Influence branching for learning to solve mixed integer programs online

Report for the MIP Workshop 2023 Computational Competition

Paul STRANG[1,2], Zacharie ALES[2,4], Côme BISSUEL[3], Olivier JUAN[3], Safia KEDAD-SIDHOUM[4], Emmanuel RACHELSON[1]

[1] ISAE-SUPAERO, Toulouse, France
[2] ENSTA Paris, Institut Polytechnique de Paris, France
[3] EDF R&D, France
[4] CNAM Paris, CEDRIC, France

Abstract. On the occasion of the 20th Mixed Integer Program Workshop's computational competition, this work introduces a new approach for learning to solve MIPs online. Influence branching, a new graph-oriented variable selection strategy, is applied throughout the first iterations of the Branch & Bound algorithm. This branching heuristic is optimized online with Thompson sampling, which ranks the best graph representations of MIP's structure according to computational speed up over SCIP. We achieve results comparable to state of the art online learning methods. Moreover, our results indicate that our method generalizes well to more general online frameworks, where variations in constraint matrix, constraint vector and objective coefficients can all occur and where more samples are available.

Keywords: Mixed integer programming · Online learning · Branch & Bound · Influence branching · Multi-armed bandit

1 Introduction

Mixed Integer Programming (MIP) is a subfield of combinatorial optimization, a discipline that aims at finding solutions to optimization problems with large but finite sets of feasible solutions. Research in the field has been in particular motivated by the countless industrial applications that can be derived in decision making and operations research. Mixed integer program solvers developed over the last decades have relied on the Branch and Bound (B&B) algorithm [Wol20] to efficiently explore the space of solutions while guaranteeing the optimality of the returned solution. Despite great discoveries in variable selection [AKM05], node selection and cuts selection strategies, along with progress in the conception of primal heuristics and presolve methods [Bes+21], MIPs remain NP-hard problems for which computational load becomes intractable as the number of integer variables increases. Besides, the most efficient selection strategies are based

on complex heuristics fine-tuned by experts on large MIP datasets to obtain the best average performance. In the context of real-world applications, in which similar instances with slightly varying inputs are solved on a regular basis, there is a huge incentive to reduce the solving time by learning efficient tailor-made heuristics.

The MIP Workshop 2023 Computational Competition challenges participants to solve as fast as possible series of 50 MIP instances with slightly varying input, in an online fashion. In this work, we adopt machine learning's statistical point of view and think of each series $s \in \mathcal{S}$ as a set of 50 instances sampled from an unknown probability distribution $\mathcal{Q}_s$. Building on prior works [Lib+16; Eth21; Chm+22], we reformulate MIPcc23's reoptimization challenge as a multi-armed bandits problem, where the reward score $f_{s,i}(a)$ associated to instance $i \in \mathcal{I}_s$, is assumed to follow an unknown distribution $\mathcal{P}_{a,\,s}$ depending on $a \in \mathcal{A}$, the algorithm chosen to solve i. Then, $\sum_{i=1}^{50} f_{s,i}(a_i)$ is the sum of the rewards to minimize for each $s \in \mathcal{S}$, as to our designated action space, $\mathcal{A}$, it consists in the set of hyperparameters couples used to parameterize our new variable selection strategy. SCIP with default parameters [Bes+21] is used as bedrock for our solution, while the implementation of our branching heuristic follows the `pyscipopt.Branchrule` API.

This report is divided into 6 parts. In Section 2, we motivate our approach by a short review of both the literature and the competition's guidelines. Section 3 introduces influence branching, a new branching heuristic that leverages a graph representation of the current instance to select the most influential variable to branch on. Section 3 also highlights influence branching's speed up potential on the competition's instances, before underlining its sensitivity to hyperparameters tuning. This sensitivity justifies the need to learn online the best parameters for each series of instances. Section 4 briefly introduces the two online bandits algorithms considered to learn hyperparameters, and compare their average performances in terms of convergence. Section 5 provides an extensive presentation of computational results obtained on the competition's public instance series. Finally, Section 6 concludes on the relevance of our approach for learning to solve mixed integer programs online.

2 Background

Over the last decade, the field of machine learning has brought many contributions to the MIP literature [Hua+21; BLP21]. Learning to cut [Hua+22; TAF20], learning to branch [ALW14; Kha+16; Eth+20], learning to select heuristics [Lib+16; Chm+22] or learning to estimate solutions efficiently [RAD10] are as many challenging tasks that the machine learning community has tried to address. However, the profile of the instances proposed by MIPcc23 along with specific requirements imposed by the competition's guidelines allow us to narrow the pool of promising approaches. The most binding requirement of the competition is undoubtedly its search for solutions that "work in practice". In fact, most contributions from the literature do not achieve better computational

performances than state of the art solvers, and compare their solution to solvers with for example disabled presolve and cut generation plugins, in order to evaluate the speed up obtained by their sole input. In these conditions, works such as [Gas+19], who achieved true state of the art performances on large MIP benchmarks by learning to imitate the strong branching strategy [App+95], appear promising. However, this achievement was made possible by the training of a large graph convolutionnal neural network on 100,000 samples drawn from 10,000 MIP instances. Such heavy computational work is not suited for our online setting, where every computation is the result of a trade off between learning a model and solving an instance.
Turning to the competition's public series, apart from the *rhs 2* series, instances are rather hard to solve, leading to B&B trees with at least 10,000 nodes and often much more. Thus, machine learning and in particular reinforcement learning approaches do not seem fitted for the challenge as they tend to scale poorly with dimension [Eth+20; Sca+22]. Finally, given the high rate at which SCIP computes LP iterations, a lot of time would probably be lost in communication if we were to implement a python callback for every branching decision, i.e. at every node.
For all these reasons, the MIPcc23 competition constitutes a challenging setup to apply branching strategy learning methods. In order to give the machine learning approach a chance to compete, the learning objective must be rationalized. Therefore, we propose to focus on learning graph representations leading to better branching decision near the root node, throughout the first iterations of the branch-and-bound algorithm. These graph representations, or influence models, and the maximal depth to apply our graph-based heuristic, are the two hyperparameters that will be optimized online with bandits algorithms.

3 Influence branching

We introduce influence branching, a graph branching heuristic encoding part of the mixed integer program structure into an influence graph which is then used for variable selection. Original works from [Eth21] evidenced influence branching's great potential for tree size reduction, especially on hard instances. We provide an adapted version of this heuristic, which achieves better performances on the competition's instances.

3.1 Principle

Influence branching was first inspired by orbital branching [Ost+11], a branching strategy that computes symmetry equivalent groups of variables to partition the search space into orbits. Similarly, influence branching as described by [Eth21], performs a clustering on an influence graph, a graph representation of the MIP instance, and splits variables into influence clusters. These clusters are then used by the brand-and-bound algorithm for variable selection, as it successively

branches on each cluster throughout first iterations. In the following, we consider a mixed integer linear program defined such as:

$$P : \begin{cases} \min c^T x \\ b^- \leq Ax \leq b^+ \ ; \ x \in \mathbb{N}^{|\mathcal{I}|} \times \mathbb{R}^{n-|\mathcal{I}|} \end{cases}$$

with $A \in \mathbb{R}^{m \times n}$, b^-, $b^+ \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, n the total number of variables, m the number of linear constraints and $\mathcal{I}$ the indexes of integer variables.

Definition 1. *(Local influence) We define the local influence w_{ij}^l exerted by variable i on variable j through constraint l. w_{ij}^l can be any function of A, b, c, in particular, we say that i has a non-zero influence on j through l if $\mathbb{1}_{A_{li} \neq 0} \mathbb{1}_{A_{lj} \neq 0} \neq 0$.*

Definition 2. *(Direct influence) We define the direct influence w_{ij} exerted by variable i on variable j over P as :*

$$w_{ij} = \mathbb{1}_{i \neq j} \sum_{l=1}^{m} w_{ij}^l$$

We can then derive a definition for influence graphs.

Definition 3. *(Influence graph) We call influence graph the directed graph $G = (V, E, W)$ where $V = \{1, ..., n\}$, $E = V \times V$ and where $W \in \mathbb{R}^{n \times n}$ the w_{ij} matrix satisfies the definition of direct influence.*

Near the root node, taking the best branching decision does not necessarily mean branching on variables that will lead to the best immediate dual gap reduction, but rather means branching on variables that will have the most impact over the other variables in terms of integrity constraints. Therefore, influential variables are understood as variables which, when branched on, drive other variables to take the value of one of their bounds. Several factors making a variable influential are identified: being involved in a large number of constraints, being associated in average to larger constraint matrix coefficients, being involved in tight constraints or having large associated objective function coefficient.
Influence graphs are designed to capture maximum information from MIP instances' structure. We propose to evaluate the performance of several models of influence on the competition's public series, using A, b, c as well as information extracted from the current LP iteration to define local influence. Influence models description can be found in Table 1. Although vectors b and c do not appear in the definition of local influence, they are used for the normalization of matrix A which will be detailed in the next section.

3.2 Adaptation to MIPcc23

Contrary to [Eth21], we do not perform a spectral clustering on the influence graph. Our implementation of the influence branching heuristic returns the variable within the graph with the maximal total influence :

$$w_i = \sqrt{1 + c_i} \sum_{j \neq i} w_{ij}(g) \tag{1}$$

Count	$w_{ij}^l = \mathbb{1}_{A_{li}} \mathbb{1}_{A_{lj}}$	**Countdual**	$w_{ij}^l = \mathbb{1}_{A_{li}} \mathbb{1}_{A_{lj}} \mathbb{1}_{(y_l{}^* \neq 0)}$
Binary	$w_{ij}^l = \frac{\mathbb{1}_{A_{li}} \mathbb{1}_{A_{lj}}}{\sum_{k=1}^m \mathbb{1}_{A_{ki}} \mathbb{1}_{A_{kj}}}$	**Auxiliary**	$w_{ij}^l = \mathbb{1}_{A_{li}} \mathbb{1}_{A_{lj}} s_i \lvert A_{li} y_l \rvert$
Dual	$w_{ij}^l = \mathbb{1}_{A_{li}} \mathbb{1}_{A_{lj}} \lvert y_l^* \rvert$	**Adversarial**	$w_{ij}^l = \mathbb{1}_{A_{li}} \mathbb{1}_{A_{lj}} s_i \lvert \frac{A_{li}}{A_{lj}} \rvert \mathbb{1}_{(y_l{}^* \neq 0)}$

Table 1. Proposed influence models, with y^* the solution of the dual problem at the current node and s_i the minimal distance to a bound for variable i in the primal solution. $\mathbb{1}_{A_{li} \neq 0}$ is noted $\mathbb{1}_{A_{li}}$ to ease the notations.

as long as the depth of the current node d is inferior or equal to k, the maximum depth. Moreover, variables' total influence are weighted according to their associated objective function coefficient c_i. For nodes of depth $d > k$, influence branching is disabled and the resolution of the MIP instance is handed over to SCIP set with default parameters. Before building the influence graph, a normalization of vectors A, b and c is carried out in order to make the matrix W invariant to problem rescaling.

$$\begin{cases} c \leftarrow c/\sigma(c) \; if \; \sigma(c) \neq 0 \\ A_k \leftarrow A_k/b_k \; if \; b_k \neq 0 \\ A_k \leftarrow A_k/\sigma(A_k) \; if \; b_k = 0 \end{cases} \tag{2}$$

with vector $b \in \mathbb{R}^m$ defined as $b_k = \mathbb{1}_{|b_k^+| < \infty} b_k^+ - \mathbb{1}_{|b_k^-| < \infty} b_k^-$ for $k \in [1, m]$.

On hard instances, influence branching can achieve impressive speed up of solving time compared to state of the art solvers. Table 2 provides the influence model g and the maximal depth k obtaining the best performance for each instance of *obj series 2*. As Table 2 highlights, the speed up potential of influence branching is significant, with an average speed up of -0.38 on *obj series 2* instances compared with default SCIP performances. However, Table 2 also exposes influence branching's extreme sensitivity to hyperparameters setting, as no pair (g, k) appears to perform consistently better on every instance of the series.

Learning which pair (g, k) performs best for any instance of any series would require to shift to a reinforcement learning framework, which would be both computationally unaffordable in the context of the competition, and theoretically challenging as it would require to find an efficient representation of MIP instances. Therefore, we adopt an online bandits framework, as we try to learn which pair (g, k) obtains the best performance in average on a whole series of instances. The sorted average performance obtained by each pair (g, k) on *obj series 2* is provided in Table 3. The average speed up obtained by the best pairs are promising enough to envisage online learning. Equivalent tables for the other series can be found in Appendix A.

Instance	Influence model	Max depth	Performance	SCIP default	Speed up
1	binary	5	0.64	0.70	-0.06
2	adversarial	5	0.53	1.01	-0.48
3	countdual	5	0.49	0.60	-0.11
4	count	4	0.48	0.76	-0.28
5	countdual	2	0.70	1.03	-0.33
6	count	4	0.26	0.44	-0.18
7	auxiliary	3	0.42	0.53	-0.11
8	countdual	2	0.71	0.98	-0.37
9	countdual	4	0.57	1.39	-0.82
10	auxiliary	1	0.35	1.00	-0.65
11	binary	3	0.67	1.53	-0.86
12	dual	1	0.84	1.19	-0.35
13	count	3	0.24	0.45	-0.21
14	binary	1	0.40	0.64	-0.24
15	countdual	3	0.80	1.21	-0.41
16	auxiliary	1	0.74	1.51	-0.77
17	binary	1	0.87	1.21	-0.34
18	countdual	3	0.68	0.76	0.08
19	binary	1	0.28	0.45	-0.17
...	...	...	...	...	...
50	binary	5	0.29	0.66	-0.34
Avg			**0.56**	**0.94**	**-0.38**

Table 2. Speed up potential of Influence branching on *obj series 2*. The performance column corresponds to $f_{s,i} = reltime + gap\,at\,time\,limit + nofeas$, while speed up indicates the performance gain obtained by influence branching compared to SCIP set with default parameters. To illustrate, for instance 2 the pair leading to the best performance of influence branching is ($g = adversarial$, $k = 5$).

Influence model	Max depth	Performance	Speed up	Rank
count	5	0.857	**-0.0862**	**1**
base	6	0.865	-0.0783	2
countdual	2	0.874	-0.0691	3
base	5	0.877	-0.0657	4
count	4	0.882	-0.0606	5
...	...	...	...	...
adversarial	3	0.953	0.0520	34
adversarial	2	0.973	0.0721	35
auxiliary	5	1.05	0.148	36

Table 3. Sorted average performance of influence branching on *obj series 2* for each pair (g, k). The performance column corresponds to the mean of $f_{s,i} = reltime + gap\,at\,time\,limit + nofeas$ over $\mathcal{I}_s$, while speed up indicates the average performance gain obtained by influence branching compared to SCIP set with default parameters.

4 Online bandits

As outlined in Sections 1 and 3, instances from series $s \in \mathcal{S}$ are assumed to be sampled from an abstract distribution $\mathcal{Q}_s$ on the space of MIP instances, and scores $\{f_{s,i}(a)\}_{i \in \mathcal{I}_s}$ with $a \in \mathcal{A} = \{(g,k) : g \in \mathcal{G},\, k \in [1,6]\}$ are assumed to follow an unknown probability distribution $\mathcal{P}_{a,s}$. The optimization task can be reformulated as a multi-armed bandits problem on action space $\mathcal{A}$ where

$$\min_{a_i \in \mathcal{A}} \sum_{i=1}^{50} (1 + 0.1i)\, f_{s,i}(a_i) \tag{3}$$

is the sum of reward to minimize. We note a_s^* the optimal action for series s, and a_0 the action corresponding to SCIP with default parameter.

4.1 Action space

For each series, only 50 samples are available in total. In order to minimize (3), the means of $(\mathcal{P}_{a,s})_{a \in \mathcal{A}}$, noted $(\mu_{a,s})_{a \in \mathcal{A}}$, need to be estimated (or at least ranked) as efficiently as possible for the heuristic to select the action leading to the expected minimum reward. The more actions in the action space, the more samples are needed to guarantee the convergence of the bandits algorithm towards optimal actions. Moreover, Table 3 showed that the spreads between $(\mu_{a,s})_{a \in \mathcal{A}}$ are rather small, comprised between 0.01 and 0.2, in front of standard deviations of $(\mathcal{P}_{a,s})_{a \in \mathcal{A}}$, noted $(\sigma_{a,s})_{a \in \mathcal{A}}$, that were measured around $0.1 - 0.3$ across public series.
In order to mitigate identification issues, five actions among the best performing pairs $(g,\, k)$ across the competition's public series are selected to build action set $\mathcal{A}$:

$$\mathcal{A} = \{(count,\, 1),\, (count,\, 5),\, (countdual,\, 2),\, (binary,\, 3),\, (dual,\, 3)\} \tag{4}$$

4.2 Algorithms

To find the best exploration-exploitation tradeoff, two bandits algorithms, Thompson sampling [Rus+20] and UCB2 [ACF02] are evaluated on public instances series. For Thompson sampling, we make the simplifying assumption that $(\mathcal{P}_{a,s})_{a \in \mathcal{A}}$ are normal distributions with unknown means $(\mu_{a,s})_{a \in \mathcal{A}}$ and fixed standard deviation $\sigma_{a,s} = \sigma = 0.2$, the approximate value measured across public series. Thompson sampling algorithm draws samples from prior distributions $\mathcal{N}(\hat{\mu_a},\, \hat{\sigma_a})$ corresponding to each action and selects the action associated with the minimum sampled value. Then, it collects a reward r, which corresponds to the performance obtained by the chosen algorithm on the instance, and performs a bayesian update of $\hat{\mu_a}$ and $\hat{\sigma_a}$. UCB2 on the other hand, does not require to make any additional assumption on $\mathcal{P}_{a,s}$. It selects the action minimizing the sum $\bar{x}_a + e_{a,r_a}$, with $\bar{x}_a$ the empirical mean of rewards associated with action a and e_{a,r_a} a measure of the exploration rate of action a. UCB2 has the advantage to be deterministic, its performance are reproducible while performances obtained by Thompson sampling must be averaged over a large number of runs.

4.3 Convergence

Convergence tests of online bandits algorithms are performed on *obj series* 2 instances. In order to assess the robustness of the convergence, results are averaged over 10,000 runs. Instance series are shuffled before every run, so that instances are never solved in the same order. Also, it should be specified that computational results from Section 3 are reused in this section and the next. In fact rewards are generated from the performance scores computed during the evaluation of influence branching: there is no further branch-and-bound tree being built at this point.

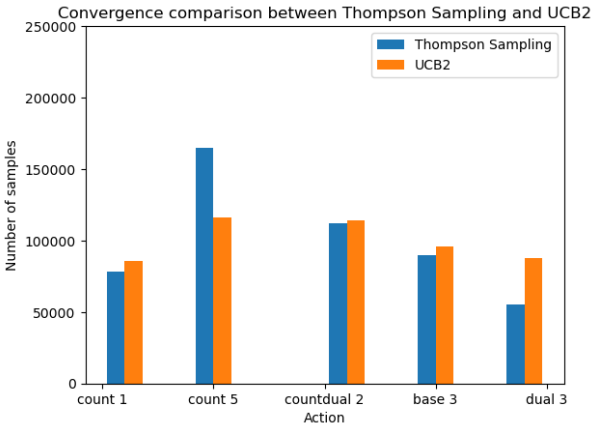

Fig. 1. Convergence comparison between Thompson sampling and UCB2. Results of 10,000 on shuffled *obj series* 2 are aggregated.

UCB2 and Thompson sampling achieve very similar performance in terms of solving time across public series. However, as shown in Fig. 1, Thompson sampling achieves better convergence towards optimal action *count* 5 than UCB2 over 50 iterations on *obj series* 2. This behaviour can be observed on every series. The evaluation metric described in (3) rewards submissions improving solving performance continuously over submission simply minimizing average performance, Thompson sampling is thus prefered over UCB2 for the implementation of our final solution.
Table 4 highlights the convergence performances of Thompson sampling across all public series averaged on 1000 runs. Once again, instance series are shuffled before every run. Convergence score is defined by:

$$CS = \frac{\sum_{i=1}^{50} \mu_{i,s}(a_i) - \mu_{i,s}(a_0)}{\sum_{i=1}^{50} \mu_{i,s}(a_s^*) - \mu_{i,s}(a_0)} \tag{5}$$

with $\mu_{i,s}(a_i) - \mu_{i,s}(a_0)$ the expected speed up of action a_i compared to SCIP and $\mu_{i,s}(a_s^*) - \mu_{i,s}(a_0)$ the speed up obtained by a theoretical oracle. For every

series, a convergence score of at least 60 % is reached. Consequently, on public series the average speed up achieved by our bandits algorithm is superior to half of the theoretical speed up obtained by an oracle performing the series' associated optimal action at every step.

Series	**Convergence score**
bnd series 1	72%
bnd series 2	65%
obj series 1	75%
obj series 2	66%
rhs series 1	64%
rhs series 2	72%
rhs obj series 1	74%

Table 4. Convergence score of Thompson sampling on MIPcc23 public instances series.

5 MIPcc23 computational results

Table 5 gathers measures of $f_{s,\,i}$ computed across every public series, including series with varying constraint matrix coefficients. Results are averaged over 2,000 runs with varying seed. The performance breakdown in terms of *reltime*, *dual gap*, $nofeas$ and *tree size* can be found in Appendix B. Despite the fact that Thompson sampling converges towards optimal action, we don't observe better average performance on instances from the fifth batch. However, given the limited number of samples available, it is not sufficient to conclude that our solution is underfitting. This could result from the instance distribution of public series, for example from a concentration of harder instances in the last batches. Further discussion is provided in Section 6.

Table 6 highlights the speed up obtained by our solution over SCIP across public series. The "easy" instance series *rhs* 2 *series* is the only series where no speed up is achieved. This does not come as a surprise, as [Eth21] showed that influence branching reduces the tree size of instances with large associated B&B trees, while the size from *rhs* 2 *series* instances' trees never exceeds a few dozen nodes. Our solution obtains good speed up performance on *bnd series* 1, *bnd series* 2, *obj series* 1, *obj series* 2, *rhs series* 1 and *mat rhs bnd obj rhs series* 1, with an average score reduction located between -0.02 and -0.06. By comparison, [Chm+22] achieves a 4% speed up over SCIP only on instances taking more than 1000 seconds to solve, while training over 175 instances. This highlights the capacity of influence branching to leverage part of the MIP instance structure to perform efficient variable selection near the root node. Turning to *rhs obj series* 1 and *mat series* 1, the speed up obtained may turn out to be more significant than it appears, for the average score improvement corresponds to a reduction of the

Average $f_{s,i}$ scores	1-50	1-10	11-20	21-30	31-40	41-50
bnd series 1	0.992 ± 0.009	1.036	1.033	0.947	1.016	**0.930**
bnd series 2	0.881 ± 0.020	0.928	**0.850**	0.853	0.858	0.917
obj series 1	0.895 ± 0.006	**0.679**	0.809	0.984	1.000	1.002
obj series 2	0.891 ± 0.022	0.847	1.018	**0.825**	0.859	0.910
rhs series 1	0.875 ± 0.027	**0.810**	0.865	0.839	0.906	0.954
rhs series 2	1.004 ± 0.0001	1.004	1.004	1.004	1.004	1.003
rhs obj series 1	1.015 ± 0.006	1.031	1.033	1.005	**1.002**	1.003
mat series 1	1.050 ± 0.013	**1.004**	1.044	1.074	1.068	1.087
mat rhs bnd obj series 1	0.677 ± 0.021	0.768	0.707	0.660	**0.538**	0.714

Table 5. Average $f_{s,i}$ results. Instances are solved in the order of the competition dataset. Results are averaged over 2,000 runs, with varying seed. As a reminder, $f_{s,i} = reltime + gap + nofeas$.

average dual gap at termination, and not to a reduction of solving time. Comparison with other candidates scores is necessary to assess our performance on these series.

Series	Average $f_{s,i}$	Speed up compared to SCIP
bnd series 1	0.992 ± 0.009	$\mathbf{-0.031 \pm 0.009}$
bnd series 2	0.881 ± 0.020	$\mathbf{-0.037 \pm 0.020}$
obj series 1	0.895 ± 0.006	$\mathbf{-0.022 \pm 0.006}$
obj series 2	0.891 ± 0.022	$\mathbf{-0.052 \pm 0.022}$
rhs series 1	0.875 ± 0.027	$\mathbf{-0.048 \pm 0.027}$
rhs series 2	1.004 ± 0.0001	0.001 ± 0.0001
rhs obj series 1	1.015 ± 0.006	-0.005 ± 0.006
mat series 1	1.050 ± 0.013	-0.009 ± 0.013
mat rhs bnd obj series 1	0.677 ± 0.021	$\mathbf{-0.061 \pm 0.021}$

Table 6. Averaged speed up obtained across public series. Results are averaged over 2,000 runs, with varying seed.

6 Perspectives

Learning to solve MIP instances online is a challenging task. Adopting a bandits framework, this work proposes to learn online the optimal setting of influence branching among five preselected pairs (g, k) of parameters. Since these pairs were selected according to their performance on public series, $s \in \mathcal{S}$, it is legitimate to wonder whether performances on hidden series, $s' \in \mathcal{S}'$, will be comparable. Two arguments can be made in favour of our approach.

First, as highlighted in Appendix A, suboptimal actions also lead to significant average speed ups. Consequently, it is likely that for a hidden series sampled from an unknown distribution $\mathcal{Q}_{s'}$, one or several actions from $\mathcal{A}$ will lead to an average performance speed up. Second, even in the case when none of the 5 actions of our action set leads to an average performance speed up, this does not disqualify our approach for learning to solve MIPs online. In fact, owing to the limited number of samples available for each series of the competition, we were constrained to reduce the size of our action set to guarantee the convergence of Thompson sampling. However, in a more general framework where possibly several hundreds of instances sampled from the same distribution are solved online [Chm+22], larger action sets could be built while preserving Thompson sampling convergence properties. This would result in a better adaptability of our solution to instance series sampled from unknown distributions $(\mathcal{Q}_{s'})_{s' \in \mathcal{S}'}$ while preserving performances obtained on instance series sampled from $(\mathcal{Q}_s)_{s \in \mathcal{S}}$.

References

[App+95] David Applegate et al. *Finding cuts in the TSP (A preliminary report)*. Tech. rep. Citeseer, 1995.

[ACF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning* 47.2 (May 1, 2002), pp. 235–256. ISSN: 1573-0565. DOI: 10.1023/A:1013689704352. URL: https://doi.org/10.1023/A:1013689704352 (visited on 02/14/2023).

[AKM05] Tobias Achterberg, Thorsten Koch, and Alexander Martin. "Branching rules revisited". In: *Operations Research Letters* 33.1 (Jan. 1, 2005), pp. 42–54. ISSN: 0167-6377. DOI: 10.1016/j.orl.2004.04.002. URL: https://www.sciencedirect.com/science/article/pii/S0167637704000501 (visited on 12/13/2022).

[RAD10] Emmanuel Rachelson, Ala Ben Abbes, and Sebastien Diemer. "Combining mixed integer programming and supervised learning for fast re-planning". In: *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*. Vol. 2. IEEE. 2010, pp. 63–70.

[Ost+11] James Ostrowski et al. "Orbital branching". In: *Mathematical Programming* 126 (2011), pp. 147–178.

[ALW14] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. "A supervised machine learning approach to variable branching in branch-and-bound". In: *In ecml*. Citeseer, 2014, p. 11.

[Kha+16] Elias Khalil et al. "Learning to Branch in Mixed Integer Programming". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1 (Feb. 21, 2016). Number: 1. ISSN: 2374-3468. DOI: 10.1609/aaai.v30i1.10080. URL: https://ojs.aaai.org/index.php/AAAI/article/view/10080 (visited on 12/14/2022).

[Lib+16] Giovanni Di Liberto et al. "DASH: Dynamic Approach for Switching Heuristics". In: *European Journal of Operational Research* 248.3 (Feb. 1, 2016), pp. 943–953. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2015.08.018. URL: https://www.sciencedirect.com/science/article/pii/S0377221715007559 (visited on 12/14/2022).

[Gas+19] Maxime Gasse et al. "Exact Combinatorial Optimization with Graph Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/d14c2267d848abeb81fd590f371d39bd-Abstract.html (visited on 12/14/2022).

[Eth+20] Marc Etheve et al. "Reinforcement Learning for Variable Selection in a Branch and Bound Algorithm". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Ed. by Emmanuel Hebrard and Nysret Musliu. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 176–185. ISBN: 978-3-030-58942-4. DOI: 10.1007/978-3-030-58942-4_12.

[Rus+20] Daniel Russo et al. *A Tutorial on Thompson Sampling*. July 14, 2020. arXiv: 1707.02038[cs]. URL: http://arxiv.org/abs/1707.02038 (visited on 12/14/2022).

[TAF20] 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*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, Nov. 21, 2020, pp. 9367–9376. URL: https://proceedings.mlr.press/v119/tang20a.html (visited on 12/14/2022).

[Wol20] Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

[BLP21] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: A methodological tour d'horizon". In: *European Journal of Operational Research* 290.2 (Apr. 16, 2021), pp. 405–421. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2020.07.063. URL: https://www.sciencedirect.com/science/article/pii/S0377221720306895 (visited on 12/14/2022).

[Bes+21] Ksenia Bestuzheva et al. *The SCIP Optimization Suite 8.0*. Technical Report. Optimization Online, Dec. 2021. URL: http://www.optimization-online.org/DB_HTML/2021/12/8728.html.

[Eth21] Marc Etheve. "Solving repeated optimization problems by Machine Learning". PhD thesis. HESAM Université, Dec. 3, 2021. URL: https://theses.hal.science/tel-03675471 (visited on 12/08/2022).

[Hua+21] Lingying Huang et al. "Branch and Bound in Mixed Integer Linear Programming Problems: A Survey of Techniques and Trends". In: (Nov. 5, 2021). DOI: 10.48550/arXiv.2111.06257. URL: https://arxiv.org/abs/2111.06257v1 (visited on 12/13/2022).

[Chm+22] Antonia Chmiela et al. "Online Learning for Scheduling MIP Heuristics". In: (2022). URL: https://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/8914 (visited on 12/21/2022).

[Hua+22] Zeren Huang et al. "Learning to select cuts for efficient mixed-integer programming". In: *Pattern Recognition* 123 (Mar. 1, 2022), p. 108353. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2021.108353. URL: https://www.sciencedirect.com/science/article/pii/S0031320321005331 (visited on 12/14/2022).

[Sca+22] Lara Scavuzzo et al. *Learning to branch with Tree MDPs*. Oct. 13, 2022. DOI: 10.48550/arXiv.2205.11107. arXiv: 2205.11107[cs, math]. URL: http://arxiv.org/abs/2205.11107 (visited on 12/21/2022).

A Influence branching average speed up potential on public series

Influence model	Max depth	Performance	Speed up
count	1	0.980	**-0.0424**
binary	1	0.981	-0.0418
binary	3	0.983	-0.0392
count	3	0.990	-0.0324
dual	1	0.991	-0.0312
...	...	...	...
dual	6	1.031	+0.0084
countdual	4	1.032	+0.0090
countdual	6	1.041	+0.0190

Table 7. Sorted average performance of influence branching on ***bnd series 1*** for each pair (g, k).

Influence model	Max depth	Performance	Speed up
countdual	2	0.845	**-0.0733**
countdual	3	0.846	-0.0721
countdual	1	0.865	-0.0527
dual	1	0.869	-0.0492
count	1	0.870	-0.0482
...	...	...	...
dual	5	0.983	+0.0654
dual	6	0.996	+0.0774
dual	4	0.996	+0.0782

Table 8. Sorted average performance of influence branching on ***bnd series 2*** for each pair (g, k).

Influence model	**Max depth**	**Performance**	**Speed up**
binary	3	0.884	**-0.0330**
countdual	5	0.885	-0.0322
dual	5	0.886	-0.0311
dual	4	0.886	-0.0308
count	3	0.888	-0.0290
...	...	...	...
countdual	1	0.904	-0.0098
count	1	0.906	-0.0081
binary	1	0.907	-0.0070

Table 9. Sorted average performance of influence branching on ***obj series 1*** for each pair (g, k).

Influence model	**Max depth**	**Performance**	**Speed up**
count	1	0.835	**-0.0872**
binary	1	0.848	-0.0746
countdual	6	0.855	-0.0676
dual	5	0.858	-0.0643
countdual	3	0.863	-0.0598
...	...	...	...
binary	5	0.914	-0.0087
count	3	0.920	-0.00321
binary	5	0.926	+0.0031

Table 10. Sorted average performance of influence branching on ***rhs series 1*** for each pair (g, k).

Influence model	**Max depth**	**Performance**	**Speed up**
count	1	1.003	+0.0008
countdual	1	1.003	+0.0009
binary	1	1.003	+0.0010
countdual	2	1.004	+0.0010
dual	1	1.004	+0.0011
...	...	...	...
binary	6	1.004	+0.0016
count	6	1.004	+0.0016
binary	5	1.004	+0.0016

Table 11. Sorted average performance of influence branching on ***rhs series 2*** for each pair (g, k).

Influence model	Max depth	Performance	Speed up
dual	3	1.012	**-0.0084**
count	8	1.013	-0.0078
binary	8	1.013	-0.078
binary	2	1.014	-0.0066
count	2	1.014	-0.0066
...	...	...	...
binary	4	1.027	+0.0065
countdual	2	1.032	+0.0011
dual	1	1.032	+0.0012

Table 12. Sorted average performance of influence branching on ***rhs obj series 1*** for each pair (g, k).

Influence model	Max depth	Performance	Speed up
count	1	1.020	**-0.0364**
countdual	1	1.030	-0.0270
binary	1	1.034	-0.0237
count	2	1.040	-0.0170
binary	2	1.041	-0.0162
...	...	...	...
dual	5	1.110	+0.0540
dual	6	1.226	+0.0656
countdual	6	1.140	+0.0831

Table 13. Sorted average performance of influence branching on ***mat series 1*** for each pair (g, k).

Influence model	Max depth	Performance	Speed up
dual	3	0.643	**-0.0957**
dual	2	0.653	-0.0852
count	1	0.659	-0.0796
count	2	0.664	-0.0745
countdual	3	0.670	-0.0679
...	...	...	...
count	6	0.726	-0.0129
countdual	6	0.730	-0.0083
count	5	0.745	+0.0063

Table 14. Sorted average performance of influence branching on ***mat rhs bnd obj series 1*** for each pair (g, k).

B Performance breakdown on public series

Average *tree size*	**1-50**	**1-10**	**11-20**	**21-30**	**31-40**	**41-50**
bnd series 1	7449 ± 216	7222	6832	9550	7277	**6364**
bnd series 2	10346 ± 369	10355	10138	10208	10586	**10442**
obj series 1	245255 ± 2825	**181684**	228534	290934	275192	249931
obj series 2	87956 ± 2077	99270	85366	88066	**81301**	85774
rhs series 1	22143 ± 999	**17996**	21930	25537	20679	24571
rhs series 2	37 ± 1	43	42	14	38	48
rhs obj series 1	587 ± 40	498	225	569	899	741
mat series 1	9084 ± 295	8902	9294	9384	8850	8986
mat rhs bnd obj series 1	3332 ± 129	3711	3632	3198	**2053**	4066

Table 15. *Tree size* results. Instances are solved in the order of the competition dataset. Results are averaged over 2,000 runs, with varying seed.

Average *reltime*	**1-50**	**1-10**	**11-20**	**21-30**	**31-40**	**41-50**
bnd series 1	0.949 ± 0.009	0.980	0.992	0.914	0.965	**0.895**
bnd series 2	0.842 ± 0.019	0.884	0.817	**0.811**	0.824	0.879
obj series 1	0.895 ± 0.006	**0.676**	0.815	0.984	1.000	1.000
obj series 2	0.802 ± 0.014	0.822	0.848	0.777	**0.741**	0.823
rhs series 1	0.874 ± 0.029	**0.805**	0.862	0.841	0.913	0.949
rhs series 2	1.000 ± 0.000	1.000	1.000	1.000	1.000	1.000
rhs obj series 1	0.998 ± 0.001	**0.988**	1.000	1.000	1.000	1.000
mat series 1	0.971 ± 0.008	**0.947**	0.971	0.968	0.977	0.996
mat rhs bnd obj series 1	0.677 ± 0.021	0.768	0.707	0.660	**0.538**	0.714

Table 16. *reltime* results. Instances are solved in the order of the competition dataset. Results are averaged over 2,000 runs, with varying seed.

Average *dual gap*	**1-50**	**1-10**	**11-20**	**21-30**	**31-40**	**41-50**
bnd series 1	0.045 ± 0.002	0.057	0.042	0.039	0.050	**0.038**
bnd series 2	0.036 ± 0.004	0.048	0.034	0.036	**0.021**	0.039
obj series 1	0.001 ± 0.000	**0.000**	0.000	0.001	0.001	0.002
obj series 2	0.095 ± 0.013	0.029	0.175	**0.046**	0.129	0.097
rhs series 1	0.001 ± 0.000	**0.000**	0.000	0.000	0.001	0.001
rhs series 2	0.003 ± 0.0001	0.003	0.003	0.003	0.003	0.003
rhs obj series 1	0.017 ± 0.006	0.039	0.038	0.005	**0.002**	0.003
mat series 1	0.075 ± 0.007	**0.053**	0.071	0.073	0.084	0.094
mat rhs bnd obj series 1	0.006 ± 0.003	0.020	0.001	**0.001**	0.002	0.004

Table 17. *dual gap* results. Instances are solved in the order of the competition dataset. Results are averaged over 2,000 runs, with varying seed.

Average $nofeas$	**1-50**	**1-10**	**11-20**	**21-30**	**31-40**	**41-50**
bnd series 1	0.000 ± 0.000	0.000	0.000	0.000	0.000	0.000
bnd series 2	0.000 ± 0.000	0.000	0.000	0.000	0.000	0.000
obj series 1	0.000 ± 0.000	0.000	0.000	0.000	0.000	1.002
obj series 2	0.000 ± 0.000	0.000	0.000	0.000	0.000	0.000
rhs series 1	0.000 ± 0.000	0.000	0.000	0.000	0.000	0.000
rhs series 2	0.000 ± 0.000	0.000	0.000	0.000	0.000	0.000
rhs obj series 1	0.000 ± 0.000	0.000	0.000	0.000	0.000	0.000
mat series 1	0.000 ± 0.000	0.000	0.000	0.000	0.000	0.000
mat rhs bnd obj series 1	0.000 ± 0.000	0.000	0.000	0.000	0.000	0.000

Table 18. $nofeas$ results. Instances are solved in the order of the competition dataset. Results are averaged over 2,000 runs, with varying seed.