds, the dataset would have $10 \times 5 \times 2 = 100$ observations. We constructed a set of features based on previous work on ML for MIP [9], and on measures that are traditionally used to score cuts [19], which we describe in Table 1. These features describe the instance itself (e.g., through statistics of the cost coefficients), the constraint itself (e.g., the right-hand side value b_j), 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 8.

As for the labels, the observation corresponding to constraint j on any round is assigned a positive label if $|\lambda_j| > \epsilon$ 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 8, and use the ML model to classify each constraint on line 9. We use the predicted class for each constraint to update MIR-Sep and fix $\lambda_j = 0$ for those constraints that the ML model classified in the negative class.

Table 1 Features for the j^{th} constraint $C_j v + A_j x = b_j$ and a fractional solution $(x^\star, v^\star)$ of a MIP.

Feature	Description	Count
Right-hand side (RHS)	The raw value of b_j and a categorical feature if it is non-zero	2
Slack	The raw value of the slack variable and a categorical feature if it is non-zero	2
Dual	The value of the dual variable in the optimal LP solution	1
Degree	The number of variables with non-zero coefficients in the constraint: considering all the variables, only the variables that are nonzero at $(x^\star, v^\star)$, only the variables that are zero at $(x^\star, v^\star)$, and only the variables that are at their upper bound at $(x^\star, v^\star)$	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 $(x^\star, v^\star)$, only the variables that are zero at $(x^\star, v^\star)$, and only the variables that are at their upper bound at $(x^\star, v^\star)$	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 $(x^\star, v^\star)$, only the variables that are zero at $(x^\star, v^\star)$, and only the variables that are at their upper bound at $(x^\star, v^\star)$	16
Euclidean distance to $(x^\star, v^\star)$	From [19].	1
Relative violation	From [19].	1
Adjusted distance to $(x^\star, v^\star)$	From [19].	1
Objective function parallelism	From [19].	1
Stats of the cost vector	Mean, standard deviation, minimum, and maximum of the cost coefficients of the variables with non-zero 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

4 Experimental Results

4.1 Setup and Model Training

To evaluate the impact of a set of cuts, we use the percentage of gap closed as defined in Section 3.2. We implemented the pipeline described above and tested it on instance families derived from three instances in the benchmark set of MIPLIB2017 [20]: `binkar10_1`, `genip0054`, and `neos5`.

These three "base instances" were selected out of 37 MIPLIB2017 "Benchmark" instances for which two conditions are satisfied: (1) random perturbations of the objective function coefficients lead to 1,000 distinct LP relaxation optima; (2) Algorithm 1 closes at least 5% gap on average across the random variations. The three base instances used here were selected on the basis of diversity. As can be seen in Table 2, `binkar10_1` and `neos5` are mixed-integer whereas `genip0054` is pure; `binkar10_1` has more than 2,000 variables and 1,000 constraints, whereas the two other base instances are smaller. While additional MIPLIB2017-based datasets could be considered for experimentation, we note that for each of the 1,000 instances, the cut generation loop of Algorithm 1 runs for up to three hours. As we compare the full and reduced separators on each instance, this necessitates thousands of CPU hours (or roughly one CPU

year) per dataset. We conducted what we believe is a sufficient amount of experiments within the computational resources available to us.

For each instance family, we ran Algorithm 1 on 1,000 variations, and set the time limit to 10 minutes for each individual run of MIR-SEP, and three hours for all the cutting loop. After running the cutting loop, we discarded (for the first set of experiments that will be discussed next) perturbed instances that had less than 5% gap closed, and split the rest into training and test sets for machine learning. Table 2 shows the number of variables/constraints and perturbations in each of the training and test sets for the three instance families.

Table 2 Descriptive statistics for the three instance sets.

Instance set	Constraints	Continuous vars.	Integer vars.	Training	Test
`binkar10_1`	1026	2128	170	478	120
`genip0054`	27	0	30	416	104
`neos5`	63	10	53	611	153

For each instance set, we used the corresponding training data to fit a gradient-boosted tree ensemble using `sklearn`'s [21] implementation with default hyperparameters except for increasing the maximum tree depth from 3 to 5. Table 3 shows that the trained models are fairly accurate as measured by the raw accuracy, the precision (fraction of constraints predicted to be useful that are labeled as useful), and the recall (fraction of constraints labeled as useful that are predicted to be useful) on the training set.

Table 3 Classification performance of the trained classifiers on the training set.

Instance set	Accuracy	Precision	Recall
`binkar10_1`	0.819	0.786	0.928
`genip0054`	0.878	0.839	0.778
`neos5`	0.940	0.776	0.820

4.2 Reducing the Separation Problem using the ML Classifier

We next use the output of the ML models to reduce the MIR-Sep problem on each round of separation. Since we know a priori that the reduced separator closes more than 5% gap in these variations, this is a more favorable comparison for the full separator. Figures 1, 3 and 5 show the behavior of our approach for each instance family. The subplots on the left- and right-hand sides show the train and test instances, respectively. The subplots on the top show the gap closed with respect to the number of rounds of separation, and the subplots at the bottom show the number of instance variations for which the cutting loop had not terminated at each round.

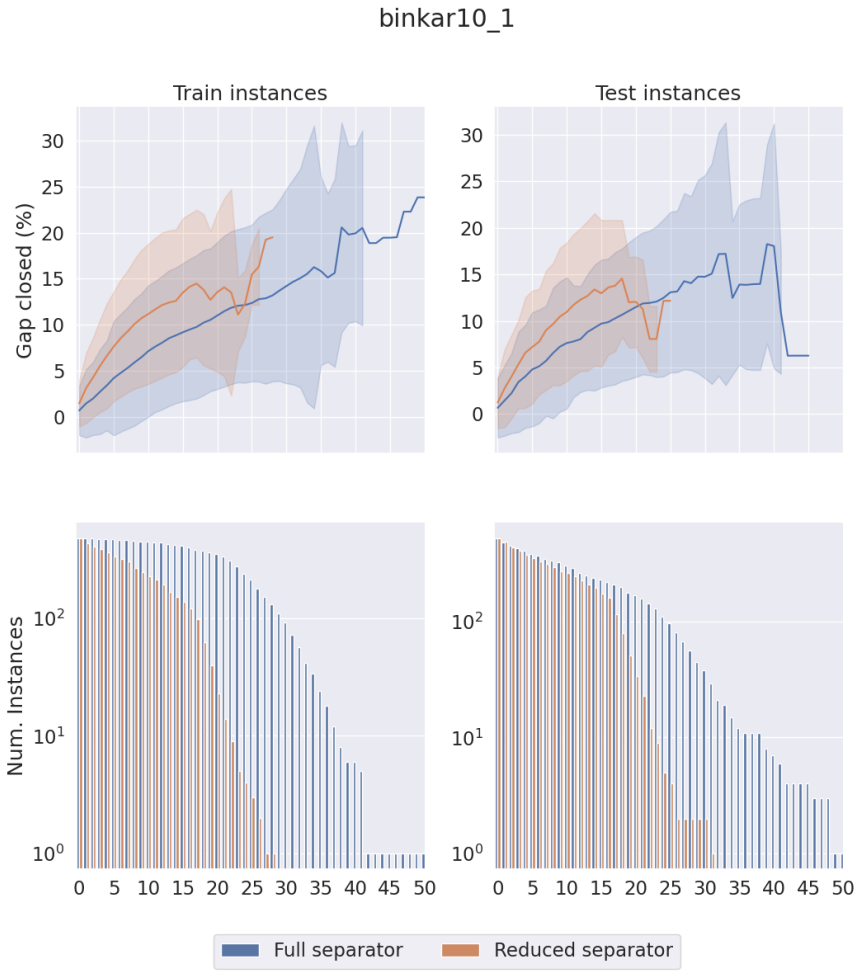

Fig. 1 Behavior on base instance `binkar10_1`. Only variations where the full separator produces more than 5% gap closed were considered. The horizontal axis represents the round of separation. On the top plots the vertical axis represents gap closed, with the lines showing the mean gap closed, and the error bars showing its standard deviation. On the bottom plots the vertical axis represents the number of variations that made it to each round.

Recall that to construct our datasets so far, we discarded some instance variations for which the total gap closed by running the cutting loop with the full separator was less than 5%. We ran the cutting loop with the reduced separator on these instances. Figures 2, 4 and 6 show the average gap closed and the number of instances that had not terminated after each round of the cutting loop for only these discarded variations.

4.2.1 Instance family `binkar10_1`

Figure 1 shows the results for instance family `binkar10_1`. The gap closed by both separators on the train instances behaves similarly in earlier rounds. However, the

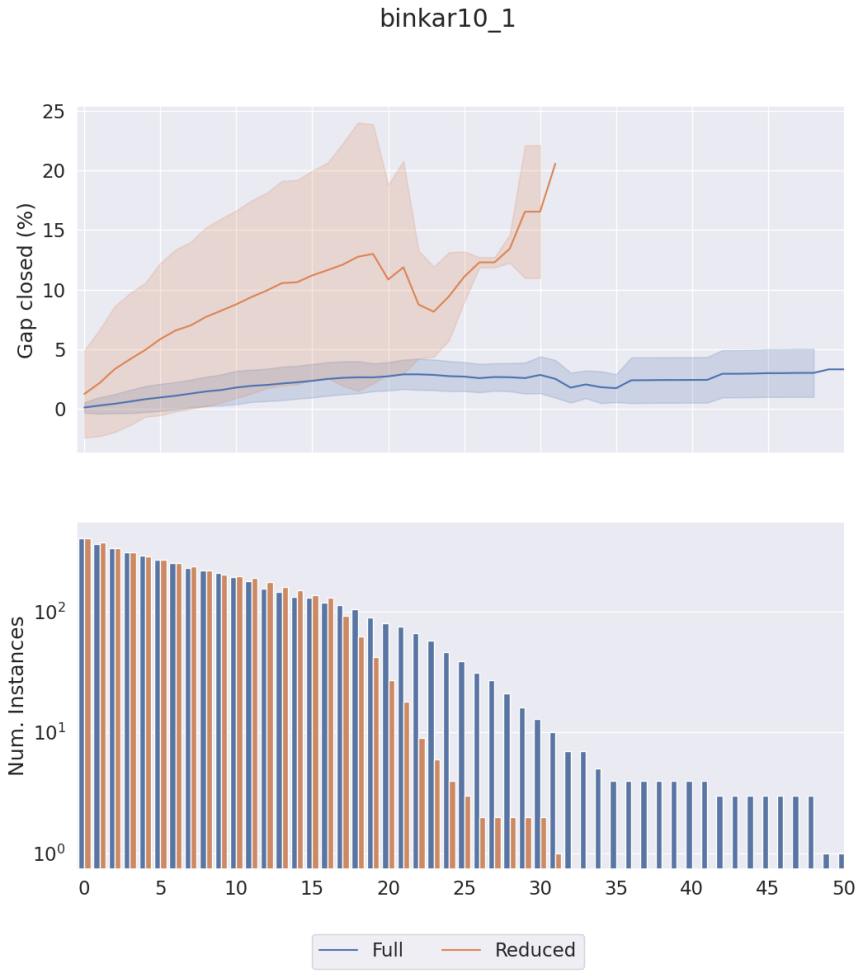

Fig. 2 Behavior on base instance `binkar10_1`. Only variations where the full separator produces less than 5% gap closed were considered. The horizontal axis represents the round of separation. On the top plot the vertical axis represents gap closed, with the lines showing the mean gap closed, and the error bars showing its standard deviation. On the bottom plot the vertical axes represent the number of variations that made it to each round.

cutting loop terminates sooner when using the reduced separator, which results in less gap closed by the end of the cutting loop. As for the test set, the reduced separator still terminates after fewer rounds, but the gap closed is larger on average.

On the second experiment, shown in figure 2, the reduced separator outperformed the full separator for all variations, and the cutting loop terminated in fewer rounds.

4.2.2 Instance family `gen-ip054`

Figure 3 shows the results for instance family `gen-ip054`. The behavior of both separators is fairly similar. On the test set, the cutting loop for the reduced separator

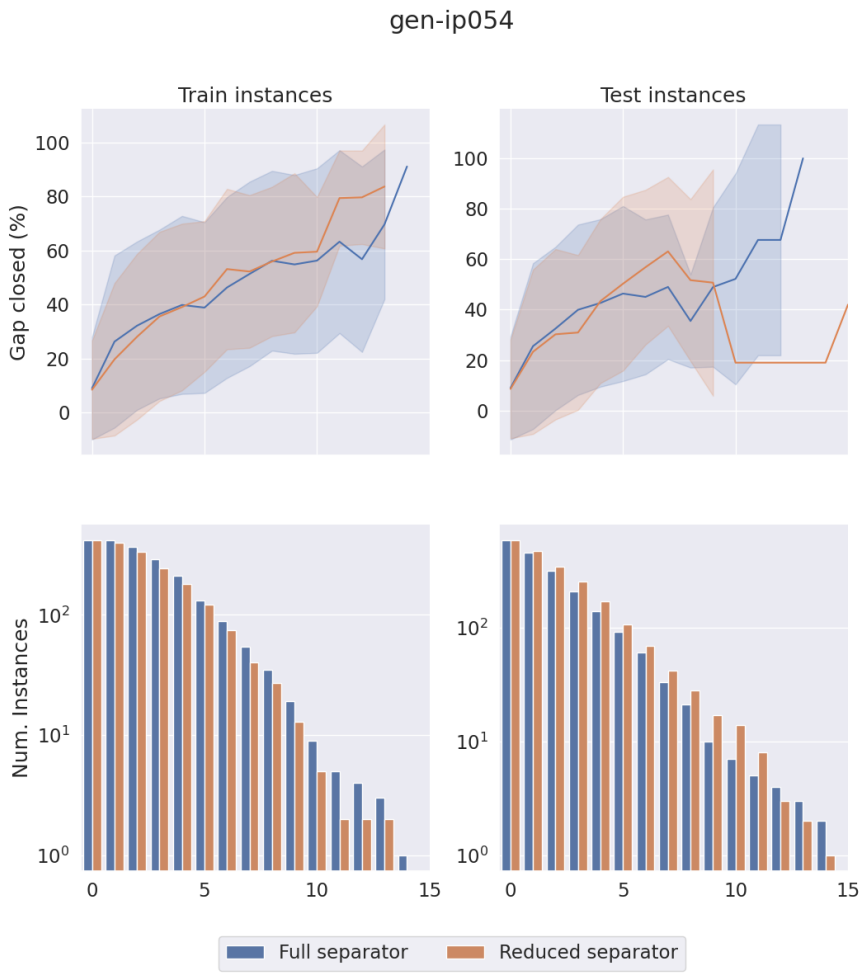

Fig. 3 Behavior on base instance `gen-ip054`. Only variations where the full separator produces more than 5% gap closed were considered. The horizontal axis represents the round of separation. On the top plots the vertical axis represents gap closed, with the lines showing the mean gap closed, and the error bars showing its standard deviation. On the bottom plots the vertical axis represents the number of variations that made it to each round.

terminates after fewer iterations in most cases. However, on the instances that do not terminate early, the full separator closes more gap.

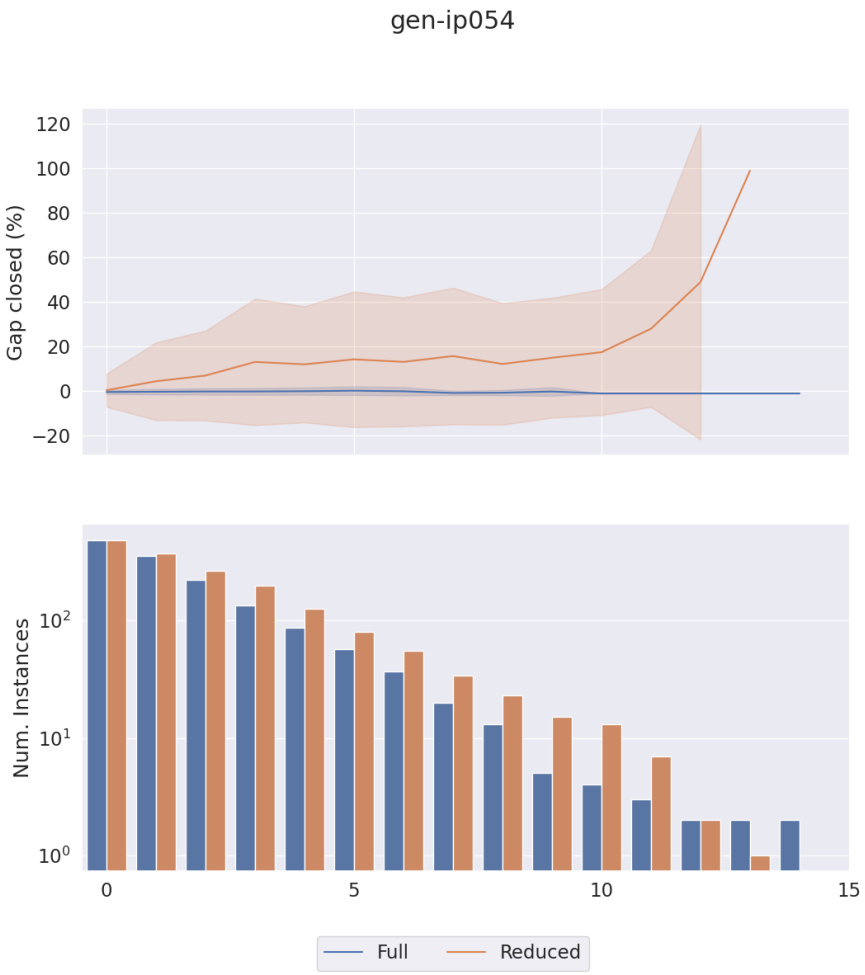

Fig. 4 Behavior on base instance `gen-ip054`. Only variations where the full separator produces less than 5% gap closed were considered. The horizontal axis represents the round of separation. On the top plot the vertical axis represents gap closed, with the lines showing the mean gap closed, and the error bars showing its standard deviation. On the bottom plot the vertical axis represents the number of variations that made it to each round.

On the second experiment, shown in figure 4, the reduced separator again outperformed the full separator for all instances and notably closed up to 100% of the gap for some instance variations.

4.2.3 Instance family `neos5`

Finally, Figures 5 and 6 show our results for instance family `neos5`. In both experiments the reduced separator consistently outperforms the full separator, and closes up to 100% of the gap for some of the variations in the second experiment.

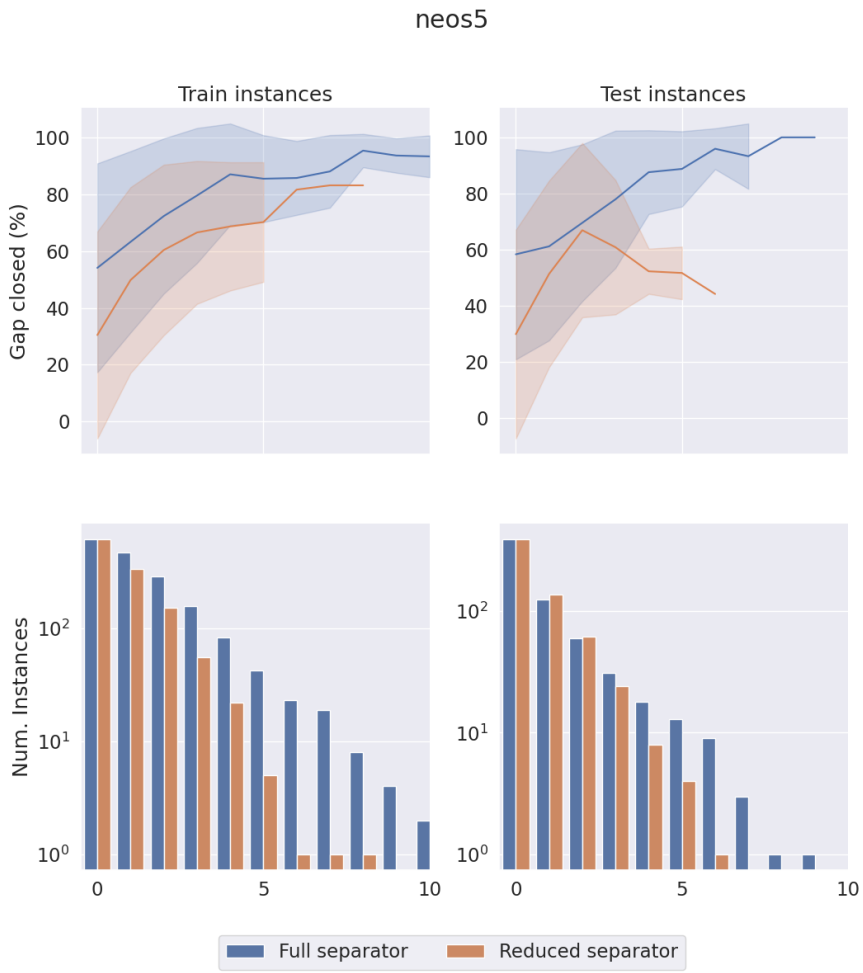

Fig. 5 Behavior on base instance `neos5`. Only variations where the full separator produces more than 5% gap closed were considered. The horizontal axis represents the round of separation. On the top plots the vertical axis represents gap closed, with the lines showing the mean gap closed, and the error bars showing its standard deviation. On the bottom plots the vertical axis represents the number of variations that made it to each round.

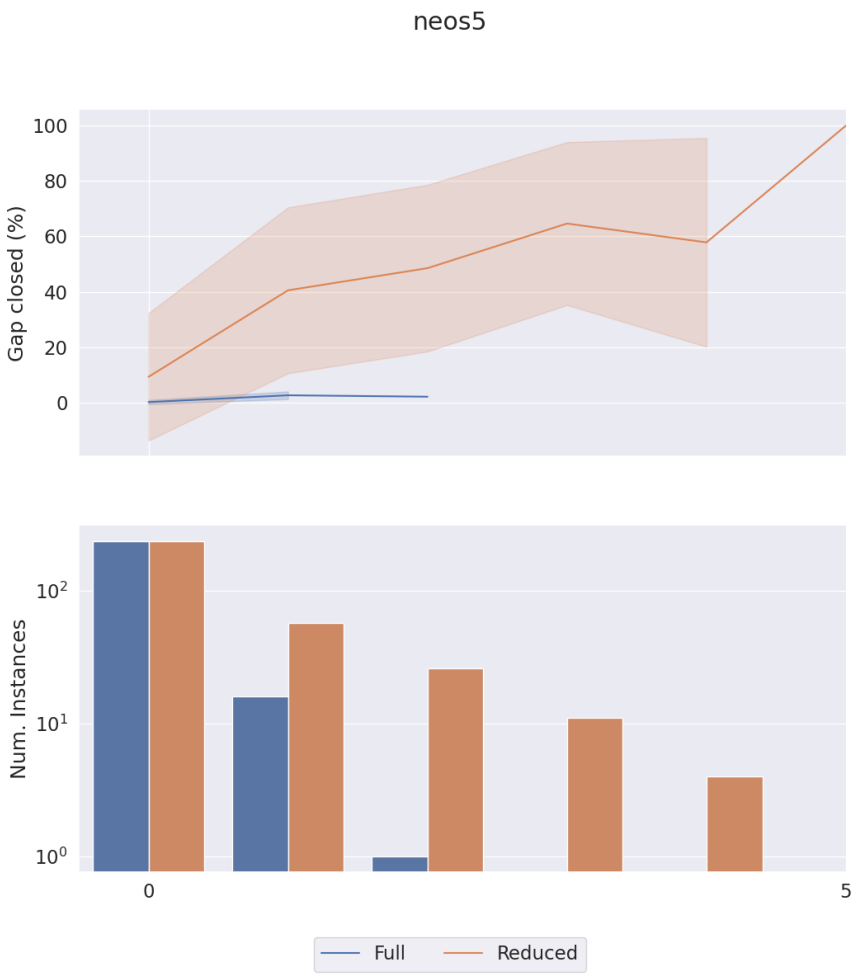

Fig. 6 Behavior on base instance `neos5`. Only variations where the full separator produces less than 5% gap closed were considered. The horizontal axis represents the round of separation. On the top plot the vertical axis represents gap closed, with the lines showing the mean gap closed, and the error bars showing its standard deviation. On the bottom plot the vertical axis represents the number of variations that made it to each round.

4.2.4 Takeaways

Takeaway 1: Figures 1, 3 and 5 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 at best imitates the cuts produced by the full separator on the training instances, it is not surprising that the performance of the reduced separator is upper bounded by that of the full separator.

Takeaway 2: These results 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 is able to correct for that and generate cuts that close much of the gap, the true goal in cut generation.

5 Conclusion

We have demonstrated that a rather simple binary classification formulation for the problem of reducing the set of constraints that are considered by an optimization-based cut separator is promising. 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 MIPLIB2017-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 sets both from MIPLIB2017 and other more structured problem classes is of immediate interest.

References

[1] Fischetti, M., Lodi, A.: Optimizing over the first Chvátal closure. Math. Program. **110**(1), 3–20 (2007)

[2] Bonami, P., Cornuéjols, G., Dash, S., Fischetti, M., Lodi, A.: Projected Chvátal–Gomory cuts for mixed integer linear programs. Mathematical Programming, Series A **113**(2), 241–257 (2008)

[3] Balas, E., Saxena, A.: Optimizing over the split closure. Mathematical Programming **113** (2008) https://doi.org/10.1007/s10107-006-0049-5

[4] Dash, S., Günlük, O., Lodi, A.: MIR closures of polyhedral sets. Math. Program. **121**(1), 33–60 (2010)

[5] Nemhauser, G.L., Wolsey, L.A.: Integer and Combinatorial Optimization vol. 55. John Wiley & Sons, New York (1999)

[6] Marchand, H., Wolsey, L.A.: Aggregation and mixed integer rounding to solve MIPs. Operations Research **49**(3), 363–371 (2001)

[7] Achterberg, T.: Constraint integer programming. Doctoral thesis, Technische Universität Berlin, Fakultät II - Mathematik und Naturwissenschaften, Berlin (2007). https://doi.org/10.14279/depositonce-1634 . http://dx.doi.org/10.14279/depositonce-1634

[8] Lodi, A.: The Heuristic (Dark) Side of MIP Solvers, pp. 273–284. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-30671-6_10 . https://doi.org/10.1007/978-3-642-30671-6_10

[9] Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., Dilkina, B.: Learning to branch in mixed integer programming. Proceedings of the AAAI Conference on Artificial Intelligence **30**(1) (2016) https://doi.org/10.1609/aaai.v30i1.10080

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

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

[12] Tang, Y., Agrawal, S., Faenza, Y.: Reinforcement learning for integer programming: Learning to cut. In: International Conference on Machine Learning, pp. 9367–9376 (2020). PMLR

[13] Turner, M., Koch, T., Serrano, F., Winkler, M.: Adaptive Cut Selection in Mixed-Integer Linear Programming. Open Journal of Mathematical Optimization **4**, 1–28 (2023) https://doi.org/10.5802/ojmo.25

[14] Balcan, M.-F.F., Prasad, S., Sandholm, T., Vitercik, E.: Sample complexity of tree search configuration: Cutting planes and beyond. Advances in Neural Information Processing Systems **34**, 4015–4027 (2021)

[15] Dragotto, G., Clarke, S., Fisac, J.F., Stellato, B.: Differentiable Cutting-plane Layers for Mixed-integer Linear Optimization (2023). https://arxiv.org/abs/2311.03350

[16] Deza, A., Khalil, E.B.: Machine learning for cutting planes in integer programming: a survey. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, pp. 6592–6600 (2023)

[17] Larsen, E., Lachapelle, S., Bengio, Y., Frejinger, E., Lacoste-Julien, S., Lodi, A.: Predicting tactical solutions to operational planning problems under imperfect information. INFORMS Journal on Computing **34**(1), 227–242 (2022)

[18] Xavier, A.S., Qiu, F., Ahmed, S.: Learning to solve large-scale security-constrained unit commitment problems. INFORMS Journal on Computing **33**(2), 739–756 (2021) https://doi.org/10.1287/ijoc.2020.0976

[19] Wesselmann, F., Suhl, U.H.: Implementing cutting plane management and selection techniques. Technical report, University of Paderborn (2012)

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

[21] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)