Les Cahiers du GERAD ISSN: 0711–2440

Learning to branch for the crew pairing problem

P. Pereira, E. Courtade, D. Aloise, F. Quesnel, F. Soumis, Y. Yaakoubi

G–2022–31

July 2022

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Learning to branch for the crew pairing problem

Pierre Pereira [a, b]

Emeric Courtade [a, b]

Daniel Aloise [a, b]

Frédéric Quesnel [a, c]

François Soumis [a, c]

Yassine Yaakoubi [a, d]

[a] *GERAD, Montréal (Qc), Canada, H3T 1J4*

[b] *Département de génie informatique et génie logiciel, Polytechnique Montréal, Montréal (Qc), Canada, H3C 2A7*

[c] *Département de mathématiques et génie industriel, Polytechnique Montréal, Montréal (Qc), Canada, H3C 2A7*

[d] *COSMO (Stochastic Mine Planning Laboratory), Université McGill, Montréal (Qc), Canada, H3A 0E8*

pierre.pereira@polymtl.ca
emeric.courtade@polymtl.ca
daniel.aloise@polymtl.ca
frederic.quesnel@gerad.ca
francois.soumis@gerad.ca
yassine.yaakoubi@gerad.ca

July 2022
Les Cahiers du GERAD
G–2022–31

Abstract : Crew pairing problems (CPP) are regularly solved by airlines to produce crew schedules. The goal of CPPs is to find a set of pairings (sequence of flights and rests forming one or more workdays) that cover all flights in a given period at a minimum cost. The CPP is a NP-hard combinatorial problem that is usually solved approximately using a branch-and-price heuristic. A common branching heuristics selects the closest-to-one pairing variable and fixes it without backtracking, which allows finding good solutions quickly. By contrast, variable selection strategies based on strong branching typically produce better solutions but are computationally prohibitive. This paper explores the possibility of learning efficient branching strategies for diving heuristics from strong branching decisions. To help the learning process, we propose a new normalisation of the branching scores which is able to better distinguish the good and bad branching candidates. Our results demonstrate that our learnt strategies make better branching decisions than the greedy strategy while executing in approximately the same computational time.

Keywords : Crew pairing problem, heuristics, machine learning, branching strategy

Acknowledgements: We thank Quentin Cappart and Maxime Gasse for their valuable discussions and advice on the manner. This study was financially supported by Ad Opt. On behalf of all authors, the corresponding author states that there is no conflict of interest.

1 Introduction

Airline crew scheduling is an inherently difficult large-scale combinatorial optimization problem. Airlines have to plan ahead for the next month which flights their crew employees have to take, taking into account geographical and temporal constraints, as well as administrative constraints and crew preferences. Thus, the Crew Scheduling Problem (CSP) consists of finding a crew schedule that ensures all flights of a given period (typically a week or a month (Kasirzadeh et al., 2017)) are assigned to one pilot and copilot. Crew expenses are the second highest fees of an airline company, after fuel (Deveci and Çetin Demirel, 2018), and therefore, any small improvement can lead to significant savings.

The CSP is typically divided into two sub-problems: the Crew Pairing Problem (CPP) and the Crew Assignment Problem (CAP). A CPP solution consists of a set of anonymous pairings that cover all flights over a time period at minimal cost. A pairing is a feasible sequence of flights separated by connections, deadheads and rest periods. For a pairing to be feasible, it needs to start and finish at the same crew base, meet the temporal and spatial constraints of the flights, and comply with collective agreements and airline regulations. Given a set of pairings, the CAP assigns them to all crew members, thus yielding a valid crew schedule. The objective of the CAP is to maximize crew satisfaction while respecting some additional constraints (e.g. a specific language must be spoken by at least one crew on some flight (Quesnel et al., 2020)).

In this work, we focus our attention to the solution of the monthly CPP, which can be formulated as a set-partitioning problem. Typical instances of the CPP contain up to tens of thousands of flights. Due to the size of the problem, the CPP is often solved in the literature by means of a Branch-and-Price (B&P) algorithm (Kasirzadeh et al., 2017; Quesnel et al., 2020). B&P is a variant of the Branch-and-Bound algorithm where linear relaxations are solved using column generation. For the CPP, pairing candidates are generated by solving resource-constrained shortest-path sub-problems.

Because monthly CPP instances are too large to be optimally solved, solvers such as the state-of-the-art GENCOL (Desrosiers, 2010) use a diving heuristic to find a near-optimal integer solution (Lavoie et al., 1988; Deveci and Çetin Demirel, 2018). Such heuristic relies on a good *variable selection strategy* to fix pairings. One of those strategies is based on the variables fractional values (Sadykov et al., 2019; Quesnel et al., 2017), and quickly reaches a relatively good solution. By contrast, the *strong branching* strategy yields better final solutions(Sadykov et al., 2019), but is too computationally expensive to be used in practice.

Recently, Machine Learning (ML) has been more intensively used in the combinatorial optimization domain. ML exploits statistical redundancies of a family of problems to leverage good solution strategies to those problems. Using ML, we can consider producing a new branching strategy that might improve state-of-the-art combinatorial optimization solvers. Specifically, one common method is to use imitation learning to approximate a computationally expensive oracle by a simpler model (Alvarez et al., 2017; Gasse et al., 2019). The goal in this paper is to apply these ideas to our specific setting of diving B&P heuristics for CPPs. We do so by *proposing a new variable selection strategy for a diving heuristic* that relies on a ML model to select which pairing to fix at each node of the B&P tree. The ML model is trained offline in an imitation learning framework, using historical decisions from a strong branching diving heuristic as training data. Our contributions are:

- We demonstrate that imitation learning can be used to devise new data-driven diving B&P heuristics. To the best of our knowledge, this is the first time this is done in the literature.
- Our method relies on a new score normalization scheme that better distinguishes between good and bad branching candidates. We show that our ML-based variable selection strategy benefits from this new normalization.
- The diving branching heuristics based on those strategies produces better solutions than the traditional fractional branching algorithm, in a similar amount of time.

The rest of the paper is organized as follows. Section 2 presents a literature review of the CPP and of ML-based branching methods. Section 3 explains our methodology, Section 4 shows our results, and finally in Section 5, concluding remarks are given.

2 Literature review

2.1 Crew pairing problem

The CPP is typically formulated as a set-partitioning problem. Let $\mathcal{F}$ be the set of flights to be covered and Ω the set of all possibles pairings. The cost of a pairing p is given by c_p and a_{fp} is a binary coefficient that is equal to 1 if the pairing $p \in \Omega$ contains flight f, and 0 otherwise Finally, let χ_p be a variable equal to 1 if pairing p is selected, and 0 otherwise.

Thus, the CPP can be expressed as (1):

$$\min \sum_{p \in \Omega} c_p \chi_p \tag{1a}$$

$$\text{s.t.} \sum_{p \in \Omega} a_{fp} \chi_p = 1, \forall f \in \mathcal{F} \tag{1b}$$

$$\chi_p \in \{0, 1\}, \forall p \in \Omega \tag{1c}$$

For medium and large instances, Ω cannot be enumerated because it contains billions of pairings. Therefore, the CPP is commonly solved using a (B&P) algorithm (Desaulniers et al., 2020) : linear relaxations of the CPP are solved using column generation (Barnhart et al., 1998; Lavoie et al., 1988; Klabjan et al., 2001). Column generation works by iteratively solving a *restricted master problem* (RMP) and one or several *subproblems*. The RMP solves the linear relaxation of (1), restricted to only some pairings of Ω. The goal of the subproblems is to find new negative-reduced-cost pairings to add to the RMP. The algorithm stops when no reduced cost candidates can be generated, at which point the current RMP solution is optimal. A branch-and-bound algorithm is applied to reach integrality. The column generation algorithm is used to solve the linear relaxation at each node.

In practice, exploring the whole branch-and-bound tree would be too computationally expensive, so a branching heuristic is used to quickly produce a good (but not necessarily optimal) solution (Kasirzadeh et al., 2017). In particular, *diving branching heuristics* explore the branching tree in a depth-first fashion and stop when an integer solution is reached, without performing backtracking. A common type of branching decision for the CPP is column fixing, in which one or several pairing variables have their values fixed to one. There exist several variable selection strategies. The *fractional branching* heuristic (Sadykov et al., 2019; Quesnel et al., 2017) selects the variables with the highest fractional values. This heuristic is fast and yields good solutions in practice. To accelerate the diving process, it is possible to fix several candidates simultaneously as long as there are no conflicts between the pairings (no flight being covered more than once). This degrades the final solution but it is a powerful way of improving the computational time over the cost of the solution (Gamache et al., 1999)

The *strong branching* heuristic employs another variable selection strategy which yields better final solutions than fractional branching (Sadykov et al., 2019), but is too computationally expensive to be used in practice. It evaluates several candidate variables for branching, and then solves one relaxed version of the underlying CPP problem for each candidate by fixing their associated values to one. The child node yielding the lowest-cost fractional solution is then selected, and the others discarded. An illustration of the described variable selection strategies is shown in Figure 1. In the figure, the strong branching strategy chooses the candidate 3 because it has the lowest relaxed solution value s_3 among the three candidate nodes, whereas the fractional branching strategy would have selected the candidate 1 because of its maximum fractional value χ_1 among the candidate variables.

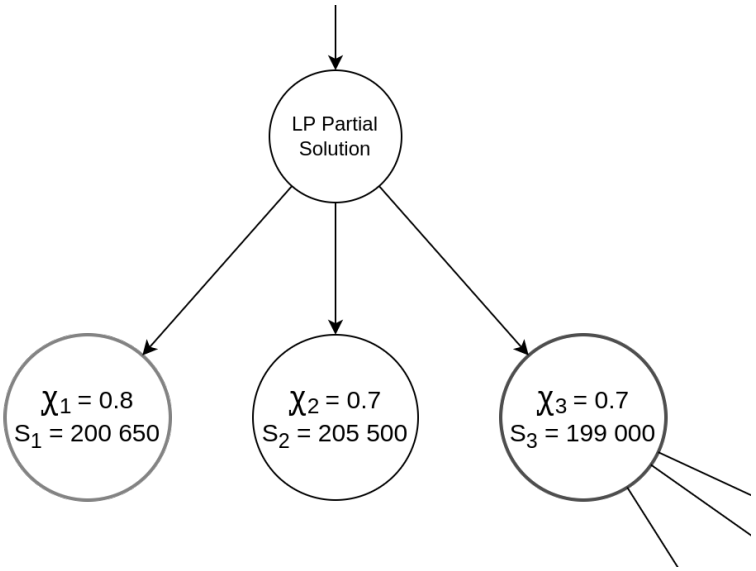

Figure 1: Example of branching selection for three candidate nodes

Although fractional branching can be combined with other variable selection strategies (Gamache et al., 1999; Quesnel et al., 2017), it remains the main branching strategy used to solve the CPP.

Other variable selection strategies exist, such as reliability branching and pseudocost branching (Achterberg et al., 2005), but they are not appropriate for a diving B&P heuristic where each variable is branched only once. A comparison between multiple diving branching strategies within B&P can be found in Sadykov et al. (2019).

We note that fixing a candidate to 0 is a decision of very low impact. Furthermore, such a decision is equivalent to forbidding a pairing, which is hard to do in column generation (although not impossible, see Villeneuve and Desaulniers (2005)). Therefore, we cannot use the product rule of Achterberg (2009) as it requires branching on both fixed candidate values, i.e., 0 and 1, to produce a score. Still, other types of branching rules can be found in the literature, such as *inter-task fixing* (Anbil et al., 1998; Desaulniers et al., 1997; Kasirzadeh et al., 2017), which forces or forbid two flights to appear consecutively in a pairing.

Recent developments include the work of Quesnel et al. (2017) where the authors proposed a *retrospective branching*, a branching heuristic which unfixes bad pairings during diving. Some works have been focused on the use of specific properties to the airline crew scheduling problem to improve the solver (Saddoune et al., 2013; Quesnel et al., 2020).

Other works looked into applying machine learning to accelerate column-generation-based solvers for crew scheduling problems. Yaakoubi et al. (2020) and Yaakoubi et al. (2021) use machine learning to produce initial solutions and modify clusters of the CPP for the column-generation based solver. Morabit et al. (2021) use a *Graph Neural Network* to select which pairings to keep during the column generation in order to reduce the time spent when reoptimizing the restricted master problem. In the same line of work, Quesnel et al. (2022) predict which pairings are useful by filtering the pairings during the column generation of a crew assignment problem. As far as we know, directly applying imitation learning to construct a new branching heuristic has never been done for the CPP.

2.2 Learning to branch

Alvarez et al. (2017) were the first to try *imitating* a branching heuristic. They learnt to approximate strong branching with *Extremely Random Trees* on the standard benchmark MIPLIB (Achterberg

et al., 2006). Khalil et al. (2016) modeled the task as a ranking problem, where the relative score of each candidate is not important. Their approach is also original as the model is instance-specific, i.e., it is learnt while solving the given problem. Gasse et al. (2019) modeled MIPs using a bipartite graph and used a *Graph Neural Network* to learn from strong branching decisions. This way of encoding MIPs reduced the number of hand-crafted features by means of a raw representation of the MIP. Nair et al. (2021) used this GNN idea to push it further and to learn heuristics for diving and branching. Since in practice, MIP solvers are not embedded in computers having GPUs for huge neural networks inference, Gupta et al. (2020) designed a cheaper framework that leverages the GNNs expressiveness while keeping it competitive when running inside a CPU-based solver. One recent innovation is the use of a parameterized search by Zarpellon et al. (2021) which combines two models. The first, called *NoTree*, is used to predict the value of a candidate based on its features. A second model, denoted *TreeGate*, is then used to encode the "tree state", thus adapting the inference of the first model. The second model allows better generalization over different sets of problems. Gupta et al. (2022) showed that it is often the case that when a candidate has been ranked number 2 by the strong branching heuristic, this candidate is the next best candidate for the next branching decision. The authors regularized their models by hardcoding this fact into their loss function, which resulted in faster solving times.

Other methods than imitation learning have been studied as well. Liu et al. (2022) used machine learning to tune the neighbourhood size of the *local branching* heuristic. They showed that the neighbourhood optimal size has to be adapted dynamically for each MIP and for each iteration of the search. Balcan et al. (2018) have learnt an adaptive weighting of different heuristics to combine them dynamically. They also provide a theoretical study of the learning to branch task. In this line of work, Chmiela et al. (2021) learn a scheduling of different well-known heuristics in order to rapidly find good feasible solutions.

Combining or improving known heuristics is a robust way of improving upon each heuristic separately but it does not allow for discovering new decision rules. Etheve et al. (2020) are the first to try to approach the learning to branch problem with reinforcement learning. This approach is attractive because it does not require any oracle (expert solver) and can potentially find new and better heuristics. Recently, Scavuzzo et al. (2022) built upon the work of Etheve et al. (2020) to improve the reinforcement learning framework by introducing *Tree MDPs* that accelerate the convergence and improve the branching decision quality of the model. Nevertheless, training such heuristics needs tons of data, which is hardly practicable for large-scale problems. A survey about learning to branch can be found in Lodi and Zarpellon (2017). Finally, other works about the use of machine learning for combinatorial optimization problems can be found in the survey of Bengio et al. (2021).

3 Methodology

This section presents the methodology used in our experiments. We first present the variable selection strategies used within diving B&P heuristics for CPPs. We then present our general methodology. Finally, we explain how the dataset is collected, what normalization we use, the different models trained and how our training and evaluation pipeline is conducted.

3.1 Variable selection strategies

As explained in Section 2.1, CPPs are typically solved via B&P (Lavoie et al., 1988; Barnhart et al., 1998). In this paper, we will compare our ML-based variable selection strategies with those used within the fractional and strong heuristics. The implemented fractional branching variant selects the single candidate χ_p having the highest (closest-to-one) fractional value and fixes it to 1. No computations are needed for this strategy as the fractional values are already computed from the parent node relaxation.

Our implementation of the strong branching variable selection strategy looks one step ahead by fixing and solving each candidate one by one, and then selects the candidate having the lowest relaxed optimal solution. Since we are interested in a feasible solution with the lowest possible cost, strong branching selects the candidate that increases the inferior bound the least. To reduce CPU times, it is common to evaluate a small subset of candidates. In our case, we only evaluate the top-5 candidates according to their fractional value. The drawback of strong branching is its computational overhead, though it is able to yield the best solutions.

3.2 Experimental framework

The goal of this paper is to imitate strong branching decisions with a faster approximation using the imitation learning framework (Alvarez et al., 2017). This new learned variable selection strategy is expected to produce better solutions than fractional branching while having the same computational complexity.

The framework is very similar to that of Alvarez et al. (2017). We use imitation learning to produce new strategies based on an oracle obtained from strong branching decisions. To develop our heuristic, we proceed as follows:

1. (Dataset generation) To collect data, we solve several CPP instances using the strong branching diving heuristic. At each strong branching evaluation step and for each candidate χ_i (among the top-5, according to their fractional value), we collect a set of features M_i and its strong branching score s_i. The dataset also contains the parent of each candidate so that sibling nodes can be grouped together and have their branching scores normalized together (see Section 3.2.3).
2. (Model training) A model is learned to predict a normalized version of the collected strong branching scores s_i based on the features M_i.
3. (Heuristic evaluation) During testing (evaluation), the trained model is used within the solver as a variable selection strategy. The resulting data-driven diving heuristic is therefore similar to fractional branching, except that the fixed variable is determined by the trained model rather than the fractional value of the variables.

An illustration of our methodological pipeline is shown in Figure 2.

3.2.1 Dataset collection

We used seven benchmark instances of the CPP literature from Kasirzadeh et al. (2017). A summary of the instances is shown in Table 1. They consist of monthly CPPs, and can be divided into two groups according to their size: instances 1, 2 and 3 are considered small-scale, while the remaining ones 4, 5, 6 and 7 are considered large-scale. They are solved using the diving strong branching heuristic, so that a set of features is collected for each branching candidate.

Table 1: The seven instances from Kasirzadeh et al. (2017) used to train and test our variable selection strategies

	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6	Instance 7
Flights	1013	1500	1854	5613	5743	5886	7765
Airports	26	35	41	49	34	52	54

The features M_i used in our experiments are listed in Table 2. We followed the work of Alvarez et al. (2017) by having three types of features:

- Static problem features (*nb_pairing_tasks, nb_cols_in_mp*).
- Dynamic problem features (*frac_val, var_cost, frac_conflicting_cols, min_cost_conflicting_cols, dual_cost*).
- Dynamic optimization features (*frac_pairing_tasks_fixed, depth*).

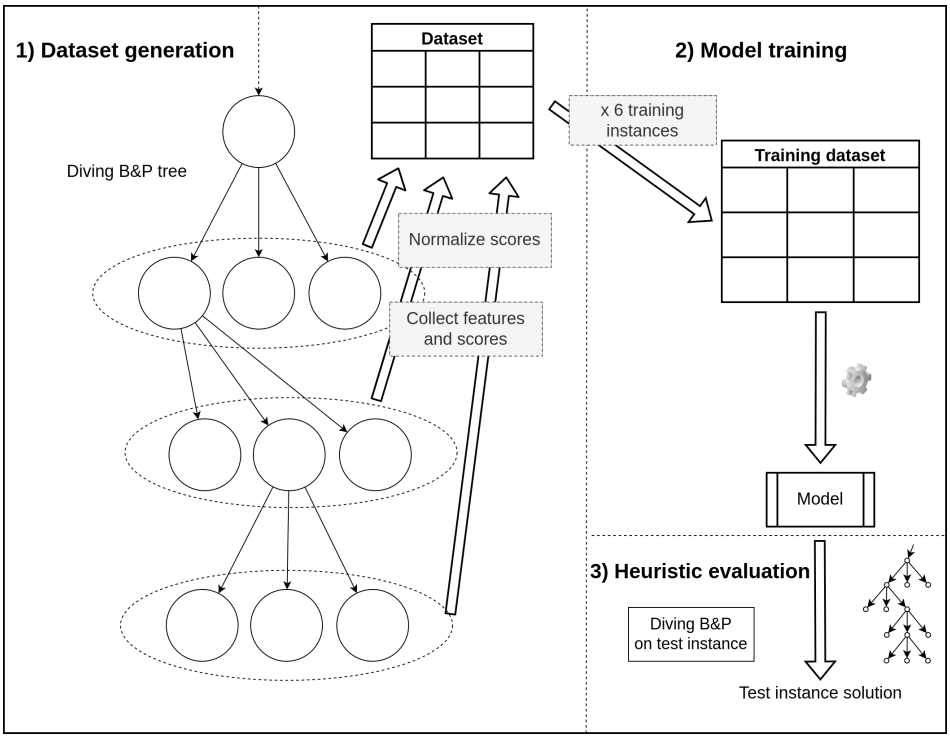

Figure 2: Our methodology pipeline

Nonetheless, some of our features are specific to CPPs (frac_pairing_tasks_fixed, nb_pairing_tasks). To limit the number of candidates, only the top-5 most promising candidates are retained according to their fractional value (i.e., the 5 candidates having the highest fractional χ_p value).

Table 2: Our collected features

Name	Description
frac_val χ_p	Fractional value of the candidate in the relaxed parent solution.
var_cost c_p	Cost of the candidate in the objective function.
frac_conflicting_cols	Fraction of conflicting columns with this candidate.
min_cost_conflicting_cols	Minimum cost in the objective function of the candidate's conflicting columns.
dual_cost (min, avg, max)	Minimum, average and maximum dual cost of the candidate among the partitioning constraints.
frac_pairing_tasks_fixed	Fraction of pairing tasks fixed.
nb_pairing_tasks	Number of pairing tasks.
nb_cols_in_mp	Number of columns in the master program.
depth	Depth of the candidate in the B&P tree.

3.2.2 Models, training and evaluation

Three ML models are used in our study.[1] The first one is a linear model which is simple enough so that it should not present generalization issues. The second one is a multi-layer perceptron (MLP) with 3 hidden layers and 150 neurons in each hidden layer. These two models take into account only the features of a candidate to predict its score. Finally, we train a third model which is a very small transformer encoder (Vaswani et al., 2017). It has 1 layer with an embedding size of 5, a feedforward hidden size of 10 and one attention head. We keep this transformer small in order to avoid overfitting. That model is able to consider all the candidates' features together to predict the score of a branching candidate.

[1]The code used to train these models can be found at https://github.com/Futurne/l2b_cpp.

A *leave-one-out* testing was used to assess the trained models, i.e., to test a model on one instance, we trained the model on all other instances. After the exclusion of the test instance, the remaining dataset is randomly split into training (80%) and validation (20%) data subsets to perform model selection. The trained model is then tested on the excluded instance within the solver.

We note that the goal of this paper is not to produce the best possible model for solving CPPs, but rather to investigate the possibility of learning a variable selection strategy for a diving branching heuristic by means of imitation learning. Therefore, a standard hyperparameter configuration was used without any particular fine-tuning: the training was performed for 100 epochs, using the ADAM optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-3} and the cross-entropy loss. We compute the validation loss on the validation dataset at each epoch. We kept as the final trained model the one having the lowest validation loss among the 100 epochs.

3.2.3 Normalizations of the strong branching scores

The score s_i of each candidate is the solution value of its relaxation once its corresponding fractional value has been fixed to 1. In order to train our models, we need to normalize this score so that the resulting normalized scores are comparable with each other. In doing so, we ensure that each normalized score falls within a fixed range of values, otherwise the range of s_i is affected by the underlying instance's size. Thus, we want the normalization to produce a meaningful ranking between the children nodes.

Recall that the model will be used to select among several candidates, the one that yields the lowest score s_i. However, we often observe that several candidates yield very similar scores. Thus, the ranking should be *soft* so that the normalized score reflects how good the node is with respect to its siblings. In other words, our model should not overly penalize branching decisions that are good but not best. Therefore, we need a normalization that keeps equivalent candidates together while distinguishing the bad from the good candidates. Indeed, evaluating the quality of such normalization is challenging.

Our procedure starts by separating all candidates χ_i according to their parent node, and builds a vector s with their associated scores s_i. The model task consists in predicting the normalized rank of every child candidate, so that a decision can be taken regarding branching on the best predicted candidate. The normalized rank of a candidate is expressed as a probability score $p(\chi_i \mid M_i)$ such that the normalized ranks of all children of a parent node sum up to 1.

We test two normalization functions to transform the branching scores. The first one is the tempered softmax from Nair et al. (2021):

$$p(\chi_i \mid M_i) = \frac{\exp(-s_i/t)}{\sum_{k \in K} \exp(-s_k/t)}, \tag{2}$$

where t is a positive constant temperature hyperparameter that needs to be set, otherwise the softmax saturates to 1 for the best candidate and to 0 for the others; K is the set of all children of the same parent node (χ_i being one of these children).

The expression (2) is the standard norm for such a machine learning task (Guo et al., 2020). The temperature t needs to be very well calibrated so that good candidates are grouped together. However, this normalization tends to bring good and bad candidates too close to one another, making it difficult for the ML predictor to provide a stable classification. In addition, this temperature is highly dependent on the instance at hand, making it difficult, if not impossible, to define a parameter value for each instance in advance.

To circumvent the above limitation, we propose an alternate norm (3) to better distinguish good and bad candidates.

$$y_i = \log s_i \tag{3a}$$

$$v_i = \frac{y_i - \bar{y}}{\sigma_y} \tag{3b}$$

$$p(\chi_i \mid M_i) = \frac{\exp(-v_i)}{\sum_{k \in K} \exp(-v_k)} \tag{3c}$$

where $\bar{y}$ is the mean value and σ_y is the standard deviation of the vector y.

The intuition behind this norm is to use the logarithm in (3a) to be less sensitive to the scale of the scores, and to use a Gaussian normalization in (3c) before applying the softmax, to obtain a more discriminating difference between good and bad candidates. Both of those objectives are learnt with a negative cross-entropy loss (Nair et al., 2021; Guo et al., 2020). The idea of the two norms is that they are soft objectives for the classifier, meaning that the model is not penalized too much if it predicts a good candidate even if it is not the best among all. Note that, in contrast to the tempered softmax norm in (2), the proposed norm in (3) does not require any hyperparameter.

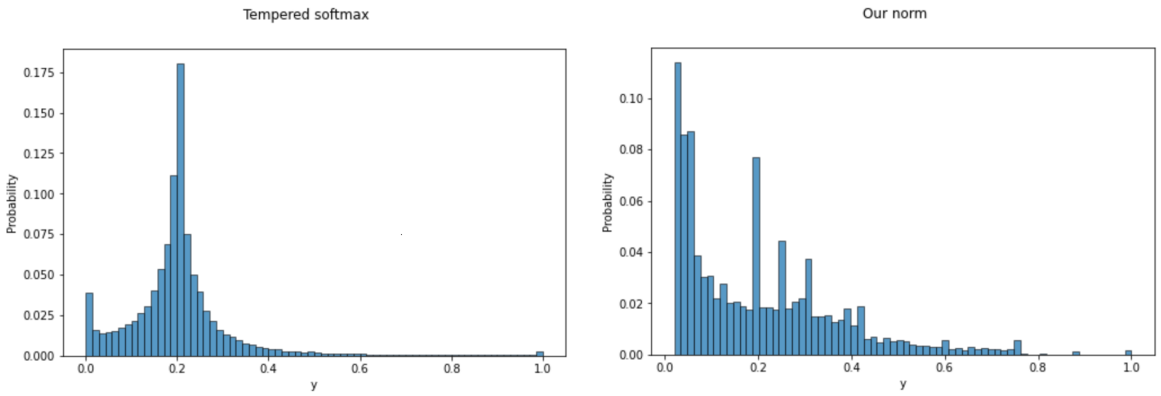

Figure 3: A comparison between the classical tempered softmax (2) (left) and our norm (3) (right)

Figure 3 shows a comparison between the distributions of the two normalizations. The tempered softmax distribution is centered around the mean value which is 0.2 since we always have 5 candidates. But it needs a well calibrated temperature to make sure that good and bad candidates are well separated. Our norm produces a more discriminating rank without being too aggressive. An average candidate will have a lower probability with our norm than with a well calibrated tempered softmax norm. This leads to a flatter distribution in the histogram as shown in Figure 3 where candidates are spread across the probability distribution. When there are 5 average candidates, they are centered around 0.2. When there is one candidate that is clearly better than the others its probability score is pushed around 0.6 to 0.8, and the four remaining bad candidates then have their probabilities around 0 to 0.1.

4 Computational results

To assess the performance of the proposed methodological pipeline, each model (linear, MLP, transformer encoder) is trained to imitate the strong branching scores, using either the tempered softmax (2) or the proposed norm (3) as the normalization function. For each model, we define a variable selection strategy that selects the candidate having the best predicted rank, according to the model. Those variable selection strategies are compared based on the quality of the proposed solutions and the execution time compared to those obtained using the strong branching heuristic.

The selection strategy based on the linear model is a direct improvement over the greedy strategy used by fractional branching, as the linear model has access to the fractional value of each candidate along with other features. The simplicity of this model prevents it from overfitting. In contrast, the

transformer is more expressive than the linear model, and if the model generalizes well, the corresponding variable selection strategy is expected to make better decisions. Finally, since the inference of a model is way quicker than optimizing the relaxed version of each candidate, we expect the overall runtime of the machine-learning-based diving heuristics to be on the same order of magnitude as the fractional branching heuristic, and significantly faster than the strong branching one.

All B&P heuristics were implemented inside a state-of-the-art CPP solver based on the GENCOL optimization software (Desrosiers, 2010). All our experiments were performed on computers equipped with Intel Xeon E3-1226 v3 3.30GHz CPUs. Because of the inherent randomness of the solver, each diving heuristic was evaluated over five distinct runs. Consequently, the following tables present average values computed from these executions.

Table 3 shows the relative solution values of all diving heuristics compared with respect to strong branching (lower is better). Every solution is obtained by using a specific variable selection strategy h within the diving B&P heuristic. We compare the solution value S_h obtained with a strategy h to the solution value S_{SB} obtained with the strong branching heuristic. This is computed as a ratio between the two, i.e., $\frac{S_h - S_{SB}}{S_{SB}}$. We show in Table 3 the percentage of this score.

Table 3: Solution values for each instance with respect to the strong branching strategy

Solution w.r.t. SB (%)	instance 1	instance 2	instance 3	instance 4	instance 5	instance 6	instance 7	mean
fractional branching	1.869	0.268	1.139	0.051	0.120	0.421	0.054	0.560
linear (our norm)	1.112	0.511	0.723	0.051	0.103	0.240	0.672	0.487
linear (softmax)	0.878	0.058	0.561	0.213	0.166	0.368	-0.043	0.314
MLP (our norm)	1.451	0.495	0.548	**-0.418**	0.213	0.473	0.146	0.415
MLP (softmax)	**0.681**	0.420	0.575	0.133	0.133	0.352	0.180	0.353
transformer (our norm)	0.916	0.420	**0.345**	-0.110	**0.066**	0.398	**-0.056**	**0.283**
transformer (softmax)	1.732	**-0.078**	0.892	0.565	0.265	**0.095**	0.121	0.513
mean (our models only)	1.128	0.304	0.607	0.072	0.158	0.321	0.170	

The data-driven variable selection strategies are on average better than that used by the fractional branching heuristic. As expected, a simple model such as the linear model already yields lower-cost solutions compared with those obtained with fractional branching. The best model is the transformer trained with the new norm. In fact, the gap between the solution values obtained by the transformer diving heuristic and the strong branching is on average two times smaller than the gap between the values obtained by the fractional branching heuristic and the strong branching. We also observe from the table that the variable selection strategies based on the other machine learning models also perform relatively well.

Moreover, we observe in Table 3 that the simpler models (linear and MLP) do not benefit from the new normalization as they did not improve the results compared to their counterparts trained with the tempered softmax norm. This could be due to the fact that the Gaussian distribution is easier to learn. Furthermore, these models can only evaluate the candidates separately. In contrast, the transformer has access to all 5 candidates for prediction, which means that it is able to better discriminate between good and bad candidates. This could explain why the transformer is able to take advantage of the more informative rank of the candidates even if the distribution is not Gaussian.

In Figure 4, we report the relative importance of the four most important features for the linear models trained with the new norm. Understanding which features contribute the most to the outcome of trained ML predictors can explain their behaviour and how they differ from other strategies. We measure the importance of each feature by taking the absolute value of its corresponding parameter, and then computing its relative percentage with respect to all the features. Features having an importance score smaller than 5% are grouped into the "others" category.

Notice that the *frac_val* value accounts for 47.1% of the prediction value. This explains why the performance of our heuristics are close to that of fractional branching. It also reveals that the fractional

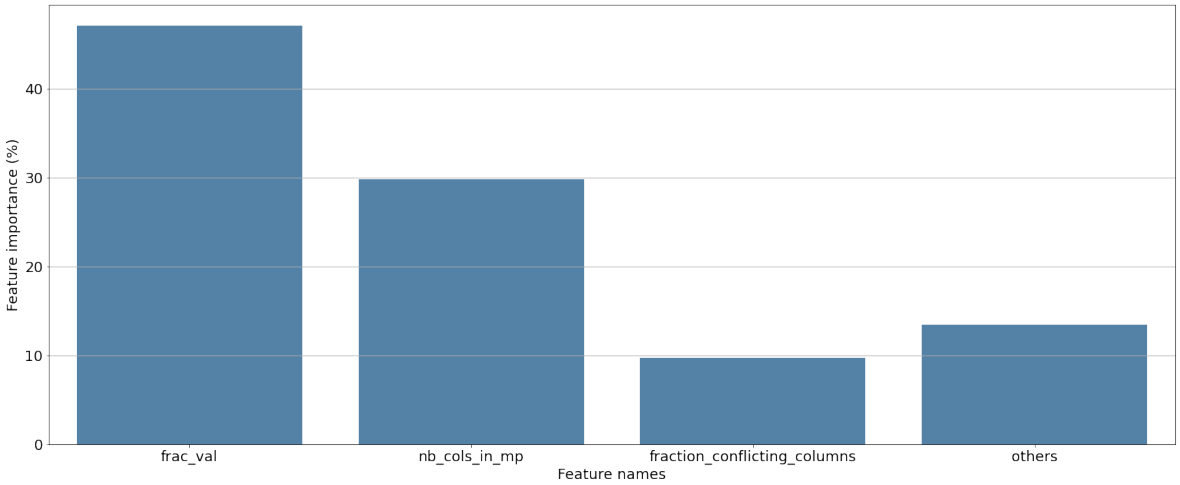

Figure 4: Feature importance of the linear model trained with our norm

branching strategy is already a good proxy for the strong branching decision, which explains why it has been successfully used in practice.

Finally, Table 4 shows the runtimes of the different diving heuristics. It also presents the relative difference with respect to strong branching in parentheses. The relative runtime is computed by comparing the computational runtime T_h of a given diving heuristic h to the runtime T_{SB} obtained with the strong branching heuristic. This value is computed as $\frac{\mathrm{T}_h - \mathrm{T}_{\mathrm{SB}}}{\mathrm{T}_{\mathrm{SB}}}$. The ML-based strategies are slightly slower than fractional branching. Nonetheless, they are significantly faster than strong branching. Unless the slight difference in execution time is unacceptable to an airline, the significant improvement in solution quality makes the proposed framework useful for airlines to reduce the cost of their CPP solutions.

5 Conclusions

In this work, we explored the use of machine learning for conceiving different data-driven variable selection strategies to be used within B&P diving heuristics. In particular, our strategies are learnt using imitation learning based on strong branching. This allows our heuristics to be as efficient as the fractional branching heuristic while making better branching decisions. Moreover, we also propose a new normalization for the strong branching scores that better distinguishes the good and the bad branching candidates. This norm shows better results than the standard normalization when used with a transformer, a model that is able to compare the branching candidates all together.

Finally, we note that our norm is harder to learn than the standard one because of its non-Gaussian distribution. A better norm would adapt the temperature hyperparameter of the standard norm based on the strong branching scores to make sure that good and bad candidates have well balanced probabilities after the softmax normalization. Besides, our branching decisions are limited by the candidates we consider at each stage of the tree. Since the trained models can only select one of the five pre-selected top candidates (according to their fractional values), we strongly prevent them from branching on bad candidates besides reducing their expressiveness. As future research venues, one could explore the use of graph neural networks (Gasse et al., 2019) with a raw representation of the candidates MIP models. Future research will also investigate the distributional shift that may occur when using such models, which can be sensitive to small feature variations, as well as ways to overcome that shift, for example by coupling graph neural networks and *DAgger* (Ross et al., 2011; Nair et al., 2021).

Table 4: Runtime for each instance with respect to the strong branching runtime

Time (s) (w.r.t. SB %)	instance 1	instance 2	instance 3	instance 4	instance 5	instance 6	instance 7	mean (%)
strong branching	178	227	889	40 481	44 639	51 478	109 365	–
fractional branching	**49 (-72.47)**	**55 (-75.75)**	218 (-75.50)	**9 037 (-77.68)**	**8 690 (-81.53)**	**11 450 (-77.76)**	**23 999 (-78.06)**	**-76.82**
linear (our norm)	52 (-70.79)	73 (-67.84)	246 (-72.35)	9 271 (-77.10)	9 880 (-77.87)	11 878 (-76.93)	26 560 (-75.71)	-74.08
linear (softmax)	57 (-67.98)	69 (-69.69)	236 (-73.48)	9 441 (-76.68)	9 833 (-77.97)	12 247 (-76.21)	24 777 (-77.34)	-74.19
MLP (our norm)	52 (-70.90)	71 (-68.55)	248 (-72.15)	9 743 (-75.93)	10 450 (-76.59)	12 406 (-75.90)	25 084 (-77.06)	-73.87
MLP (softmax)	57 (-68.75)	73 (-67.84)	246 (-72.31)	10 000 (-75.30)	10 013 (-77.57)	13 092 (-74.57)	26 191 (-76.05)	-73.06
transformer (our norm)	55 (-69.21)	62 (-72.69)	**217 (-75.57)**	10 713 (-73.93)	10 052 (-77.48)	12 027 (-76.64)	26 763 (-75.53)	-74.38
transformer (softmax)	54 (-69.78)	68 (-70.04)	244 (-72.60)	9 899 (-75.55)	10 739 (-76.94)	12 012 (-76.67)	25 142 (-77.01)	-73.94

References

Tobias Achterberg. Scip: solving constraint integer programs. Math. Prog. Comp., 1(1):1–41, July 2009. doi: 10.1007/s12532-008-0001-1.

Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. Oper. Res. Lett., 33(1): 42–54, January 2005. doi: 10.1016/j.orl.2004.04.002.

Tobias Achterberg, Thorsten Koch, and Alexander Martin. Miplib 2003. Operations Research Letters, 34(4): 361–372, 2006. doi: 10.1016/j.orl.2005.07.009.

Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. INFORMS J. Comput., January 2017. doi: 10.1287/ijoc.2016.0723.

Ranga Anbil, John Forrest, and William Pulleyblank. Column generation and the airline crew pairing problem. Documenta Mathematica, 35, 08 1998.

Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In Jennifer Dy and Andreas Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 344–353. PMLR, 10–15 Jul 2018.

Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. Oper. Res., June 1998. doi: 10.1287/opre.46.3.316.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research, 290(2):405–421, 2021. doi: 10.1016/j.ejor.2020.07.063.

Antonia Chmiela, Elias Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, Advances in Neural Information Processing Systems, volume 34, pages 24235–24246. Curran Associates, Inc., 2021.

G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M.M. Solomon, and F. Soumis. Crew pairing at air france. European Journal of Operational Research, 97(2):245–259, 1997. doi: 10.1016/S0377-2217(96)00195-6.

Guy Desaulniers, François Lessard, Mohammed Saddoune, and François Soumis. Dynamic constraint aggregation for solving very large-scale airline crew pairing problems. SN Oper. Res. Forum, 1(3):19–23, August 2020. doi: 10.1007/s43069-020-00016-1.

Jacques Desrosiers. Gencol: une équipe et un logiciel d'optimisation. Stud. Inform. Univ., 8:61–96, 01 2010.

Muhammet Deveci and Nihan Çetin Demirel. A survey of the literature on airline crew scheduling. Engineering Applications of Artificial Intelligence, 74:54–69, 2018. doi: 10.1016/j.engappai.2018.05.008.

Marc Etheve, Zacharie Alès, Côme Bissuel, Olivier Juan, and Safia Kedad-Sidhoum. Reinforcement learning for variable selection in a branch and bound algorithm. In Emmanuel Hebrard and Nysret Musliu, editors, Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pages 176–185, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58942-4.

Michel Gamache, François Soumis, Gérald Marquis, and Jacques Desrosiers. A column generation approach for large-scale aircrew rostering problems. Oper. Res., April 1999. doi: 10.1287/opre.47.2.247.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In Advances in Neural Information Processing Systems 32, 2019. doi: 10.48550/arXiv.1906.01629.

Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W. Bruce Croft, and Xueqi Cheng. A deep look into neural ranking models for information retrieval. Information Processing & Management, 57(6):102067, 2020. doi: 10.1016/j.ipm.2019.102067.

Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 18087–18097. Curran Associates, Inc., 2020.

Prateek Gupta, Elias B. Khalil, Didier Chetélat, Maxime Gasse, Yoshua Bengio, Andrea Lodi, and M. Pawan Kumar. Lookback for learning to branch. arXiv, June 2022. doi: 10.48550/arXiv.2206.14987.

Atoosa Kasirzadeh, Mohammed Saddoune, and François Soumis. Airline crew scheduling: models, algorithms, and data sets. EURO Journal on Transportation and Logistics, 6(2):111–137, 2017. doi: 10.1007/s13676-015-0080-x.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1), Feb. 2016. doi: 10.1609/aaai.v30i1.10080.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Diego Klabjan, Ellis L. Johnson, George L. Nemhauser, Eric Gelman, and Srini Ramaswamy. Solving large airline crew scheduling problems: Random pairing generation and strong branching. Comput. Optim. Appl., 20(1):73–91, October 2001. doi: 10.1023/A:1011223523191.

Sylvie Lavoie, Michel Minoux, and Edouard Odier. A new approach for crew pairing problems by column generation with an application to air transportation. European Journal of Operational Research, 35(1): 45–58, 1988. doi: 10.1016/0377-2217(88)90377-3.

Defeng Liu, Matteo Fischetti, and Andrea Lodi. Learning to search in local branching. Proceedings of the AAAI Conference on Artificial Intelligence, 36(4):3796–3803, Jun. 2022. doi: 10.1609/aaai.v36i4.20294.

Andrea Lodi and Giulia Zarpellon. On learning and branching: a survey. Top, 25(2):207–236, July 2017. doi: 10.1007/s11750-017-0451-6.

Mouad Morabit, Guy Desaulniers, and Andrea Lodi. Machine-learning-based column selection for column generation. Transportation Science, June 2021. doi: 10.1287/trsc.2021.1045.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks, 2021.

Frédéric Quesnel, Guy Desaulniers, and François Soumis. A new heuristic branching scheme for the crew pairing problem with base constraints. Computers & Operations Research, 80:159–172, 2017. doi: 10.1016/j.cor.2016.11.020.

Frédéric Quesnel, Guy Desaulniers, and François Soumis. A branch-and-price heuristic for the crew pairing problem with language constraints. European Journal of Operational Research, 283(3):1040–1054, 2020. doi: 10.1016/j.ejor.2019.11.043.

Frédéric Quesnel, Alice Wu, Guy Desaulniers, and François Soumis. Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering. Computers & Operations Research, 138:105554, 2022. doi: 10.1016/j.cor.2021.105554.

Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, volume 15 of Proceedings of Machine Learning Research, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

Mohammed Saddoune, Guy Desaulniers, and François Soumis. Aircrew pairings with possible repetitions of the same flight number. Computers & Operations Research, 40(3):805–814, March 2013. doi: 10.1016/j.cor.2010.11.003.

Ruslan Sadykov, François Vanderbeck, Artur Pessoa, Issam Tahiri, and Eduardo Uchoa. Primal heuristics for branch and price: The assets of diving methods. INFORMS J. Comput., April 2019. doi: 10.1287/ijoc.2018.0822.

Lara Scavuzzo, Feng Yang Chen, Didier Chételat, Maxime Gasse, Andrea Lodi, Neil Yorke-Smith, and Karen Aardal. Learning to branch with tree mdps, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc., 2017.

Daniel Villeneuve and Guy Desaulniers. The shortest path problem with forbidden paths. European Journal of Operational Research, 165(1):97–107, 2005. doi: 10.1016/j.ejor.2004.01.032.

Yassine Yaakoubi, François Soumis, and Simon Lacoste-Julien. Machine learning in airline crew pairing to construct initial clusters for dynamic constraint aggregation. EURO Journal on Transportation and Logistics, 9(4):100020, 2020. doi: 10.1016/j.ejtl.2020.100020.

Yassine Yaakoubi, Francois Soumis, and Simon Lacoste-Julien. Structured convolutional kernel networks for airline crew scheduling. In Marina Meila and Tong Zhang, editors, Proceedings of the 38th International Conference on Machine Learning, volume 139 of Proceedings of Machine Learning Research, pages 11626–11636. PMLR, 18–24 Jul 2021.

Giulia Zarpellon, Jason Jo, Andrea Lodi, and Yoshua Bengio. Parameterizing branch-and-bound search trees to learn branching policies. Proceedings of the AAAI Conference on Artificial Intelligence, 35(5):3931–3939, May 2021.