One-shot Learning for MIPs with SOS1 Constraints

Charly Robinson La Rocca[1*], Jean-François Cordeau[2] and Emma Frejinger[1]

[1*]Department of Computer Science and Operations Research, Université de Montréal, Montreal, Canada.
[2]Department of Logistics and Operations Management, HEC Montréal, Montreal, Canada.

*Corresponding author(s). E-mail(s): charly.robinson.la.rocca@umontreal.ca;
Contributing authors: jean-francois.cordeau@hec.ca; emma.frejinger@umontreal.ca;

Abstract

Efficient algorithms and solvers are required to provide optimal or near-optimal solutions quickly, enabling organizations to react promptly to dynamic situations such as supply chain disruptions or changing customer demands. State-of-the-art mixed-integer programming (MIP) solvers are crafted to tackle a wide variety of problems, yet many real-world situations present problem instances that originate from a narrow distribution. This has inspired the creation of tailored approaches that exploit historical data to inform heuristic design. Deep learning (DL) methods are typically used in this context to extract patterns from data, but they require large datasets and comprehensive hyperparameter tuning for strong performance. This article describes a one-shot learning heuristic that leverages solutions discovered within the branch-and-bound tree to construct a model with minimal overhead. We evaluate our method on the locomotive assignment problem (LAP), a recurring real-world problem that is challenging to solve at scale. Experimental results reveal a tenfold acceleration compared to a general-purpose solver (CPLEX) with a relative gap of less than 2%. We also demonstrate that the method is competitive with CPLEX on the majority of selected MIPLIB instances with SOS1 constraints.

Keywords: Machine Learning, Mixed-integer programming, Learning Heuristics, Fleet Management Problem, MIPLIB.

1 Introduction

The intersection of mixed-integer programming (MIP) and machine learning (ML) presents a unique opportunity to advance the frontiers of optimization and tackle complex problems in areas such as transportation, energy, and finance. This trend has emerged in recent years with the maturation of specialized hardware [43] and software [34] stacks. Combined with the availability of large datasets, ML can exploit patterns from data to design tailored algorithms. Today, there are many strategies that take advantage of learning methods to improve upon standard solvers. Bengio et al. [4] provide a tour d'horizon on ML for combinatorial optimization (CO) and describe the different ways they can be integrated. The first idea uses ML to directly predict the solution of the problem instance from the input features. One example of this is the application of the pointer network [40] to learn to predict the optimal permutation for the travelling salesman problem (TSP). The second approach is to learn to optimize the hyperparameters of a CO algorithm [46]. Third, ML can also be used alongside an optimization algorithm. In practice, this is often done via the integration of learning in the branch-and-bound (B&B) framework [31]. Our approach belongs to the first class of hybrid methods.

This paper proposes a primal heuristic for MIPs with special ordered sets of type 1 (SOS1 or S1) [3] that we refer to as Probe and Freeze[1] (PNF). We use the shorter notation S1 to reference special ordered sets of type 1 in the remaining of this paper. The idea of PNF is to use the probing data to train a classifier to predict the optimal assignment in each S1 constraint of the MIP. Training is done in a one-shot fashion, analogous to the one-shot learning (OSL) paradigm, which is the main difference compared to other works in the literature. The method uses entropy, a key concept in information theory, to quantify uncertainty. The predictions are used to freeze a subset of the variables in the S1 constraints to produce a reduced problem that can be significantly easier to solve.

Motivations for S1. The motivation for this approach comes from the hypothesis that S1 constraints have a structuring effect on the final solution. These constraints are useful when a choice involves multiple options or resources, and only one can be selected. For example, in the facility location problem (FLP), the objective is to determine the optimal locations for a set of facilities to serve a given set of customers. The S1 constraints can either be used to model which type of facility to build or how to assign customers to facilities. In the locomotive assignment problem (LAP), which is a problem studied in this paper, S1 constraints model the assignment of consists (i.e., groups of locomotives) to trains. The flow variables in the LAP are often trivial to optimize for a given assignment of consists to trains. In other words, the S1 constraints are the main source of difficulty in the problem.

Motivations for OSL. The main motivation behind OSL is the cost associated with the data generation process. In the context of MIPs, the training dataset typically comes from solving a large number of instances. This is a time-consuming process that requires a significant amount of computational resources. Another reason is the fact that most of the rich data structure generated by the B&B is usually overlooked

[1] Source code available at https://github.com/laroccacharly/MLSOS

during the search. This work explores how to make use of this readily available data using statistical tools to better inform a heuristic. As a side effect of using OSL, we can produce a method that is easy to understand and reproduce. It is also worth noting that the OSL paradigm is not a hard requirement for our approach. In fact, we can use any ML classifier to predict the optimal class for each S1 constraint. The OSL model acts as a strong baseline which can be combined with other modern ML models to improve the accuracy of predictions.

This document first presents some related work on the integration of ML in MIP solvers. This is followed by an overview of our methodology which is composed of three routines: probe, select and freeze. Finally, we include results for the speed up and the gap on a set of LAP and MIPLIB instances. The LAP is a problem that is not only challenging to solve but also recurring i.e., similar problem instances are solved repeatedly over time. Hence, there is a substantial economic incentive to find a heuristic that can solve the LAP quickly. We use the Canadian National Railway Company (CN) as the subject of our case study for the LAP.

2 Related Works

Commercial solvers such as CPLEX [23] and Gurobi [17] are widely considered the state-of-the-art (SOA) for solving MIPs. The underlying algorithm of these solvers is the "divide and conquer" framework called branch-and-bound [29]. Alone, the classical B&B implementation will often fail to close the optimality gap in an acceptable amount of time. Extensive research has been done to improve the performance of B&B. In the following, we first present some of the most relevant works in the operational research (OR) literature. We then discuss the recent advances in ML for MIPs.

Operational research approaches. Heuristics for MIPs can be classified in two general classes: constructive and improvement heuristics [19]. An example of a constructive heuristic is the Feasiblity Pump (FP) [1, 5, 13] where the main goal is to quickly produce feasible solutions. Typically, solutions generated by this type of heuristic do not come with guarantees. Improvement heuristics iteratively apply updating steps to an initial solution (feasible or not) to refine the quality of solutions. In that category, there are Large Neighborhood Search (LNS) heuristics such as Relaxation Induced Neighborhood Search (RINS) [10] and, more recently, Adaptive Large Neighborhood Search (ALNS) [20]. LNS heuristics define a neightborhood of interest and explore that space to improve the current incumbent solution.

Constructive heuristics, also called Start Heuristics (SH) [6], are most similar to the methodology proposed in this paper. Diving [30] and Rounding [2, 32] heuristics are two other examples that fit in that category. Diving heuristics employ a depth-first-search (DFS) strategy which focuses on decreasing the number of fractional variables in the LP. At every iteration, the algorithm selects a fractional variable to bound. Fractional, coefficient and pseudocost diving all share this idea but they use a different strategy to select which variable to bound [6]. Rounding heuristics use the notion of up and down locks to perform rounding that guarantees that all linear constraints are satisfied [2]. The Zero Integer (ZI) round [42] heuristic calculates the bounds

for fractional variables such that the LP stays feasible. Variables are shifted to the corresponding bound that minimizes fractionality.

We found two relevant references [15, 24] in the OR literature that exploit information theory principles to aid decision-making during branching. In that context, the LP value of a variable is interpreted as a probability. Similar to our approach, the first paper [15] quantifies the amount of uncertainty of a binary variable using the notion of entropy [35]. They describe Entropic Branching (EB), a look-ahead strategy similar to strong branching, where variable selection is done in a way that minimizes uncertainty in child nodes. The second paper [24] uses a restart strategy to collect partial assignments from fathomed nodes called clauses. They experiment with both branching alternatives and clause inequalities that use this information to guide the search. This approach is quite similar to the Rapid Learning heuristic [7, 8] (related to no-good or conflict learning) which applies a depth-first search to collect valid conflict constraints. From this probing step, it identifies explanations for the infeasibility and exploits that information in the remainder of the search to prune parts of the tree.

ML for MIPs. In the previous paragraphs, we introduced heuristics that are readily available within SOA solvers. The solvers typically tune the schedules of these heuristics using the empirical performance on a broad set of test instances. The works of [9] use a data-driven approach to obtain a problem-specific schedule for different heuristics. By accomplishing this, they are able to improve the primal integral by 49% on two classes of instances. The scheduling of heuristics is just one of many possible ways to improve upon the default behaviour of SOA solvers. Other papers use ML to improve the node and variable selection strategies [14, 26]. Learning to cut is another promising approach for strengthening the linear relaxation [38].

Graph neural networks (GNN) [18] are commonly used to model the graph representation of a MIP. This architecture is popular because a MIP can be represented as a bipartite graph on variables and constraints. They can be trained via supervised learning to predict the stability [11], the bias [27] or the value [22] of binary variables. The predictions are used to guide the search in the B&B tree. These approaches keep the exactness of the algorithm because all possible decisions made by the ML model are valid by design. The optimality certificate is often obtained by iteratively improving both the upper and lower bounds until they converge to the same solution. However, for many practical applications, it is preferred to have a fast heuristic that can quickly produce a good solution without the need to prove optimality. For example, the Deep Learning Tree Search (DLTS) [21] algorithm uses neural network predictions for both branching and bounding. Given that it relies on an estimated bound instead of a proven one, this approach is not guaranteed to converge to the optimal solution. Stochastic policies trained to generate solutions to CO problems have also been proposed [39]. This type of approach iteratively generates solutions by sampling from a learned distribution over the variables, hence it does not produce any optimality certificate. Nonetheless, the authors show that the solutions generated by their approach are of high quality.

Our work differs from previous hybrid approaches in two key aspects. First, we fix a subset of the variables in the MIP formulation (before calling the solver) instead of directly guiding the search. In that sense, our approach is most similar to the Relax

and Fix [45] primal heuristic where variables are fixed based on the fractional solution of the LP. Second, we do not rely on a large dataset to train the model. Instead, analogous to one-shot learning, training is done on the probing data which contains few samples.

One-shot learning. One-shot learning has emerged as a significant challenge in the ML literature, aiming to mimic human-like learning from few examples. Two notable approaches in this domain are Matching Networks [41] and Prototypical Networks [36]. Matching Networks employ an attention mechanism over a learned embedding of the training set of examples to predict classes for unlabeled points, functioning as a weighted nearest-neighbour classifier. Conversely, Prototypical Networks pose that an embedding exists around a single prototype representation for each class. It produces a distribution over classes for a query point based on a softmax of the distances to the prototypes. Both approaches have been successfully applied to vision and language tasks.

In summary, we noticed that a GNN trained on a large dataset is the dominant tool used to learn to improve SOA MIP solvers. However, they require extensive training and tuning to yield acceptable performance. In contrast, our method aims to be as efficient as possible by exploiting readily available data from the B&B tree. As a consequence, we can take advantage of simpler models that are less prone to overfitting. This design choice creates the opportunity for an accurate model with low opportunity costs.

3 Methodology

A combinatorial optimization problem $\mathcal{P}$ can be formulated as a mixed-integer linear programming (MIP) model using the notation

$$\mathcal{P} := \arg\min_{\mathbf{x}} \{c^T \mathbf{x} \,|\, A\mathbf{x} \leq b,\, \mathbf{x} \in \mathcal{B} \cup \mathcal{Q} \cup \mathcal{W}\}, \tag{1}$$

where $\mathbf{x}$ is the vector of decision variables that is partitioned into $\mathcal{B}$, $\mathcal{Q}$ and $\mathcal{W}$, the index sets of binary, integer and continuous variables, respectively. Our approach makes the assumption that some subset of the constraints are S1 [3]. Each variable in the set should be binary, and at most one variable in the set can be equal to 1. The motivation for working with S1 constraints comes from the fact that it can be modelled as a classification problem. Hence, we can use ML tools to predict the *class* associated with each binary variable in the set. The solution vector for the binary variables is analogous to the one-hot encoding of the optimal class. The mathematical representation of S1 is given by the equation

$$\sum_{k \in K_v} x_v^k = 1 \qquad \forall v \in \mathcal{V}, \tag{2}$$

where x_v^k is the binary variable that indicates whether the class $k \in K_v$ belongs to the S1 set $v \in \mathcal{V}$. The set $\mathcal{V}$ contains all the S1 constraints in the problem and we assume that each variable belongs to at most one S1 constraint. If that assumption does not

hold, the method might produce an infeasible reduced problem. Each constraint $v \in \mathcal{V}$ has a set of possible outcomes K_v and the goal is to find k^* which is the optimal class in the set. The motivation for working with S1 constraints comes from the effectiveness of machine learning classification tools that can help predict the optimal class k^*. In that context, we define k' as the predicted class and we use it to freeze the corresponding binary variable $x_v^{k'}$. We will describe an S1 constraint v as *frozen* whenever we add the constraint $x_v^{k'} = 1$ in the MIP to create a reduced problem $\mathcal{P}'$. If a classifier can accurately predict the optimal class ($k' = k^*$), then we can expect $\mathcal{P}'$ to be significantly easier to solve than the original problem $\mathcal{P}$. The approach can be decomposed into three steps which are probe, select and freeze. These routines are explained in more detail in the remaining of this section.

3.1 Probe

The probing step is used to collect data about the decision variables x_v^k. We call the solver on $\mathcal{P}$ for a predefined probing time budget T_p and fetch intermediate solutions in the callback. The probing routine does not discriminate any intermediate solution; it stores all integer feasible and fractional solutions. We define x_{vt}^k as the value of the variable x_v^k at iteration t, which corresponds to a node in the B&B tree. At every iteration, we compute the most likely class based on which variable has the highest value $k_{vt} = \arg\max_{k \in K_v}\{x_{vt}^k\}$. We then concatenate the k_{vt} into the vector $\mathbf{k}_v = [k_{v1}, k_{v2}, ..., k_{vn}]$, where n is the probing sample size. The vector $\mathbf{k}_v$ corresponds to the probing data for the S1 constraint v, and it is used as input to the classifier. Finally, we define $\mathcal{K}$ as the set of all probing data vectors $\mathbf{k}_v$.

3.2 Select

Given the heuristic nature of freezing variables, we need to select the S1 constraints whose variables will be the object of a freezing in a way that minimizes potential assignment errors. The selection strategy uses a scoring system to sort the S1 constraints based on the entropy H to infer uncertainty. It is defined as

$$H(\mathbf{k}_v) = -\sum_{k \in \mathbf{k}_v} P(k|\mathbf{k}_v) \log P(k|\mathbf{k}_v), \tag{3}$$

where $P(k|\mathbf{k}_v)$ is the probability associated with the class k given the probing data $\mathbf{k}_v$. The probability is computed using the corresponding frequency:

$$P(k|\mathbf{k}_v) = \frac{|\{z \in \mathbf{k}_v \,|\, z = k\}|}{|\mathbf{k}_v|}. \tag{4}$$

Next, we compute the score which is the negative of the entropy:

$$\text{score}(v) = -H(\mathbf{k}_v). \tag{5}$$

Finally, we select the first $r \cdot |\mathcal{V}|$ constraints from the sorted list to produce the set of selected constraints $\mathcal{V}'$. The ratio of constraints to freeze r is a hyperparameter

that can be tuned to modify the aggressiveness of the algorithm. The idea behind this strategy is reasonably intuitive. On the one hand, when the entropy $H(\mathbf{k}_v)$ is low, it means that the solutions found during probing are similar and the most frequent class in $\mathbf{k}_v$ is likely to be optimal. On the other hand, when the entropy is high, the most frequent class changes often and therefore we are less confident about what the optimal class is.

3.3 Freeze

Once probing is done, we restart the solver and create a reduced problem $\mathcal{P}'$ by freezing the variables x_v^k in the selected constraints $\mathcal{V}'$. The freezing routine builds *freezing cuts* (FC) defined as follows:

$$FC(v, k') := \{x_v^{k'} = 1\}, \tag{6}$$

where the predicted class k' is the most likely class in the probing vector:

$$k' = \arg\max_k (P(k|\mathbf{k}_v)). \tag{7}$$

The underlying classifier uses the histogram method with discrete bins for each class in $\mathbf{k}_v$. The reduced problem $\mathcal{P}'$ is then solved to completion or until it reaches the total time limit. The probe and freeze methodology is summarized in Algorithm 1.

Algorithm 1 Probe and Freeze (PNF) algorithm

procedure PNF($\mathcal{P}, T_p, r$)
  $\mathcal{P}' \leftarrow \mathcal{P}$ ▷ Make a copy of the problem
  $\mathcal{K} \leftarrow \text{solve}(P, T_p)$ ▷ Solve to collect probing data
  $\mathcal{V}' \leftarrow \text{select}(\mathcal{K}, r)$ ▷ Select the constraints to freeze
  for $v \in \mathcal{V}'$ **do**
    $\mathbf{k}_v \leftarrow \mathcal{K}$ ▷ Get probing data for the constraint v
    $k' \leftarrow \arg\max_k (P(k|\mathbf{k}_v))$ ▷ Predict the optimal class
    $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{FC(v, k')\}$ ▷ Freeze the selected constraint
  end for
  $\mathbf{x}' \leftarrow \text{solve}(\mathcal{P}')$ ▷ Solve the reduced problem and produce an approximate solution $\mathbf{x}'$
end procedure

4 Experimental Results

We test our approach on two different sets of instances. The first set includes LAP instances constructed using historical data from CN. The second set contains instances with S1 constraints from the MIPLIB 2017 [16] library. These two sets differ with respect to the underlying distribution. The first set is regular, whereas the second set

is much more diverse. The LAP instances are generated using consecutive weeks of data from CN, which implies that all instances come from the same distribution. In contrast, instances in MIPLIB were built independently and relate to many different problems. For this reason, MIPLIB can be seen as a more challenging test bed given our methodology targets a regular setting. Since each instance in MIPLIB is unique, it is not possible to effectively calibrate the two hyperparameters of PNF (r, T_p). Nevertheless, we believe it is important to test PNF on MIPLIB to assess its limitations and understand where it may not be applicable. The results are presented in Sections 4.1 and 4.2.

Metrics. The main goal of this work is to predict the optimal solution in a MIP with S1 constraints to speed up the solution process. Naturally, the main metric we focus on is the runtime speed up (RS), which is the ratio between the runtime of the baseline CPLEX and the algorithm as defined in the following equation:

$$\text{RS} = \frac{\text{runtime(CPLEX)}}{\text{runtime(PNF)} + T_p}. \tag{8}$$

There are two points that are worth noting about how RS is computed. First, the runtime is not measured when the solver stops, but instead, it is measured relative to the moment when the best feasible solution is found. Second, since PNF has a probing phase, we add the probing time budget T_p to the measured runtime. This is required to make a fair comparison with CPLEX. The second metric we report is the relative gap (RG) in percentage which is defined as follows:

$$\text{RG} = \frac{c^T \mathbf{x}' - c^T \mathbf{x}_{\text{CPLEX}}}{c^T \mathbf{x}_{\text{CPLEX}}} \times 100, \tag{9}$$

where $\mathbf{x}'$ and $\mathbf{x}_{\text{CPLEX}}$ are the best solutions found by the heuristic and CPLEX, respectively. The value of RG can be negative if the heuristic finds a better solution than CPLEX.

Setup. Our software stack is fully written in Julia and we use CPLEX version 12.10. By default, we run each scenario for one hour on a single thread of Intel Gold 6148 Skylake (2.4 GHz) and 35 GiB of memory.

4.1 LAP Instances

The LAP is a scheduling problem that is used to assign locomotives to trains in a railway network. The problem is modelled using a space-time network in which each node represents a time and space location, whereas each arc represents a movement of locomotives. The goal of the LAP is to find the cheapest way to send a certain number of locomotives through a network such that all capacity and demand constraints are satisfied. The main reasons that motivate the use of a learning heuristic for the LAP are the following. First, the problem is NP-hard and it is challenging to find good solutions in a reasonable amount of time. Second, given the scheduling nature of the problem, instances follow daily and weekly cyclical patterns which are useful for

learning. Third, there is a vast amount of historical data available from CN which can be used to generate instances and train/validate the strategy.

MIP model. The S1 constraints in the LAP are used to model the fact that only one set of locomotives (i.e., a consist) can be assigned to each train. The decisions x_v^k represent the configuration of locomotives k assigned to each train arc v in the instance. The MIP of the LAP also contains flow conservation constraints for each node in the network. We will not discuss the details of the model here as it is not the focus of this paper. We refer to [33] for the details on the MIP formulation of the LAP.

Instance generation. The historical data used to build the instances were provided by CN. The dataset contains the origin and destination stations for each train operated during the year. We study the performance on different difficulty levels to access the generalization properties of the heuristic. To adjust the difficulty level, we subsample the data in a way that preserves the weekly and daily patterns. This is done by selecting the set of trains that operate in a given region of the network. We tune the radius of the region to control the size of the instances and its difficulty as shown in Table 1. The difficulty level is labeled as easy (E), medium (M), or hard (H). Instances E and M are generated using the subsample process whereas H instances are full instances. These full instances are generated using the same methodology as in [33] and we run them with a time limit of 6 hours instead of 1 hour. The main difference compared to [33] is that we test on the most difficulty instances (all trains) whereas [33] split the test set into mainline only and all trains.

Results. The quantiles for the speed up and relative gap of the PNF heuristic are presented in Tables 2 and 3, respectively. The suffixes of the scenarios are the fixing ratio r and the probing time budget T_p in seconds. The tables include a variety of scenarios with different hyperparameters to reveal their impact on the performance of the algorithm. The fixing ratio r adjusts the number of variables that are fixed and, therefore, the aggressiveness of the heuristic. When $r = 0.9$, the heuristic can produce on average a speed up of 27.75x for easy instances and 17.1x for hard instances. In both cases, the relative gap is less than 2% on average and the 95th quantile is below 4%. Inversely, when r is small, the speed up is lower but the quality of solutions is better. For r between 0.2 and 0.5, the speed up is at most 2x but the relative gap is usually negligible. In fact, we discover a better solution than CPLEX for most M instances because the relative gap is negative. The hyperparameter T_p has the opposite effect on performance. A large probing time budget leads to a smaller speed up because more time is spent probing. However, the relative gap can improve because the heuristic will collect more data. The first two scenarios (E+PNF_0.2_10, E+PNF_0.2_20) reveal that effect as the speed up goes from 1.84x to 1.32x but the relative gap improves from 0.02% to 0.01% on average.

Full instances. The performance metrics on the full instances (H) have a distinct behaviour compared to the other instance types. In most cases, the speed up is close to 1, which is due to the fact that closing the optimality gap is difficult and, therefore, all scenarios reach the time limit. To better appreciate the acceleration benefit of PNF on full instances we computed the performance over time as shown in Tables 5 and 6. These tables respectively display the gap relative to the best known solution and the number of instances solved at different points in time. On average, with

the best configuration (PNF_0.5_1800), PNF is able to find an upper bound within 1.48% of the best known solution after 1 hour (1800s for probing + 1800s for solving) whereas CPLEX's average gap is above 4%. Furthermore, PNF finds a feasible upper bound for more instances during the first two hours (10/20 for PNF_0.2_1800 and 7/20 for CPLEX). Even in the worst case, the best configuration finds a better solution than CPLEX with a 95th quantile of -0.36% (see Table 3). The worst configuration (PNF_0.2_600) has a 95th quantile of 4.25%, which shows that the heuristic needs a longer probing time budget to perform well on full instances.

In Table 4, we report the correlation between the relative gap and each hyperparameter to understand their impact on the performance of the algorithm. We choose to compute the Spearman [37] and Kandall [25] correlations because they are robust to outliers. As expected, the fixing ratio r has a strong positive correlation with the relative gap (0.56 on average). This is because the fixing ratio controls the number of variables to freeze; the quality of solutions is more likely to decline when r increases. The probing time budget T_p has a negative but weaker correlation with the relative gap (-0.37 on average). This is because T_p controls the amount of data that is collected and, therefore, the accuracy of the heuristic.

Table 1 Average summary metrics for LAP instances using CPLEX

Difficulty	Runtime (s)	Train count	Optimality gap (%)
E	206.42	141.77	0.01
M	2450.07	336.40	0.26
H	21600.20	5673.20	2.94

Table 2 Quantiles for runtime speed up in LAP instances

Scenario	Quantiles					Mean	Sample size
	0.05	0.25	0.5	0.75	0.95		
E+PNF_0.5_8	1.02	1.84	3.21	5.46	9.73	4.07	30.00
E+PNF_0.9_6	2.73	4.46	19.79	40.64	50.40	27.75	30.00
E+PNF_0.2_10	0.52	0.90	1.27	2.23	4.82	1.84	30.00
E+PNF_0.2_20	0.53	0.66	1.01	1.68	2.66	1.32	30.00
E+PNF_0.5_30	0.37	0.65	1.77	2.91	5.29	2.39	30.00
M+PNF_0.2_60	0.45	0.93	1.00	1.08	3.98	1.32	30.00
M+PNF_0.5_60	0.92	1.01	1.16	2.64	9.85	2.63	30.00
M+PNF_0.2_120	0.48	0.92	0.99	1.11	3.51	1.25	30.00
M+PNF_0.5_120	0.96	1.03	1.37	1.82	7.50	2.27	30.00
M+PNF_0.9_120	1.51	4.07	21.74	26.37	28.83	17.10	30.00
H+PNF_0.2_600	0.90	0.99	1.00	1.02	1.19	1.02	20.00
H+PNF_0.2_1800	0.97	0.99	1.00	1.02	1.02	1.00	20.00
H+PNF_0.5_1800	0.92	0.98	0.99	1.00	1.00	0.98	20.00

Table 3 Quantiles for relative gap (%) in LAP instances

Scenario	Quantiles					Mean	Sample size
	0.05	0.25	0.5	0.75	0.95		
E+PNF_0.5_8	-0.00	0.01	0.05	0.20	0.38	0.11	30.00
E+PNF_0.9_6	0.55	0.94	1.50	2.31	3.40	1.73	30.00
E+PNF_0.2_10	-0.00	-0.00	0.00	0.02	0.08	0.02	30.00
E+PNF_0.2_20	-0.00	-0.00	0.00	0.00	0.05	0.01	30.00
E+PNF_0.5_30	-0.00	0.00	0.03	0.13	0.22	0.07	30.00
M+PNF_0.2_60	-0.42	-0.12	-0.00	0.00	0.04	-0.09	30.00
M+PNF_0.5_60	-0.52	-0.11	0.01	0.04	0.12	-0.08	30.00
M+PNF_0.2_120	-0.38	-0.05	-0.00	0.00	0.03	-0.06	30.00
M+PNF_0.5_120	-0.44	-0.03	0.01	0.03	0.11	-0.07	30.00
M+PNF_0.9_120	0.46	0.65	1.02	1.47	2.43	1.15	30.00
H+PNF_0.2_600	-1.58	-1.09	-0.87	-0.58	4.25	-0.10	20.00
H+PNF_0.2_1800	-1.14	-0.99	-0.74	-0.48	0.90	-0.52	20.00
H+PNF_0.5_1800	-1.93	-1.16	-0.98	-0.69	-0.36	-1.03	20.00

Table 4 Correlations with relative gap for LAP instances

Parameter	Spearman	Kendall	Average
Fixing ratio	0.61	0.51	0.56
Probing time budget (s)	-0.42	-0.33	-0.37

Table 5 Gap (%) for full LAP instances

Scenario	Time after probing (s)			
	1800	3600	5400	7200
cplex	nan	4.04	3.94	2.49
PNF_0.2_600	1.72	1.48	1.09	0.88
PNF_0.2_1800	2.01	1.67	1.18	1.17
PNF_0.5_1800	1.48	1.39	0.89	0.69

Table 6 Full LAP instances with feasible solution

Scenario	Time after probing (s)			
	1800	3600	5400	7200
cplex	0	6	7	7
PNF_0.2_600	7	9	9	9
PNF_0.2_1800	7	10	10	10
PNF_0.5_1800	8	9	9	9

4.2 MIPLIB Instances

The MIPLIB [16] dataset is an open source library of MIP instances. It contains a variety of instances from different fields such as transportation, scheduling, and others related to combinatorial optimization. The instances in MIPLIB are generated independently and are not related to each other. This makes it a more challenging test bed for our approach. The main reason for using MIPLIB is to show that our approach can generalize to instances that have S1 constraints as the only common trait.

Identifying S1 constraints. The main assumption of the approach is the presence of S1 constraints in the MIP. Hence, the first step is to identify the subset of MIPLIB instances with that type of constraint. To do so, we need an algorithm to detect S1 constraints in the MIP. A constraint i can be decomposed into the so-called left-hand side (LHS) and right-hand side (RHS) as follows:

$$\text{LHS} := \sum_j a_{ij} x_j, \quad \text{RHS} := b_i, \tag{10}$$

where a_{ij} are the coefficients of the variables x_j and b_i is the right-hand side. For a constraint to be S1 as defined in (2), the a_{ij} must be 0 when x_j is not binary, a_{ij} must be 0 or 1 if x_j is binary, and the b_i must be equal to 1. Also, there must be an equality sign between the LHS and RHS.

Instance selection. The No Free Lunch (NFL) theorem [44] shows that, for optimization problems, the average performance of any pair of algorithms across all possible problems is identical. In that sense, the PNF heuristic is unlikely to perform well on all MIPLIB instances. The task of selecting the appropriate algorithm for a given problem is often called the Algorithm Selection Problem (ASP) [28]. This can be done by building a mapping between the features of the problem and the expected performance of an algorithm. In our case, we are interested in selecting the MIPLIB instances on which the PNF heuristic is likely to perform well. To achieve this, we identify the features related to the implicit assumption of the approach. The main assumption of PNF is related to the stability of the solutions; if the solutions found during probing are stable then they are likely to be optimal. For that reason, the PNF heuristic can perform poorly when the average entropy of solutions is high ($\sum_{v\in\mathcal{V}} H(\mathbf{k}_v)/\left|\mathcal{V}\right| > 1$) and, therefore, we should not apply the method when that is the case. The other indicator of interest is the probing sample size (n) which is related to the probing time budget (T_p). We remove every scenario where $n \leq 5$ because the sample size would be too small to be representative. For reference, the results on the full MIPLIB dataset are available in the Appendix.

Instance types. We split the MIPLIB dataset into three categories based on the runtime required to solve the instance with CPLEX. The distribution of runtimes is presented in Table 7. The easy category contains instances that can be solved in less than 60 seconds, hard instances reach the time limit of 3,600 seconds, and the medium category contains the remaining instances. This classification is required to adjust the probing time budget T_p to be proportional to the instance difficulty. It is worth noting that MIPLIB instances already have a classification based on their difficulty. However, we found that the classification based on the runtime is more relevant to our approach.

Results. The quantiles for the speed up and the relative gap are shown in Tables 9 and 10, respectively. For the MIPLIB dataset, we use a more conservative set of hyperparameters to reduce the likelihood of outliers. In particular, the fixing ratio is set to 0.2 for all scenarios and the probing time budget is around 10% of the CPLEX runtime. The results reveal an average speed up between 1.19x and 2.8x, which is comparable to the LAP results when $r = 0.2$. However, the results for the relative gap differ significantly. We observe extreme outliers in both directions. For example, in scenario H+PNF_0.2_300, the 95th quantile is 23.83% and the 5th quantile is -21.62%. This is due to the fact that the set of instances in MIPLIB is diverse and the hyperparameters are not tuned appropriately for each instance. Interestingly, the average relative gap is negative for both hard scenarios (-1.16% and -4.87%). This shows that the proposed approach has the potential to find better solutions than CPLEX within 1 hour. In Fig. 1, we summarize the results via a histogram which shows the number of improved scenarios based on an indicator that includes both the speed up and the relative gap. A scenario is defined as improved if there is a speed up above 1x and a relative gap of at most 1%. The main takeaway is that the PNF

heuristic is able to outperform CPLEX on the majority of selected instances (58.3%) at the cost of potentially getting a strictly worse outcome (6% of the time).

Limitations. An important limitation of PNF is that it is not guaranteed to produce a feasible reduced problem $\mathcal{P}'$. As reported in Table 8, given the best configuration, the PNF heuristic is unable to find a feasible solution for 11% of selected instances (21/188). In particular, this will happen if the same variable is frozen in two different constraints with different values. The PNF heuristic assumes that S1 constraints in $\mathcal{P}$ are independent from each other, which is not always the case in MIPLIB instances. In general, PNF does not come with any guarantee. We recommend using the PNF heuristic only when the user is aiming for a substantial speed up and is willing to handle possible feasibility issues. The integration of a feasibility recovery strategy is left for future work. For now, we propose a few possible paths to mitigate the risk of infeasibility. Whenever the reduced problem $\mathcal{P}'$ is infeasible, the user could fallback to the original problem $\mathcal{P}$ to solve the instance, reduce the parameter r or increase the probing time budget T_p. It is also possible to repair MIP infeasibility through local branching [12]. Another solution would be to solve a feasibility relaxation of the problem to see which constraint is not satisfied in the reduced problem. From that experience, the user can decide to change the selection routine to avoid fixing variables in S1 constraints that are not independent. The user might also realize that the OSL classifier returns poor predictions for his problem. In the end, the exact reason for infeasibility is problem specific and, therefore, we cannot provide a general solution. It is worth noting that we never encountered infeasibility in the LAP because the S1 constraints are independent by construction.

Table 7 CPLEX runtime on MIPLIB instances with S1 constraints

Scenario	Quantiles					Mean	Sample size
	0.05	0.25	0.5	0.75	0.95		
E+cplex	2.98	4.00	7.05	13.53	39.29	11.84	68.00
M+cplex	76.81	116.72	449.53	1307.78	2281.55	770.16	44.00
H+cplex	3600.00	3600.01	3600.02	3600.06	3600.83	3598.90	76.00

Table 8 Number of solved MIPLIB instances

Scenario	Number of instances	Solved	Unsolved
E+PNF_0.2_1	68	67	1
E+PNF_0.2_2	68	57	11
M+PNF_0.2_60	44	39	5
M+PNF_0.2_120	44	37	7
H+PNF_0.2_300	76	61	15
H+PNF_0.2_600	76	58	18
Total	376	319	57

Table 9 Quantiles for runtime speed up in selected MIPLIB instances

Scenario	Quantiles					Mean	Sample size
	0.05	0.25	0.5	0.75	0.95		
E+PNF_0.2_2	0.83	1.07	1.19	1.24	1.68	1.19	14.00
M+PNF_0.2_60	0.10	0.38	0.61	3.62	10.07	2.80	11.00
M+PNF_0.2_120	0.06	0.30	0.63	1.34	4.78	1.23	13.00
H+PNF_0.2_300	0.01	0.29	0.83	1.01	4.40	1.32	18.00
H+PNF_0.2_600	0.01	0.38	0.91	1.31	4.32	1.22	18.00

Table 10 Quantiles for relative gap (%) in selected MIPLIB instances

Scenario	Quantiles					Mean	Sample size
	0.05	0.25	0.5	0.75	0.95		
E+PNF_0.2_2	-0.00	-0.00	0.00	0.00	0.32	0.06	14.00
M+PNF_0.2_60	-0.00	0.00	0.00	2.32	24.07	4.80	11.00
M+PNF_0.2_120	-0.00	0.00	0.00	0.00	8.94	1.57	13.00
H+PNF_0.2_300	-21.62	-0.06	0.00	0.00	23.83	-1.16	18.00
H+PNF_0.2_600	-31.21	-0.00	0.00	0.06	2.75	-4.87	18.00

5 Perspectives and Future Work

The main purpose of the paper was to explore if a one-shot learning heuristic could improve the performance of a commercial solver on MIPs with S1 constraints. According to our reading of the literature, most other works on this subject use some form of neural networks combined with supervised or reinforcement learning. The overhead of DL is problematic in practice because of the specialized hardware requirements and the challenge of generating representative training datasets. Our method bypasses this issue by using a classical ML model that can be trained using readily available data. This means that it can be integrated natively within a commercial solver without requiring any additional resources. Despite the small capacity of our model, we were able to find good quality solutions for the LAP faster than CPLEX for both small and full instances. Furthermore, we discovered a clear correlation between the quality of solutions and the hyperparameters of the heuristic (r, T_p). Therefore, they can be tuned by the end user to achieve the desired trade-off between quality and speed. On the MIPLIB dataset, the performance of PNF was less reliable, which reveals the core limitation of the method. This was expected because there are some problems with a strong dependence between S1 constraints, which PNF does not take into account. Nevertheless, we remain optimistic about the potential of PNF since it can find better solutions than CPLEX on many hard instances of MIPLIB. This study has demonstrated that the initial solutions found during the first few iterations of the B&B search can serve as reliable proxies for the optimal solution. A significant contribution of this research lies in the use of entropy as an effective measure of solution uncertainty. By identifying and avoiding freezing the constraints with high entropy, we minimize the risk of compromising the quality of the final solution. Future research might attempt to augment the heuristic by adding more features or more data. However, we found that this did not improve the performance enough to justify the additional cost. In fact, we

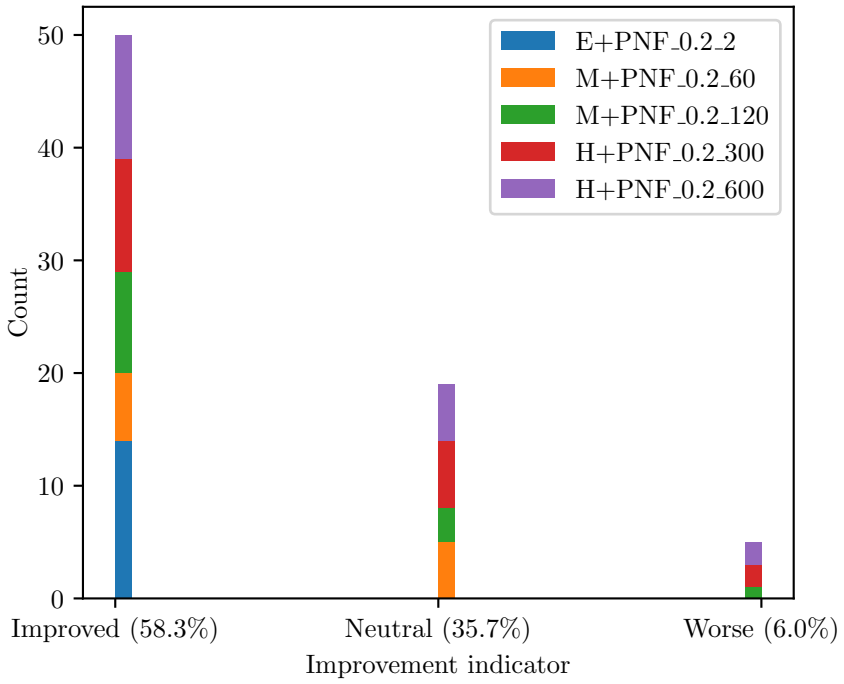

Fig. 1 Histogram for the improvement indicator in MIPLIB instances (improved if speed up > 1 and relative gap less than 1%, worse if speed up below 1 and relative gap above 1%, neutral otherwise)

realized that the simpler heuristic was more robust and performed better in practice. This is consistent with Occam's razor principle which states that the simplest explanation is usually the best. For that reason, we believe PNF should be seen as a strong baseline for future works on learned heuristics for MIPs because it is easy to understand and reproduce. An interesting next step would be to add a worst-case analysis and a feasibility recovery strategy on top of the heuristic to increase its robustness and reliability.

Statements and Declarations

Acknowledgments

We acknowledge the support of the Institute for Data Valorization (IVADO), the Canada First Research Excellence Fund (Apogée/CFREF), and the Canadian National Railway Company (through the CN Chain in Optimization of Railway Operations)for their financial backing and provision of valuable data which was instrumental in advancing this research.

Ethical approval

No ethical approval was required for this study.

Competing interests

We, the authors of this study, declare that we have no competing interests. This encompasses both financial and non-financial interests that could undermine the integrity of our research.

Authors' contributions

Charly Robinson La Rocca was involved in all aspects of the research, conceptualization, methodology, programming, preparing experimental results, and writing the manuscript. Jean-François Cordeau and Emma Frejinger contributed to the conceptualization, methodology, and editing the manuscript. All authors reviewed the manuscript.

Funding

Charly Robinson La Rocca received funding from Institute for Data Valorization (IVADO). Emma Frejinger received funding from Canada Research Chairs.

Availability of data and materials

The data that support the findings of this study come from the Canadian National Railway Company. The data for the LAP is subject to confidentiality restrictions. However, the code and the related data for MIPLIB instances are open source, accessible at https://github.com/laroccacharly/MLSOS.

Appendix A Extra MIPLIB Results

This appendix contains the results for the full MIPLIB dataset. We report the speed up and relative gap in Tables A1 and A2, respectively.

Table A1 Quantiles for runtime speed up in all MIPLIB instances

Scenario	Quantiles					Mean	Sample size
	0.05	0.25	0.5	0.75	0.95		
E+PNF_0.2_1	0.99	1.18	1.59	2.23	4.23	2.28	67.00
E+PNF_0.2_2	0.85	1.07	1.19	1.52	5.69	1.88	57.00
M+PNF_0.2_60	0.14	0.53	1.21	3.20	11.35	3.03	39.00
M+PNF_0.2_120	0.07	0.36	0.73	1.71	5.74	1.74	37.00
H+PNF_0.2_300	0.01	0.09	0.94	1.80	11.27	2.14	61.00
H+PNF_0.2_600	0.00	0.04	0.63	1.39	5.62	1.25	58.00

Table A2 Quantiles for relative gap (%) in all MIPLIB instances

Scenario	Quantiles					Mean	Sample size
	0.05	0.25	0.5	0.75	0.95		
E+PNF_0.2_1	0.00	0.00	0.00	4.17	338.54	194.80	67.00
E+PNF_0.2_2	-0.00	0.00	0.00	0.34	104.49	23.63	57.00
M+PNF_0.2_60	-1.86	0.00	0.00	4.71	67.83	72.57	39.00
M+PNF_0.2_120	-0.82	0.00	0.00	2.25	75.14	75.09	37.00
H+PNF_0.2_300	-58.90	-0.97	0.00	1.18	47.42	9.54	61.00
H+PNF_0.2_600	-60.36	-0.19	0.00	1.00	20.62	3.23	58.00

References

[1] Achterberg T, Berthold T (2007) Improving the feasibility pump. Discrete Optimization 4(1):77–86

[2] Achterberg T, Berthold T, Hendel G (2012) Rounding and propagation heuristics for mixed integer programming. In: Operations Research Proceedings 2011: Selected Papers of the International Conference on Operations Research (OR 2011), August 30-September 2, 2011, Zurich, Switzerland, Springer, pp 71–76

[3] Beale E, Tomlin J (1970) Special facilities in a general mathematical programming system for non–convex problems using ordered sets of variables. In: Lawrence J (ed) Proceedings of the Fifth International Conference on Operational Research. Tavistock Publications, London, pp 447–454

[4] Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research 290(2):405–421. https://doi.org/10.1016/j.ejor.2020.07.063

[5] Bertacco L, Fischetti M, Lodi A (2007) A feasibility pump heuristic for general mixed-integer problems. Discrete Optimization 4(1):63–76

[6] Berthold T (2006) Primal heuristics for mixed integer programs. PhD thesis, Zuse Institute Berlin (ZIB)

[7] Berthold T, Feydy T, Stuckey PJ (2010) Rapid Learning for Binary Programs. In: Lodi A, Milano M, Toth P (eds) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, pp 51–55, https://doi.org/10.1007/978-3-642-13520-0_8

[8] Berthold T, Stuckey PJ, Witzig J (2019) Local Rapid Learning for Integer Programs. In: Rousseau LM, Stergiou K (eds) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. Springer International Publishing, Cham, Lecture Notes in Computer Science, pp 67–83, https://doi.org/10.1007/978-3-030-19212-9_5

[9] Chmiela A, Khalil E, Gleixner A, et al (2021) Learning to schedule heuristics in branch and bound. In: Advances in Neural Information Processing Systems, vol 34. Curran Associates, Inc., pp 24235–24246

[10] Danna E, Rothberg E, Pape CL (2005) Exploring relaxation induced neighborhoods to improve MIP solutions. Mathematical Programming 102:71–90

[11] Ding JY, Zhang C, Shen L, et al (2020) Accelerating primal solution findings for mixed integer programs based on solution prediction. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp 1452–1459

[12] Fischetti M, Lodi A (2008) Repairing MIP infeasibility through local branching. Computers & Operations Research 35(5):1436–1445

[13] Fischetti M, Glover F, Lodi A (2005) The feasibility pump. Mathematical Programming 104:91–104

[14] Gasse M, Chételat D, Ferroni N, et al (2019) Exact combinatorial optimization with graph convolutional neural networks. In: Advances in Neural Information Processing Systems, pp 15580–15592

[15] Gilpin A, Sandholm T (2011) Information-theoretic approaches to branching in search. Discrete Optimization 8(2):147–159. https://doi.org/10.1016/j.disopt.2010.07.001

[16] Gleixner A, Hendel G, Gamrath G, et al (2021) MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. Mathematical Programming Computation

[17] Gurobi Optimization, LLC (2023) Gurobi Optimizer Reference Manual. URL https://www.gurobi.com

[18] Hamilton WL (2020) Graph representation learning. Synthesis Lectures on Artifical Intelligence and Machine Learning 14(3):1–159

[19] Hanafi S, Todosijević R (2017) Mathematical programming based heuristics for the 0–1 MIP: a survey. Journal of Heuristics 23(4):165–206

[20] Hendel G (2022) Adaptive large neighborhood search for mixed integer programming. Mathematical Programming Computation 14(2):185–221. https://doi.org/10.1007/s12532-021-00209-7

[21] Hottung A, Tanaka S, Tierney K (2020) Deep learning assisted heuristic tree search for the container pre-marshalling problem. Computers & Operations Research 113:104781

[22] Huang L, Chen X, Huo W, et al (2022) Improving primal heuristics for mixed integer programming problems based on problem reduction: A learning-based approach. In: 2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV), IEEE, pp 181–186

[23] IBM Corporation (2021) IBM ILOG CPLEX Optimization Studio (Version 12.10)

[24] Karzan FK, Nemhauser GL, Savelsbergh MW (2009) Information-based branching schemes for binary linear mixed integer problems. Mathematical Programming Computation 1(4):249–293

[25] Kendall MG (1938) A new measure of rank correlation. Biometrika 30(1/2):81–93

[26] Khalil E, Bodic PL, Song L, et al (2016) Learning to Branch in Mixed Integer Programming. Proceedings of the AAAI Conference on Artificial Intelligence 30(1). https://doi.org/10.1609/aaai.v30i1.10080

[27] Khalil EB, Morris C, Lodi A (2022) MIP-GNN: A data-driven framework for guiding combinatorial solvers. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp 10219–10227

[28] Kotthoff L (2012) Algorithm selection for combinatorial search problems: A survey. AI Mag 35:48–60

[29] Land AH, Doig AG (1960) An automatic method of solving discrete programming problems. Econometrica 28(3):pp. 497–520

[30] Lazić J (2016) Variable and single neighbourhood diving for MIP feasibility. Yugoslav Journal of Operations Research 26(2)

[31] Lodi A, Zarpellon G (2017) On learning and branching: A survey. Top 25(2):207–236

[32] Neumann C, Stein O, Sudermann-Merx N (2019) A feasible rounding approach for mixed-integer optimization problems. Computational Optimization and Applications 72:309–337

[33] Ortiz-Astorquiza C, Cordeau JF, Frejinger E (2021) The Locomotive Assignment Problem with Distributed Power at the Canadian National Railway Company. Transportation Science 55(2):510–531

[34] Paszke A, Gross S, Massa F, et al (2019) Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems 32:8026–8037

[35] Shannon CE (1948) A mathematical theory of communication. The Bell system technical journal 27(3):379–423

[36] Snell J, Swersky K, Zemel R (2017) Prototypical Networks for Few-shot Learning. In: Advances in Neural Information Processing Systems, vol 30. Curran Associates, Inc., pp 4080–4090

[37] Spearman C (1904) The proof and measurement of association between two things. The American Journal of Psychology 15(1):72–101

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

[39] Toenshoff J, Ritzert M, Wolf H, et al (2021) Graph neural networks for maximum constraint satisfaction. Frontiers in Artificial Intelligence 3:580607

[40] Vinyals O, Fortunato M, Jaitly N (2015) Pointer Networks. In: Advances in Neural Information Processing Systems, pp 2692–2700

[41] Vinyals O, Blundell C, Lillicrap T, et al (2016) Matching Networks for One Shot Learning. Advances in Neural Information Processing Systems 29:3630–3638

[42] Wallace C (2010) Zi round, a MIP rounding heuristic. Journal of Heuristics 16:715–722

[43] Wang YE, Wei GY, Brooks D (2019) Benchmarking TPU, GPU, and CPU Platforms for Deep Learning. arXiv preprint arXiv:1907.10701 [cs.LG]

[44] Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1):67–82

[45] Wolsey LA (2020) Integer programming. John Wiley & Sons

[46] Xu L, Hutter F, Hoos HH, et al (2011) Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In: RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the International Joint Conference on Artificial Intelligence (IJCAI), pp 16–30