Optimal Decision Trees for the Algorithm Selection Problem: Integer Programming Based Approaches

Matheus Guedes Vilas Boas *[1], Haroldo Gambini Santos †[1,2], Luiz Henrique de Campos Merschmann‡[3], and Greet Vanden Berghe§[2]

[1]Department of Computing, Federal University of Ouro Pretoo, St. Diogo de Vasconcelos 328, 35400-000, Ouro Preto, MG, Brazil
[2]Computer Science Technology Campus, University of KU Leuven, 9000, Ghent, Belgium
[3]Department of Computer Science, Federal University of Lavras, P.O.Box 3037, 37200-000, Lavras, MG, Brazil

September 25, 2019

arXiv:1907.02211v3 [cs.LG] 24 Sep 2019

Abstract

Even though it is well known that for most relevant computational problems different algorithms may perform better on different classes of problem instances, most researchers still focus on determining a single best algorithmic configuration based on aggregate results such as the average. In this paper, we propose Integer Programming based approaches to build decision trees for the Algorithm Selection Problem. These techniques allow automate three crucial decisions: (*i*) discerning the most important problem features to determine problem classes; (*ii*) grouping the problems into classes and (*iii*) select the best algorithm configuration for each class. To evaluate this new approach, extensive computational experiments were executed using the linear programming algorithms implemented in the COIN-OR Branch & Cut solver across a comprehensive set of instances, including all MIPLIB benchmark instances. The results exceeded our expectations. While selecting the single best parameter setting across all instances decreased the total running time by 22%, our approach decreased the total running time by 40% on average across 10-fold cross validation experiments. These results indicate that our method generalizes quite well and does not overfit.

1 Introduction

Given that different algorithms may perform better on different classes of problems, Rice (1976) proposed a formal definition of the Algorithm Selection Problem (ASP). The main components of this problem are depicted in Fig. 1. Formally the ASP has the following input data:

$\mathscr{P}$: the problem space, a probably very large and diverse set of different problem instances; these instances have a number of characteristics, i.e. for linear programming, each possible constraint matrix defines a different problem instance;

$\mathscr{A}$: the algorithm space, the set of available algorithms to solve instances of problem $\mathscr{P}$; since many algorithms have parameters that significantly change their behavior, differently from Rice (1976), we consider that each element in $\mathscr{A}$ is an algorithm with a specific parameter setting; thus, selecting the best algorithm also involves selecting the best parameter tuning for this algorithm;

$\mathscr{F}$: the feature space; ideally elements of $\mathscr{F}$ have a significantly lower dimension than elements of $\mathscr{P}$, since not every problem instance influences the selection of the best algorithm; these features are also important to cluster problems having a common best algorithm, e.g., for linear programming some algorithms are known for performing well on problems with a dense constraint matrix;

*matheusgueedes91@gmail.com
†haroldo@ufop.edu.br
‡luiz.hcm@dcc.ufla.br
§greet.vanden.berghe@kuleuven.be

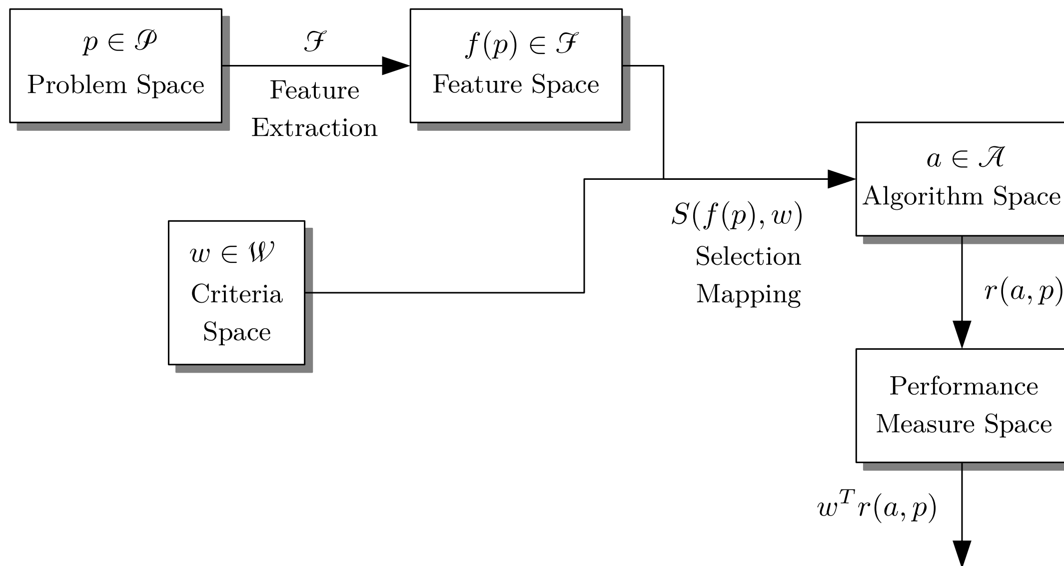

Figure 1: The Algorithm Selection Problem (Rice, 1976)

$\mathcal{W}$: the criteria space, since algorithms can be evaluated with different criteria, such as processing time, memory consumption and simplicity, the evaluation of the execution results $r = \mathbb{R}^n$ produced using an algorithm a to solve a problem instance p may be computed using a weight vector $w \in \mathcal{W} = [0, 1]^n$ which describes the relative importance of each criterion.

The objective is to define a function S that, considering problem features, maps problem instances to the best performing algorithms. This function is a mapping function that always selects the best algorithm for every instance. Thus, if B is the ideal function, the objective is to define S minimizing:

$$\sum_{p \in \mathcal{P}} |w^T r(B(f(p), w)) - w^T r(S(f(p), w))| \quad (1)$$

The main motivation for solving the ASP is that usually there is no *best algorithm* in the general sense: even though some algorithms may perform better on average, usually some algorithms perform much better than others for some groups of instances. A "winner-take-all" approach will probably discard algorithms that perform poorly on average, even if they produce excellent results for a small, but still relevant, group of instances.

This paper investigates the construction of S using decision trees. The use of decision trees to compute S was one of the suggestions included in the seminal paper of Rice (1976). To the best of our knowledge however, the use of decision trees for algorithm selection was mostly ignored in the literature. One recent exception is the work of Polyakovskiy et al. (2014) who evaluated many heuristics for the traveling thief problem and built a decision tree for algorithm recommendation. Polyakovskiy et al. (2014) did not report which algorithm was used to build this tree, but did note that the MatLab® Statistics Toolbox was used to produce an initial tree that was subsequently pruned to produce a compact tree. This is an important consideration: even though deep decision trees can achieve 100% of accuracy in the training dataset, they usually overfit, achieving low accuracy when predicting the class of new instances. The production of compact and accurate decision trees is an NP-Hard problem (Hyafil and Rivest, 1976). Thus, many greedy heuristics have been proposed, such as ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993) and CART (Breiman et al., 1984). These heuristics recursively analyze each split in isolation and proceed recursively. Recently, (Bertsimas and Dunn, 2017) proposed Integer Programming for producing *optimal* decision trees for classification. Thus, the entire decision tree is evaluated to reach global optimality. Their results showed that much better classification trees were produced for an extensive test set. This result was somewhat unexpected since there is a popular belief that optimum decision trees could overfit at the expense of generalization. Trees are not only the organizational basis of many machine learning methods, but also an important structural information (Zhang et al., 2018). The main advantage of methods that produce a tree as result is the interpretability of the produced model, an important feature in some applications such as healthcare.

At this point it is important to clarify the relationship between decision trees for the ASP and decision trees for *classification* and *regression*, their most common applications. Although the ASP can be seen as the classification problem of selecting the best algorithm for each instance, this modeling does not capture some

important problem aspects. Firstly, it is often the case that many algorithms may produce equivalent results for a given instance, a complication which can be remedied by using multi-label classification algorithms (Tsoumakas and Katakis, 2007). Secondly, the evaluation of the individual decisions of a classification algorithm always returns zero or one for incorrect and correct predictions, respectively. In the ASP each decision is evaluated according to a real number which indicates *how far* the performance of the suggested algorithm is from the performance of the best algorithm, as stated in the objective function (1). Thus, the construction of optimal decision trees for the ASP can be seen as a generalization of the multi-label classification problem and is at least as hard. Another approach is to model the ASP as a regression problem, in which one attempts to predict the performance of each algorithm for a given instance and select the best one (Xu et al., 2008; Leyton-Brown et al., 2003b; Battistutta et al., 2017). In this approach, the cost of determining the recommended algorithm for a new instance grows proportionally to the number of available algorithms and the performance of the classification algorithm. By contrast, a decision tree with limited depth can recommend an algorithm in constant time. Another shortcoming of regression-based approaches is related to the loss of precision in solution evaluation: consider the case when the results produced by a regression algorithm are always the correct result *plus* some additional large constant value. Even though the ranking of the algorithms for a given instance would remain correct and the right algorithm would always be selected, this large constant would imply an (invalid) estimated error for the regression algorithm. The present paper therefore investigates the applicability of Integer Programming to build a mapping S with decision trees using the precise objective function (1) of the ASP.

It is also important to distinguish the ASP from the problem of discovering improved parameter settings. Popular software packages such as irace (López-Ibáñez et al., 2016) and ParamILS (Hutter et al., 2014a) embed several heuristics to guide the search of improved parameters to a parameter setting that, ideally, performs well across a large set of instances. During this search, parameter settings with a poor performance in an initial sampling of instances may be discarded. In the ASP, even parameter settings with poor results for many instances may be worth investigating since they may well be the best choice to a small group of instances with similar characteristics. Thus, exploring parameter settings for the ASP may be significantly more computationally expensive than finding the best parameter setting on average, requiring more exhaustive computational experiments. An intermediate approach was proposed by Kadioglu et al. (2010): initially the instance set is divided into clusters and then the parameter settings search begins. One shortcoming of this approach is the requirement of an *a priori* distance metric to cluster instances. It can be hard to decide which instance features may be more influential for the parameter setting phase before the results of an initial batch of experiments is available. Optimized decision trees for the ASP provide important information regarding which instance features are more influential to parameter settings since these parameters will appear in the first levels of the decision tree. Also, instances are automatically clustered in the leaves. It is important to observe that an iterative approach is possible: after instances are clustered using a decision tree for the ASP, a parallel search for better parameters for instance groups may be executed, generating a new input for the ASP.

Another fundamental consideration is that the ASP is a *static* tuning technique: no runtime information is considered to dynamically change some of the suggested algorithm/parameter settings, as in the so called reactive methods (Mascia et al., 2014). The static approach has the advantage that usually no considerable additional computational effort is required to retrieve a recommended setting, but its success obviously depends on the construction of a sufficiently diverse set of problem instances for the training dataset to cover all relevant cases. After the assembly of this dataset, a possibly large set of experiments must be performed to collect the results of many different algorithmic approaches for each instance. Finally, a clever recommendation algorithm must be trained to determine relevant features for recommending the best parameters for new problem instances. Misir and Sebag (2017) tackle the problem of recommending algorithms with incomplete data, i.e., if the experiments results matrix is sparse and only a few algorithms were executed for each problem instance. In this paper we consider the more computationally expensive case, where for the training dataset all problem instances were evaluated on all algorithms.

This paper proposes the construction of optimal decision trees for the ASP using Integer Programming techniques. To accelerate the production of high quality feasible solutions, a variable neighborhood descent based mathematical programming heuristic was also developed. To validate our proposal, we set ourselves the challenging task of improving the performance of the COIN-OR Linear Programming Solver - CLP, which is the Linear Programming (LP) solver employed within the COIN-OR Branch & Cut - CBC solver (Lougee-Heimer, 2003). CLP is currently considered the *fastest* open source LP solver (Mittelmann, 2018; Gearhart et al., 2013). The LP solver is the main component in Integer Programming solvers (Atamtürk and Savelsbergh, 2005) and it is executed at every node of the search tree. Mixed-Integer Programing is the most successful technique to optimally solve NP-Hard problems and has been applied to a large number of problems, from production planning (Pochet and Wolsey, 2006) to prediction of protein structures (Zhu, 2007).

To the best of our knowledge, this is the first time that mathematical programming based methods

have been proposed and computationally evaluated for the ASP. As our results demonstrate, not only our algorithm produces more accurate predictions for the best algorithm with respect to unknown instances, considering a 10-fold validation process (Section 4.3) but it also has the distinct advantages of recommending algorithms in constant time and producing easily interpretable results.

The remainder of the paper is organized as follows: Section 2 presents the Integer Programming formulation for the construction of optimal decision trees for the ASP. Section 3 presents a variable neighborhood descent based mathematical programming heuristic. Our extensive computational experiments and their results are presented in Section 4 and, finally, Section 5 presents the results and provides some future research directions.

2 Optimal Decision Trees for the Algorithm Selection Problem

This section presents our integer programming model proposed for the construction of optimal decision trees for the ASP. The sets and parameters are described in Section 2.1. The corresponding decision variables are described in Section 2.2. The objective function associated with the problem and the constraints are described in Section 2.3.

2.1 Input data

$\mathcal{P}$ set of problem instances $= \{1, \ldots, \overline{p}\}$;

$\mathcal{A}$ set of available algorithms with parameter settings $= \{1, \ldots, \overline{a}\}$;

$\mathcal{F}$ set of instance features $= \{1, \ldots, \overline{f}\}$;

$\mathcal{C}_f$ set of valid branching values for feature f, $C_f = \{1, \ldots, \overline{c}_f\}$, $\overline{c}_f$ is at most $\overline{p}$ when all instances have different values for feature f;

d maximum tree depth;

τ threshold indicating a minimum number of instances per leaf node;

β penalty incorporated into the objective function when a leaf node contains a number of problem instances smaller than threshold τ;

$v_{p,f}$ value of feature f for problem instance p;

$g_{l,n}$ parameter that indicates which is the parent node of a given node n (considering a child node n where its parent is at its left);

$h_{l,n}$ parameter that indicates which is the parent node of a given node n (considering a child node n where its parent is at its right);

The maximum allowed tree depth is defined by d. To prevent overfitting, an additional cost (parameter β) is included into the objective function to penalize the occurrence of leaf nodes containing a number of problem instances smaller than threshold τ.

Parameters $v_{p,f}$ indicate the value of each feature f for each problem instance p. Parameters $g_{l,n}$ and $h_{l,n}$ indicate the parent node of a given node located at the left or right, respectively. Thus, if the parent of the node n is at left ($n \mod 2 = 0$), then $g_{l,n} = \lfloor (n+1)/2 \rfloor$, otherwise $g_{l,n} = -1$. Similarly, if the parent of the node n is at the right ($n \mod 2 = 1$), then $h_{l,n} = \lfloor (n+1)/2 \rfloor$, otherwise, $h_{l,n} = -1$.

2.2 Decision variables

The main decision variables $x_{l,n,f,c}$ are related to the feature and branch values at each branching node of the decision tree. From the choices defined by the model, the problem instances are grouped (variables $y_{l,n,p}$) according to the features and the cut-off points that were imposed on the branches. The model will determine the best algorithm for each group of problem instances placed on each leaf node (variables $z_{n,a}$). Variables u_n are used to check if there are problem instances allocated to a given leaf node. This set will be linked to the set of variables m_n - explained later in this section.

$$x_{l,n,f,c} = \begin{cases} 1, \text{if feature } f \in \mathcal{F} \text{ and cut-off point } c \in \mathcal{C}_f \text{ is used for node } n \in \{1,\ldots,2^l\} \\ \qquad \text{of level } l \in \{0,\ldots,(d-1)\}. \\ 0, \text{otherwise.} \end{cases}$$

$$y_{l,n,p} = \begin{cases} 1, \text{if problem instance } p \in \mathcal{P} \text{ is included for node } n \in \{1,\ldots,2^l\} \\ \qquad \text{of level } l \in \{1,\ldots,d\}. \\ 0, \text{otherwise.} \end{cases}$$

$$z_{n,a} = \begin{cases} 1, \text{if algorithm } a \in \mathcal{A} \text{ is used in the leaf node } n \in \{1,\ldots,2^d\}. \\ 0, \text{otherwise.} \end{cases}$$

$$u_n = \begin{cases} 1, \text{if leaf node } n \in \{1,\ldots,2^d\} \text{ has problem instances.} \\ 0, \text{otherwise.} \end{cases}$$

The next two sets of decision variables are used in the objective function. With the exception of set m_n ($m_n \in \mathbb{Z}^+$), all other sets of variables are binary. To penalize leaf nodes with few instances, which could result in overfitting, variables m_n are used to compute the number of problem instances that are missing for the leaf node n to reach a pre-established threshold of problem instances per leaf node, determined by the parameter τ. The set of decision variables $w_{p,n,a}$ is responsible for connecting the sets of decision variables $y_{l,n,p}$ and $z_{n,a}$, i.e., to ensure that all problem instances allocated to a particular leaf node have the same recommended algorithm and that this algorithm is exactly the one corresponding to $z_{n,a}$. In addition, the connection between the set $w_{p,n,a}$ and $y_{l,n,p}$ ensures that the problem instances allocated to leaf nodes respect branching decisions on parent nodes.

$$m_n = \begin{cases} \text{number of problem instances missing from the leaf node } n \text{ to reach} \\ \qquad \text{a pre-established threshold of problem instances per leaf node.} \end{cases}$$

$$w_{p,n,a} = \begin{cases} 1, \text{if problem instance } p \in \mathcal{P} \text{ is selected for leaf node } n \in \{1,\ldots,2^d\} \\ \qquad \text{with algorithm } a \in \mathcal{A}. \\ 0, \text{otherwise.} \end{cases}$$

2.3 Objective function and constraints

The objective of our model is to construct a tree of determined maximum depth that minimizes the distance of the performance obtained using the recommended algorithm from the ideal performance for each problem p. Here we consider that this non-negative value is already computed in r. There is an additional cost involved in the objective function to penalize the occurrence of leaf nodes with only a few supporting instances. Follows the objective function (2) and the set of constraints (3-18) of our model:

$$min \sum_{n=1}^{2^d} \sum_{p=1}^{\mathcal{P}} \sum_{a=1}^{\mathcal{A}} r_{p,a} \times w_{p,n,a} + \sum_{n=1}^{2^d} \beta \times m_n \tag{2}$$

subject to

$$\sum_{f \in \mathcal{F}} \sum_{c \in \mathcal{C}_f} x_{l,n,f,c} = 1 \qquad \forall\, l \in \{0, \ldots, (d-1)\},\, n \in \{1, \ldots, 2^l\} \tag{3}$$

$$\sum_{n=1}^{2^d} \sum_{a=1}^{\mathcal{A}} w_{p,n,a} = 1 \qquad \forall\, p \in \mathcal{P} \tag{4}$$

$$\sum_{a \in \mathcal{A}} z_{n,a} = 1 \qquad \forall\, n \in \{1, \ldots, 2^d\} \tag{5}$$

$$w_{p,n,a} \leq z_{n,a} \qquad \forall\, p \in \mathcal{P},\, n \in \{1, \ldots, 2^d\},\, a \in \mathcal{A} \tag{6}$$

$$w_{p,n,a} \leq y_{d,n,p} \qquad \forall\, p \in \mathcal{P},\, n \in \{1, \ldots, 2^d\},\, a \in \mathcal{A} \tag{7}$$

$$u_n \geq y_{d,n,p} \qquad \forall\, n \in \{1, \ldots, 2^d\},\, p \in \mathcal{P} \tag{8}$$

$$\sum_{p \in \mathcal{P}} y_{d,n,p} + m_n \geq \tau \times u_n \qquad \forall\, n \in \{1, \ldots, 2^d\} \tag{9}$$

$$y_{l,n,p} \leq y_{(l-1),max(g_{l,n},h_{l,n}),p} \qquad \forall\, l \in \{2, \ldots, d\},\, n \in \{1, \ldots, 2^l\},\, p \in \mathcal{P} \tag{10}$$

$$y_{l,n,p} \leq 1 - x_{(l-1),g_{l,n},f,c} \qquad \forall\, l \in \{1, \ldots, d\},\, n \in \{1, \ldots, 2^l\},\ p \in \mathcal{P},\, f \in \mathcal{F},\, c \in \mathcal{C}_f : g_{l,n} \neq -1 \wedge v_{p,f} \leq c \tag{11}$$

$$y_{l,n,p} \leq 1 - x_{(l-1),h_{l,n},f,c} \qquad \forall\, l \in \{1, \ldots, d\}, n \in \{1, \ldots, 2^l\},\ p \in \mathcal{P},\, f \in \mathcal{F},\, c \in \mathcal{C}_f : h_{l,n} \neq -1 \wedge v_{p,f} > c \tag{12}$$

$$x_{l,n,f,c} \in \{0, 1\} \qquad \forall\, l \in \{0, \ldots, (d-1)\},\, n \in \{1, \ldots, 2^l\},\, f \in \mathcal{F},\, c \in \mathcal{C} \tag{13}$$

$$y_{l,n,p} \in \{0, 1\} \qquad \forall\, l \in \{1, \ldots, d\},\, n \in \{1, \ldots, 2^l\},\, p \in \mathcal{P} \tag{14}$$

$$z_{n,a} \in \{0, 1\} \qquad \forall\, n \in \{1, \ldots, 2^d\},\, a \in \mathcal{A} \tag{15}$$

$$u_n \in \{0, 1\} \qquad \forall\, n \in \{1, \ldots, 2^d\} \tag{16}$$

$$w_{p,n,a} \in \{0, 1\} \qquad \forall\, p \in \mathcal{P},\, n \in \{1, \ldots, 2^d\},\, a \in \mathcal{A} \tag{17}$$

$$m_n \in \mathbb{Z}^+ \qquad \forall n \in \{1, \ldots, 2^d\} \tag{18}$$

Equations 3 ensure that each internal node of the tree must have exactly one feature and branching value selected. Each problem instance must be allocated to exactly one leaf node and one algorithm (Equations 4) and each leaf node must have exactly one associated algorithm (Equations 5). Inequalities 6 guarantee that the recommended algorithm for a leaf node is the same as the algorithm of the problem instances allocated to this node. Inequalities 7 guarantee that allocations of algorithms to problem instances are performed respecting the leaf node selection for each problem instance.

Constraint set 8 ensures that variables u_n are 1 if and only if there is at least one problem instance associated with leaf node n. Constraints 9 ensure that variable m_n is set to the number of problem instances missing from the leaf node n to reach the threshold τ. If $m_n = 0$, then the leaf node n contains at least τ problem instances.

Constraints 10 ensure that any problem instance allocated in a particular node must belong to the associated parent node. Finally, constraints 11 and 12 ensure that problem instances allocated in a particular node respect the feature and branching values selected at the parent node. Constraints 11 are generated when $g_{l,n} \neq -1$ and $v_{p,f} \leq c$ and ensure that problem instance p cannot be allocated at node n of level l ($y_{l,n,p} = 0$), when feature f and branching value c are chosen for its parent node ($x_{(l-1),g_{l,n},f,c} = 1$). Similarly, Constraints 12 are generated when $h_{l,n} \neq -1$ and $v_{p,f} > c$ and ensure that problem instance p cannot be allocated at node n of level l ($y_{l,n,p} = 0$), when feature f and branching value c are chosen for its parent node ($x_{(l-1),h_{l,n},f,c} = 1$). Constraints 13-18 are related to the domain of the decision variables defined in the model.

3 VND to accelerate the discovery of better solutions

The model proposed in Section 2 can be optimized by standalone Mixed-Integer Programming (MIP) solvers and in *finite time* the optimal solution for the ASP will be produced. Despite the continuous evolution of MIP solvers (Johnson et al., 2000; Gamrath et al., 2015), the optimization of large MIP models in restricted times, in the general case, is still challenging. Thus, we performed some scalability tests (Section 4.2) to check how practical it is the use of the our complete model to create optimal decision trees for the ASP in datasets of different sizes in limited times. Since our objective is to produce a method that can tackle large datasets of experiment results, we also propose a mathematical programming heuristic (Fischetti and

Fischetti, 2016) based on Variable Neighborhood Descent (VND) (Mladenović and Hansen, 1997) to speed the production of feasible solutions. VND is a local search method that consists of exploring the solution space through systematic change of neighborhood structures. Its success is based on the fact that different neighborhood structures do not usually have the same local minimum.

Algorithm 1 shows the pseudo-code for our approach called VND-ASP. The overall method operates as follows: a Greedy Randomized algorithm is employed to generate initial feasible solutions (GRC-ASP). Multiple runs of this constructive algorithm are used to construct an Elite Set of solutions. The best solution of this set is used in our VND local search (MIPSearch), where parts of this solution are fixed and the remaining parts are optimized with the MIP model presented previously. Also, a subset $\mathcal{Q}$ including at most q algorithms is built. It includes all algorithms that appear in the elite set $\mathcal{E}$ and additional algorithms selected with a MIP model as follows: a covering like model to select q algorithms is solved where each instance should be covered by at least $q' < q$ algorithms, minimizing the cost of covering each problem instance with the selected algorithm.

Algorithm 1 VND-ASP (r, $\hbar$, ℓ, D, d, α, m, n, $\mathcal{Q}$, $\mathcal{A}$, $\mathcal{P}$, q, q')

Input. matrix r: algorithm performance matrix; set D: all different branching values for all features ($\mathcal{F} \times \mathcal{C}_f$); set $\mathcal{A}$: set of algorithms; set $\mathcal{P}$: set of problems.

Parameters. $\hbar$: matheuristic execution timeout; ℓ: MIP search execution timeout; d: maximum depth; α: represents a continuous value between 0.1 and 1.0 that controls the greedy or random construction of the solution; m: maximum number of trees; n: maximum number of iterations without improvement; q: defines the number of algorithms of the set $\mathcal{A}$, q': minimum number of algorithms to cover each problem instance.

1: $\mathcal{E} \leftarrow \{\}$; $\mathcal{T} \leftarrow \{\}$; $st \leftarrow$ time()
2: $\mathcal{Q} \leftarrow$ algsubset (r, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{E}$, q, q')
3: GRC-ASP ($\mathcal{P}$, $\mathcal{A}$, r, $\mathcal{T}$, 0, D, d, 1.0, m, $\mathcal{E}$, $\mathcal{Q}$)
4: $i \leftarrow 0$
5: **while** ($i < n$) **do**
6: $\quad\mathcal{T} \leftarrow \{\}$
7: $\quad$GRC-ASP ($\mathcal{P}$, $\mathcal{A}$, r, $\mathcal{T}$, i, D, d, rand (0.1, 1.0), m, $\mathcal{E}$, $\mathcal{Q}$)
8: $\quad$**if** (PerformanceDegradation ($\mathcal{T}$) < HigherPerformanceDegradation ($\mathcal{E}$)) **then**
9: $\quad\quad$**if** (PerformanceDegradation ($\mathcal{T}$) < LowerPerformanceDegradation ($\mathcal{E}$)) **then**
10: $\quad\quad\quad i \leftarrow$ -1
11: $\quad\quad$**end if**
12: $\quad\quad i \leftarrow i + 1$
13: $\quad\quad$**if** ($\mathcal{T} \notin \mathcal{E}$) **then**
14: $\quad\quad\quad$**if** ($|\mathcal{E}| < m$) **then**
15: $\quad\quad\quad\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{\mathcal{T}\}$
16: $\quad\quad\quad$**else**
17: $\quad\quad\quad\quad \mathcal{s} \leftarrow$ SimilarityTrees ($\mathcal{E}$, $\mathcal{T}$); $\mathcal{E}_{\mathcal{s}} \leftarrow \mathcal{T}$
18: $\quad\quad\quad$**end if**
19: $\quad\quad\quad \mathcal{Q} \leftarrow \mathcal{Q} \cup$ AlgorithmsLeafNodes($\mathcal{T}$)
20: $\quad\quad$**end if**
21: $\quad$**else**
22: $\quad\quad i \leftarrow i + 1$
23: $\quad$**end if**
24: **end while**
25: Let s be the best solution of the set $\mathcal{E}$
26: Let $\mathcal{G}$ be the list of all subproblems in all neighborhoods
27: $\mathcal{G} \leftarrow$ Shuffle(); $k \leftarrow 1$; $ft \leftarrow$ time(); $et \leftarrow ft - st$
28: **while** ($k \leq |\mathcal{G}|$ **and** $et \leq \hbar$) **do**
29: $\quad s^{'} \leftarrow$ MIPSearch (s, k, ℓ, $\mathcal{Q}$. $\mathcal{G}$)
30: $\quad$**if** ($f(s^{'}) < f(s)$) **then**
31: $\quad\quad s \leftarrow s^{'}$; $k \leftarrow 1$; $\mathcal{G} \leftarrow$ Shuffle($\mathcal{G}$)
32: $\quad$**else**
33: $\quad\quad k \leftarrow k + 1$
34: $\quad$**end if**
35: $\quad ft \leftarrow$ time(); $et \leftarrow ft - st$
36: **end while**
37: Return s;

Our matheuristic has the following parameters: $\hbar$ indicates the time limit for running the entire algorithm; ℓ indicates the time limit for a single exploration of a sub-problem in the MIP based neighborhood search; d is the maximum tree depth; parameter $\alpha \in [0, 1]$ controls the randomization of the greedy algorithm; parameter m controls the maximum size of the set of trees $\mathcal{E}$; executions of the GRC-ASP algorithm are restricted to at most n iterations without updates in the elite set. Set D represents the different features and cut-off points of the problem instances.

The list $\mathcal{G}$ contains all sub-problems that can be obtained by fixing solution components in all neighborhoods. We shuffle these sub-problems in a list so that there is no priority for searching first in one neighborhood relative to another. This strategy is inspired by (Souza et al., 2010), where several neighborhood orders were in tested in a VND algorithm and the randomized order obtained better results. Whenever the incumbent solution is updated, the list $\mathcal{G}$ is shuffled again and the algorithm starts to explore it from the beginning.

In the following sections we describe in more details the algorithm used to generate initial feasible solutions (Section 3.1) and the MIP based neighborhoods employed in our algorithm (Section 3.2).

3.1 Constructive Algorithm

The initial solution s is obtained from the best solution of the set of trees $\mathcal{E}$. This set is obtained using a hybrid approach inspired by Quinlan (1993)'s C4.5 algorithm to generate a decision tree and the Greedy Randomized Constructive (GRC) (Resende and Ribeiro, 2014) search. GRC searches to a certain randomness in the greedy criterion adopted by the C4.5 algorithm. Algorithm 2 shows the hybrid approach. Lines 7-16 of Algorithm 2 (GRC-ASP) show the adaptation made in the C4.5 algorithm for use of the restricted candidate list of the GRC search.

Another adaptation considers the metric to split the nodes. Algorithm C4.5 uses **information gain** metric. This metric aims to choose the attribute that minimizes the impurity of the data. In a data set, it is a measure of the lack of homogeneity of the input data in relation to its classification. In our case, we used the **performance degradation** metric to split the nodes. This metric searches for the attribute that minimizes the degradation of performance obtained using the recommended algorithm from the ideal performance for each problem p.

Algorithm 2 GRC-ASP ($\mathcal{P}$, $\mathcal{A}$, r, $\mathcal{T}$, i, D, d, α, m, $\mathcal{E}$, $\mathcal{Q}$)

1: **if** (current depth $= d$) **then**
2:     terminate
3: **end if**
4: **for all** (f, c) in D **do**
5:     $\kappa_{f,c} \leftarrow$ PerformanceDegradationTree ($\mathcal{T}$, r, f, c)
6: **end for**
7: RCL $\leftarrow \emptyset$
8: $\underline{a} \leftarrow \max(\kappa_{f,c})$
9: $\overline{a} \leftarrow \min(\kappa_{f,c})$
10: $pdt \leftarrow \underline{a} + \alpha \times (\overline{a}$ - $\underline{a})$
11: **for all** (f, c) in D **do**
12:     **if** $\kappa_{f,c} \leq pdt$ **then**
13:         RCL $\leftarrow$ RCL $\cup$ (f, c)
14:     **end if**
15: **end for**
16: a_{rcl} = Randomly select a (f, c) from the RCL list.
17: t = Create a decision node in $\mathcal{T}$ that tests a_{rcl} in the root
18: D_l = Induced sub-dataset of r whose value of feature f are less than or equal to the cut-off point a_{rcl}
19: D_r = Induced sub-dataset of r whose value of feature f are greater cut-off point a_{rcl}
20: $\mathcal{T}_l$ = GRC-ASP ($\mathcal{P}$, $\mathcal{A}$, r, $\mathcal{T}$, i, D_l, d, α, m, n, $\mathcal{E}$, $\mathcal{Q}$)
21: Attach $\mathcal{T}_l$ to the corresponding branch of t
22: $\mathcal{T}_r$ = GRC-ASP ($\mathcal{P}$, $\mathcal{A}$, r, $\mathcal{T}$, i, D_r, d, α, m, n, $\mathcal{E}$, $\mathcal{Q}$)
23: Attach $\mathcal{T}_r$ to the corresponding branch of t

3.2 Neighborhoods

Since optimizing the complete MIP model may be too expensive, five neighborhoods have been designed to be employed in a fix-and-optimize context. Each neighborhood defines a set of subproblems to be optimized. These neighborhoods are explained in what follows, together with examples shown in Figs. 2-6. Examples

consider a decision tree with 4 levels: variables in gray are fixed and variables highlighted in black will be optimized. These neighborhoods are explored in a Variable Neighborhood Descent using the MIP solver in a fix-and-optimize strategy. Not only the neighborhoods, but the sub-problems of all neighboorhoods, are explored in a random order.

We will explain how the decision variables $x_{l,n,f,c}$, $y_{l,n,p}$ and $z_{n,a}$ are optimized in each of the neighborhoods. The following neighborhoods were developed:

- **Neighborhood** $\mathcal{n}_1$: optimizes the selection of the **feature** and the **cut-off point** of an internal node and consequently optimizes the allocation of problems in child nodes as well as optimizes the choice of the recommended algorithm in the respective leaf nodes.

  In the example of Figure 2, we consider optimizing the feature and cut-off point of internal node 2 at level 1 of the tree (binary variables $x_{1,2,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$). Since these variables determine the problems that will be allocated to the left and right child nodes, the binary variables $y_{2,3,1,\ldots,\overline{p}}$, $y_{2,4,1,\ldots,\overline{p}}$, $y_{3,5,1,\ldots,\overline{p}}$, $y_{3,6,1,\ldots,\overline{p}}$, $y_{3,7,1,\ldots,\overline{p}}$ and $y_{3,8,1,\ldots,\overline{p}}$ will also be optimized. Moreover, the recommended algorithm to the problems allocated in all child nodes in relation to the chosen node - internal node 2 of level 1 of the tree - which are leaf nodes should also be optimized. In our example, these would be the binary variables $z_{5,1,\ldots,\overline{a}}$, $z_{6,1,\ldots,\overline{a}}$, $z_{7,1,\ldots,\overline{a}}$ and $z_{8,1,\ldots,\overline{a}}$ of the leaf nodes 5,. . . ,8 of level 3 of the tree.

- **Neighborhood** $\mathcal{n}_2$: optimizes the selection of the **feature** and the **cut-off point** of an internal node (this node cannot be the root node) and optimizes the choice of the **feature** and the **cut-off point** of the associated **parent node**, it consequently optimizes the allocation of problems in child nodes of associated parent node as well as the choice of the recommended algorithm in the respective leaf nodes.

  In the example of Figure 3, we consider optimizing the feature and cut-off point of internal node 2 of level 2 of the tree (binary variables $x_{2,2,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$) and optimizing the feature and cut-off point of associated parent node (binary variables $x_{1,1,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$). Since these variables determine the problems that will be allocated to the left and right child nodes, the binary variables $y_{2,1,1,\ldots,\overline{p}}$, $y_{2,2,1,\ldots,\overline{p}}$, $y_{3,1,1,\ldots,\overline{p}}$, $y_{3,2,1,\ldots,\overline{p}}$, $y_{3,3,1,\ldots,\overline{p}}$ and $y_{3,4,1,\ldots,\overline{p}}$ will also be optimized. Moreover, the recommended algorithm to execute the problems allocated in all child nodes in relation to the associated parent node - internal node 1 of level 1 of the tree - which are leaf nodes should also be optimized. In our example, these would be the binary variables $z_{1,1,\ldots,\overline{a}}$, $z_{2,1,\ldots,\overline{a}}$, $z_{3,1,\ldots,\overline{a}}$ and $z_{4,1,\ldots,\overline{a}}$ of the leaf nodes 1,. . . ,4 at level 3 of the tree.

- **Neighborhood** $\mathcal{n}_3$: optimizes the selection of the **feature** and the **cut-off point** of all nodes at one level (the level of the root node cannot be chosen) of the decision tree. Consequently optimizes the allocation of the problems in the nodes of the subsequent levels to the chosen level, it in addition to the choice of the recommended algorithm in the respective leaf nodes.

  In the example of Figure 4, we consider optimizing the feature and cut-off point of all nodes of level 2 of the tree (binary variables $x_{2,1,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$, $x_{2,2,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$, $x_{2,3,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ and $x_{2,4,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$). Since these variables determine the problems that will be allocated at subsequent levels, the binary variables $y_{3,1,1,\ldots,\overline{p}}$, $y_{3,2,1,\ldots,\overline{p}}$, $y_{3,3,1,\ldots,\overline{p}}$, $y_{3,4,1,\ldots,\overline{p}}$, $y_{3,5,1,\ldots,\overline{p}}$, $y_{3,6,1,\ldots,\overline{p}}$, $y_{3,7,1,\ldots,\overline{p}}$ and $y_{3,8,1,\ldots,\overline{p}}$ will also be optimized. In addition, the recommended algorithm to execute the problems allocated in all leaf nodes should also be optimized. In our example, these would be binary variables $z_{1,1,\ldots,\overline{a}}$, $z_{2,1,\ldots,\overline{a}}$, $z_{3,1,\ldots,\overline{a}}$, $z_{4,1,\ldots,\overline{a}}$, $z_{5,1,\ldots,\overline{a}}$, $z_{6,1,\ldots,\overline{a}}$, $z_{7,1,\ldots,\overline{a}}$, and $z_{8,1,\ldots,\overline{a}}$ of leaf nodes 1,. . . ,8 at level 3 of the tree.

- **Neighborhood** $\mathcal{n}_4$: optimizes the selection of the **feature** and the **cut-off point** of the root node and the choice of the **feature** and the **cut-off point** of an internal node, so that this node is at least at the third level of the tree ($l = 2$). Consequently it optimizes the allocation of problems in all nodes of the tree as well as the choice of the recommended algorithm in the respective leaf nodes.

  In the example of Figure 5, we consider optimizing the feature and cut-off point of both the root node (binary variables $x_{0,1,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$) and optimizing internal node 2 at level 2 of the tree (binary variables $x_{2,2,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$). Since these variables determine the problems that will be allocated to all other nodes of the tree, binary variables $y_{1,1,1,\ldots,\overline{p}}$, $y_{1,2,1,\ldots,\overline{p}}$, $y_{2,1,1,\ldots,\overline{p}}$, $y_{2,2,1,\ldots,\overline{p}}$, $y_{2,3,1,\ldots,\overline{p}}$, $y_{2,4,1,\ldots,\overline{p}}$, $y_{3,1,1,\ldots,\overline{p}}$, $y_{3,2,1,\ldots,\overline{p}}$, $y_{3,3,1,\ldots,\overline{p}}$, $y_{3,4,1,\ldots,\overline{p}}$, $y_{3,5,1,\ldots,\overline{p}}$, $y_{3,6,1,\ldots,\overline{p}}$, $y_{3,7,1,\ldots,\overline{p}}$ and $y_{3,8,1,\ldots,\overline{p}}$ will also be optimized. In addition, the recommended algorithm to execute the problems allocated to all leaf nodes should also be optimized. In our example, these would be binary variables $z_{1,1,\ldots,\overline{a}}$, $z_{2,1,\ldots,\overline{a}}$, $z_{3,1,\ldots,\overline{a}}$, $z_{4,1,\ldots,\overline{a}}$, $z_{5,1,\ldots,\overline{a}}$, $z_{6,1,\ldots,\overline{a}}$, $z_{7,1,\ldots,\overline{a}}$, and $z_{8,1,\ldots,\overline{a}}$ of leaf nodes 1,. . . ,8 at level 3 of the tree.

- **Neighborhood** $\mathcal{n}_5$: optimizes the selection of the **feature** and the **cut-off point** of a particular path from the root node to one of the tree's leaf nodes. Consequently it both optimizes the allocation

of problems in all nodes of the tree and the choice of the recommended algorithm in the respective leaf nodes.

In the example of Figure 6, we consider the path from the root node to leaf node 8. We consider optimizing the feature and cut-off point of both the root node (binary variables $x_{0,1,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$), internal node 2 at level 1 of the tree (binary variables $x_{1,2,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$), and internal node 4 at level 2 of the tree (binary variables $x_{2,4,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$). Since these variables determine the problems that will be allocated to all other nodes of the tree, binary variables $y_{1,1,1,\ldots,\overline{p}}$, $y_{1,2,1,\ldots,\overline{p}}$, $y_{2,1,1,\ldots,\overline{p}}$, $y_{2,2,1,\ldots,\overline{p}}$, $y_{2,3,1,\ldots,\overline{p}}$, $y_{2,4,1,\ldots,\overline{p}}$, $y_{3,1,1,\ldots,\overline{p}}$, $y_{3,2,1,\ldots,\overline{p}}$, $y_{3,3,1,\ldots,\overline{p}}$, $y_{3,4,1,\ldots,\overline{p}}$, $y_{3,5,1,\ldots,\overline{p}}$, $y_{3,6,1,\ldots,\overline{p}}$, $y_{3,7,1,\ldots,\overline{p}}$ and $y_{3,8,1,\ldots,\overline{p}}$ will also be optimized. In addition, the recommended algorithm to execute the problems allocated to all leaf nodes should also be optimized. In our example, these would be binary variables $z_{1,1,\ldots,\overline{a}}$, $z_{2,1,\ldots,\overline{a}}$, $z_{3,1,\ldots,\overline{a}}$, $z_{4,1,\ldots,\overline{a}}$, $z_{5,1,\ldots,\overline{a}}$, $z_{6,1,\ldots,\overline{a}}$, $z_{7,1,\ldots,\overline{a}}$, and $z_{8,1,\ldots,\overline{a}}$ of leaf nodes 1,. . . ,8 at level 3 of the tree.

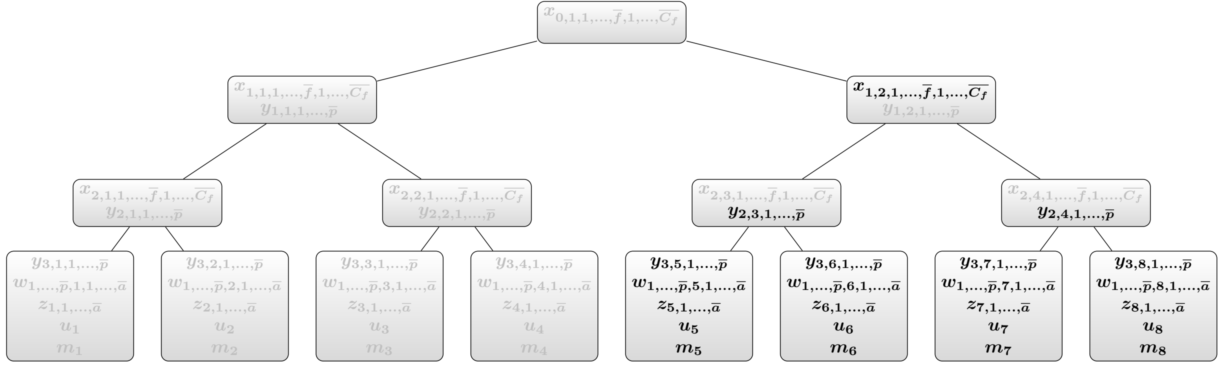

Figure 2: Example of neighborhood $\mathcal{N}_1$ variables: variables highlighted in gray are fixed and variables highlighted in black will be optimized.

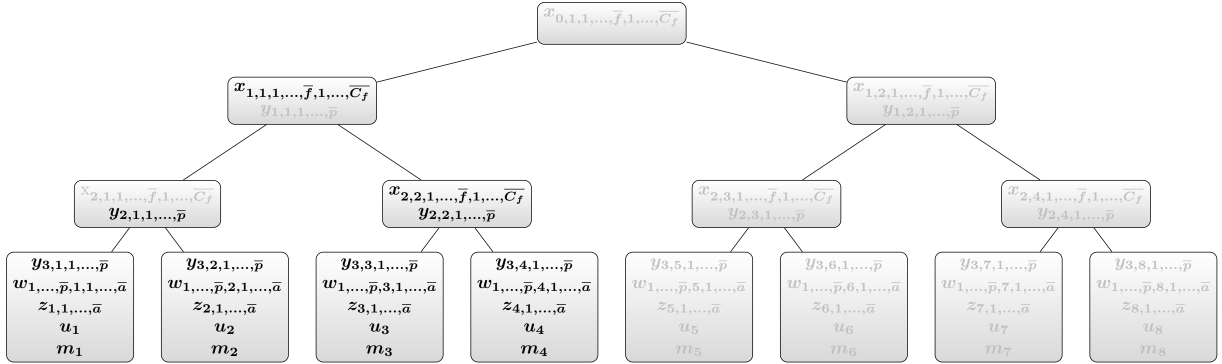

Figure 3: Example of neighborhood $\mathcal{N}_2$ variables: variables highlighted in gray are fixed and variables highlighted in black will be optimized.

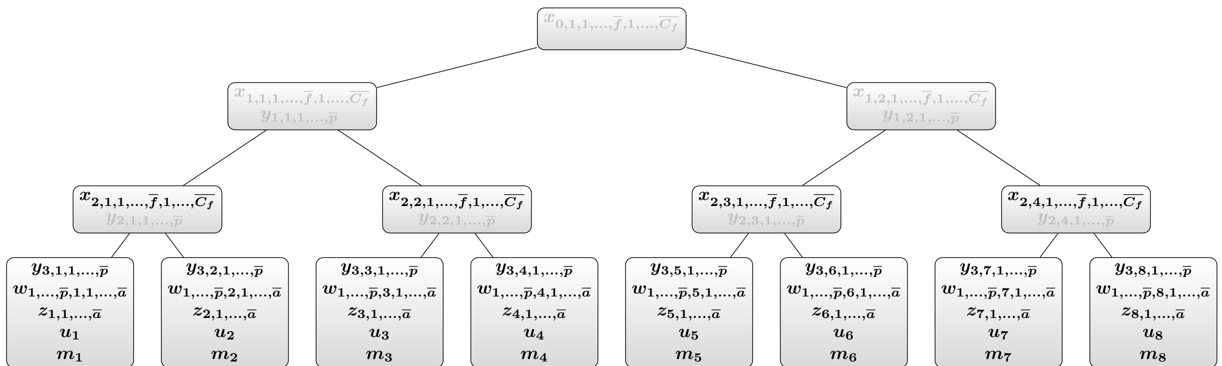

Figure 4: Example of neighborhood $\mathcal{N}_3$ variables: variables highlighted in gray are fixed and variables highlighted in black will be optimized.

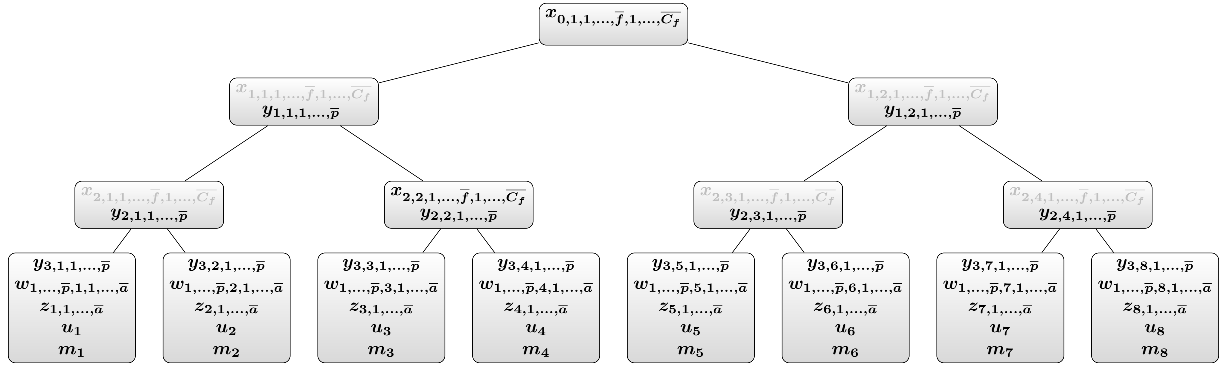

Figure 5: Example of neighborhood $\mathcal{N}_4$ variables: variables highlighted in gray are fixed and variables highlighted in black will be optimized.

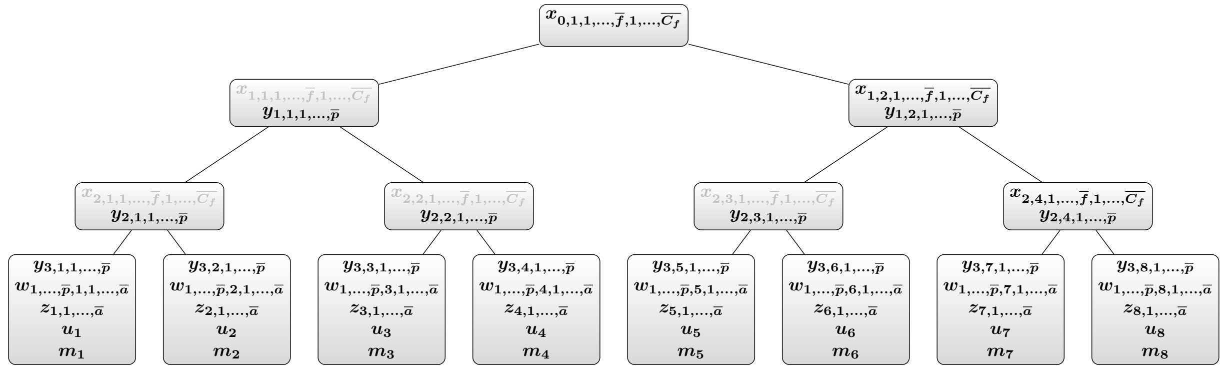

Figure 6: Example of neighborhood $\mathcal{N}_5$ variables: variables highlighted in gray are fixed and variables highlighted in black will be optimized.

4 Experiments

The computational experiments are divided into two groups: the first group (Section 4.2) is dedicated to check the scalability of our integer programming model using a standalone MIP solver in datasets with different numbers of problem instances and algorithms. The second group of experiments (Section 4.4) uses cross-validation and divides the complete base into 10 partitions, where training and test partitions are used. Section 4.1 presents the initial configurations for these two experiments.

4.1 Computational experiments dataset

This section describes how the experiments database was created. Details of the sets of problem instances and algorithms will be presented in Sections 4.1.1 and 4.1.2.

4.1.1 Problem instances

Computational experiments were performed for a diverse set of 1004 problem instances including the MIPLIB 3, 2003, 2010 and 2017 (Koch et al., 2011) benchmark sets. Additional instances from Nurse Rostering (Santos et al., 2016), School Timetabling (Fonseca et al., 2017) and Graph Drawing (Silva and Santos, 2017) were also included. We extracted 37 features ($|\mathcal{F}| = 37$) associated to variables, constraints and coefficients in the constraint matrix for problem instances. These features are similar to the ones used in (Hutter et al., 2014b) with the notable exception that features that are computationally expensive to extract were discarded to ensure that our approach would incur no overhead when incorporated into an algorithm. When building the problem instances dataset, special care was taken to ensure that no application was over-represented. Table 1 shows the minimum (min), maximum (max), average (avg) and standard deviation (sd) of each feature over the complete set of problem instances. The density feature was computed as $(\frac{nz}{rows \times cols}) \times 100$.

Table 1: Distribution of problem instances according to features

feature	min	max	avg	sd	feature	min	max	avg	sd
cols	3	2277736	35368.04	120402.72	rflowint	0	120201	428.23	4052.12
bin	0	2277736	25402.59	103781.75	rflowmx	0	410733	1813.33	17306.95
int	0	440899	2301.02	16289.21	rvbound	0	0	0	0
cont	0	799416	7664.43	47456.96	rother	0	2365080	26491.79	110176.92
objMin	-1.72E+11	5084550	-1.72E+08	5.431E+09	rhsMin	-1E+100	367127	-2.98E+97	5.45E+98
objMax	-400071	1.13E+09	4398763.75	5.9E+07	rhsMax	-6.47	1E+100	1.39E+98	1.17E+99
objAv	-2.54E+10	5.00E+07	-2.52E+07	8E+08	rhsAv	-5.24E+95	5.12E+098	5.13E+95	1.61E+97
objMed	-1.55E+08	40212300	-192578.19	7E+06	rhsMed	-29961800	999949	-33807.20	979700.89
objAllInt	0	1	0.70	0.46	rhsAllInt	0	1	0.82	0.39
objRatioLSA	-1	2.24E+12	2.448E+09	7.081E+10	rhsRatioLSA	-1	1.01E+103	2.02E+100	4.50E+101
rows	1	2897380	38440.14	146476.08	equalities	0	416449	5745.37	27584.05
rpart	0	18431	255.05	1084.42	nz	3	27329856	390757.39	1567477.92
rpack	0	773664	4598.49	44125.82	aMin	-4E+09	1	-4523186.29	1E+08
rcov	0	88452	381.60	3481.73	aMax	-1	370795000	1891549.90	2E+07
rcard	0	430	9.95	33.88	aAv	-107447000	1148410	-101191.22	3391461.70
rknp	0	103041	189.01	3356.63	aMed	-41014	10000	-25.70	1353.80
riknp	0	547200	2062.42	29593.52	aAllInt	0	1	0.69	0.46
rflowbin	0	381806	2210.27	19281.02	aRatioLSA	1	5.787E+12	6.117E+09	1.824E+11
density	0.0001819	100	5.44	17.50					

Fig. 7 summarizes the 37 features for ASP, grouped by features related to variables, constraints and coefficients in the constraint matrix.

Variable features:

1.	**Number of variables**: cols.
2 – 4.	**Number of variables of type**: bin, int and cont.
5 – 10.	**Variation of the objective function coefficient**:: objMin, objMax, objAv, objMed, objAllInt and objRatioLSA.
11 – 12.	**Number of non-zeros and density**: nz and density.

Constraint features:

11 – 12.	**Number of non-zeros and density**: nz and density.
13.	**Number of constraints**: rows.
14 – 25.	**Number of constraints of type**: rpart, rpack, rcov, rcard, rknp, riknp, rflowbin, rflowint, rflowmx, rvbound, rother and equalities.
26 – 31	**Right-hand Side Features**: rhsMin, rhsMax, rhsAv, rhsMed, rhsAllInt and rhsRatioLSA.

Coefficients in the constraint matrix features:

32 – 37.	**Variation of the coefficients**: aMin, aMax, aAv, aMed, aAllInt, aRatioLSA.

Figure 7: Features of problem instances of Algorithm Selection Problem: variables, constraints and coefficients in the constraint matrix.

4.1.2 Available algorithms

The definition of the solution method for the LP solver in CBC involves selecting the algorithm, such as dual simplex or the barrier and defining several parameters, such as the perturbation value and the pre-solve effort. Overall 532 different algorithm configurations were evaluated for each one of the 1004 problem instances . A timeout $T = 4000$ was set for each execution. The computational results matrix r was filled with the execution time for regular executions, i.e. executions that finished before the time limit and provided correct results. Executions for a given problem instance p and algorithm a that crashed, exceeded the time limit or produced wrong results were penalized by setting $r_{pa} = 8000$. This large batch of experiments was executed in computers with 32 Gb of RAM and 10 Intel ®i9-7900X processing cores. Tasks where scheduled in parallel (7 threads simultaneously) with the GNU Parallel package (Tange, 2011). Table 2 shows the algorithms and parameter values evaluated. This experiment to generate the experimental results dataset produced some interesting results itself: a new better single parameter setting was discovered that decrease the solution time by 22% in average, a remarkable improvement considering that CLP is already the fastest open source linear programming solver.

Table 2: Algorithms and some parameters values evaluated

$\mathcal{A}$	idiot	crash	dualize	pertv	sprint	primalp	psi	scal	passp	subs	perturb	presolve	spars
primal simplex	{3,4,5,6 7,9,10,11 15,20,25, 30,35,40, 50,60,80, 100}	{idiot1...7, on,lots,so}	{0,1,2, 4}	{-3500, -3157, -3000, -2395, -2000, -1483, -1000, 61}	{1217, 1557, 1804, 3384, 4826}	{change, exa}	{-0.84, -0.62, -0.35 0.62,0.66, 0.84,0.91}	{geo, off}	{-138, 22,40, 80,138}	{1,57, 251,270, 294}	{off}	{more, off}	{off}
default	-1	off	3	50	-1	auto	-0.5	auto	5	3	on	on	on
$\mathcal{A}$	**idiot**	**crash**	**dualize**	**pertv**	**sprint**	**dualp**	**psi**	**scal**	**passp**	**subs**	**perturb**	**presolve**	**spars**
duals simplex		{idiot1...7, on,lots,so}	{1}	{-4900, ...,820}	{0,468, 620, 1612, 2228}	{pesteep, steep}	{-1.1, ...,1.1} ,	{geo, rows}	{-167, -81, -67, -33, 0,36,67, 93}	{37, 40,41 297, 4354, 4392}	{off}	{more, off}	{off}
default		off	3	50	-1	auto	-0.5	auto	5	3	on	on	on
$\mathcal{A}$	**cholesky**	**gamma**	**dualize**	**pertv**	**sprint**	**dualp**	**psi**	**scal**	**passp**	**subs**	**perturb**	**presolve**	**spars**
barrier	{univ, dense}			{-208, -61,-50, 51,56, 61,102, 208}				{geo}	{83}	{132}			
default	native	off	3	50				auto	5	3	on	on	on

4.2 Experiments to evaluate scalability of the integer programming model

To evaluate the performance and the scalability of the proposed formulation in a standalone MIP solver, models for generating trees with different depths ($d = \{1, \ldots, 3\}$) with datasets of different sizes built by randomly selecting subsets of results of the complete experimental results of the COIN-OR CBC solver were solved with the state-of-the-art CPLEX 12.9 MIP solver on a computer with 32GB of RAM and 6 Intel®i7-4960X cores. In this experiment, we measured the final gap reported by the solver between the best lower and upper bound at the end of execution with one hour time limit. These experiments considered generating trees with a minimum number of 10 instances per leaf node ($\tau = 10$) and penalty of ($\beta = 50$) for leaf nodes violating this constraint. Fig. 8 shows the performance of our integer programming model, considering bases with 50 algorithms and problem instances ranging from 50 to 500.

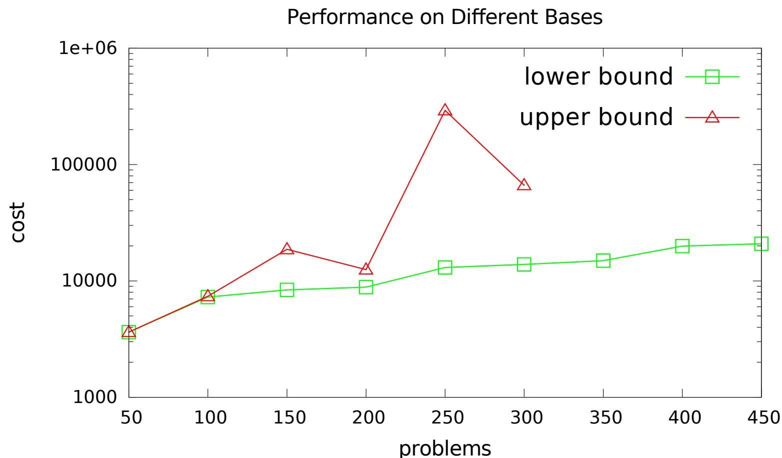

Figure 8: Performance of the integer programming model over sets of problem instances of different sizes and 50 algorithms.

As it can be seen, optimal or near optimal decision trees were generated for models with up to 200 problem instances. For larger datasets the execution terminated with increasingly larger gaps for the produced bounds, at the point that for models with more than 300 instances a feasible solution was not found in the time limit. For the model with 500 problem instances not even the LP relaxation of the MIP model was computed in the time limit and no lower bound was available. Thus, for the complete dataset, experiments in the next subsections were performed only with the proposed VND-ASP heuristic.

4.3 Experiments with the complete dataset

To create optimized decision trees considering the entire experimental results dataset is quite challenging: there are more than half a million observations[1], far beyond the limits indicated in the previous section. Thus, only experiments with our mathematical programming heuristics were conducted for this dataset.

Fig. 9 presents the decision tree constructed with VNS-ASP using the following parameters: maximum tree depth $d = 3$, total time limit $\hbar = 72000$, MIP search timeout $l = 4000$, elite set size $m = 20$, initial algorithms subsetsize $q = 100$, $q' = 20$, minimum number of instances per leaf node $\tau = 50$ and penalty cost $\beta = 500$. The estimated performance improvement with this decision tree is 61%, a remarkable improvement. Please note, however, that this improvement does not reflects the expected performance improvement of this tree in unknown instances, which is the really important estimate. The estimated results of the decision trees produced with our method on unknown instances is computed in the next section in 10-fold cross validation experiments.

An inspection in the contents of our decision tree shows that the range of the coefficients in the constraint matrix plays an important role for determining the best algorithm. The feature selected for the root node *aRatioLSA* is computed as the ratio between the largest and the smallest absolute non-zero values in the constraint matrix. Each leaf node has a set of instances allocated to it, depending on the the decision on all parent nodes and a recommended algorithm, which is the algorithm with better results on these instances. As an example, for the left-most branch of the tree, the best algorithm configuration select used the Primal simplex algorithm setting the "idiot" parameter to value 7 considering 127 LP problems allocated to this node.

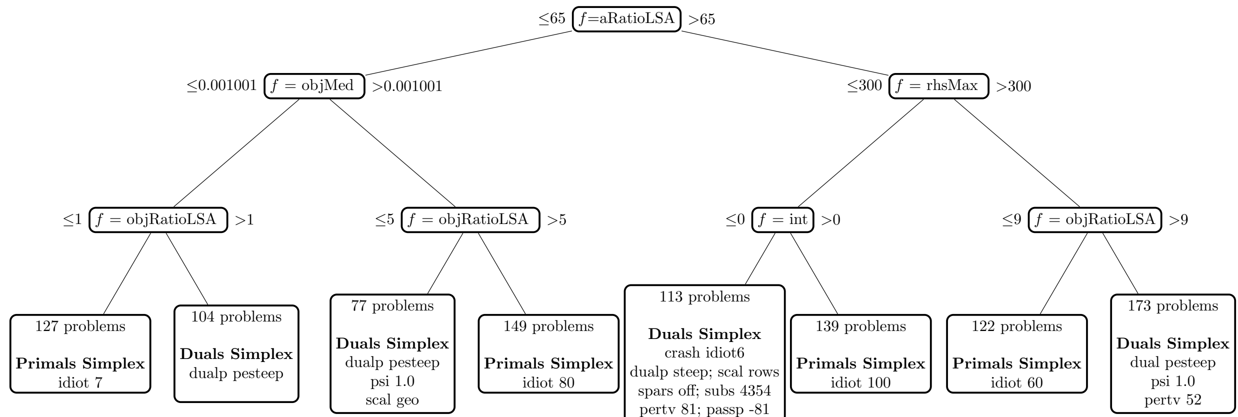

Figure 9: Decision tree with maximum depth = 3

4.4 Experiment using cross-validation on the complete base of problem instances

To evaluate the predictive power of our method, i.e. the expected performance on unknown instances, a 10-fold cross validation experiment was performed: a randomly shuffled complete dataset was divided into 10 partitions and at each iteration 9 of the partitions were used to create the decision tree (training dataset) and the remaining partition used for evaluating the decision tree (test dataset). Each partition had 480928 examples (904 problem instances × 532 available algorithms), with the exception of the last four partitions that contained 480396 examples (903 problem instances × 532 available algorithms). The results of the cross-validation are given in Fig. 10. This figure shows the average performance degradation considering the ideal performance to solve the LP relaxation of all problem instances (the lower bound). Results of VND-ASP with maximum tree depth 4 (VND-ASP(D=5)) and 5 (VND-ASP(D=5)) are included. The remaining parameters of VND-ASP are the same described in the previous subsection. We also compare

[1] 534128 execution results produced by solving 1004 LP problems with 532 different algorithm configurations each

our results with the results produced with different configurations of the Random Forest (RF) algorithm implemented in Weka (Hall et al., 2009) (RF(T=1,...,T=200, where T is the number of trees)). Default CBC settings (Default) and results obtained selecting a single best algorithm (SBA) are also included.

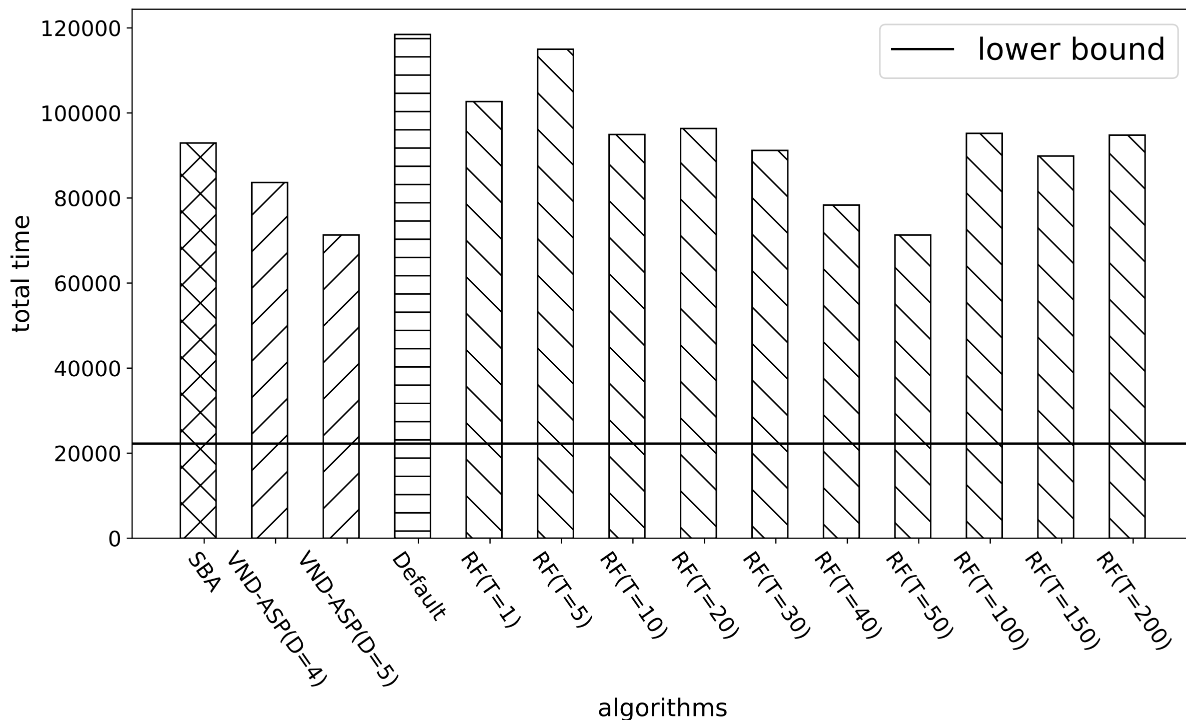

Figure 10: Cross-validation results for all partitions

As can be seen, our results indicate performance gains of 40% compared against default settings. Moreover, they are noticeably better than those obtained when selecting only the single best algorithm (22%). VND-ASP results are also mostly superior to the ones obtained with RF, with the exception of RF(T=50), where equivalent results were obtained. We believe that this is a very positive result, since the result produced by our algorithm (a single tree) is more easy to interpret than those produced by RF. More importantly, algorithms can be recommended much faster (in constant time) with our approach since the processing cost does not depends on the number of available algorithms as in RF, where a series of regression problems must be solved in order to recommend an algorithm for each new instance.

5 Discussion and closing remarks

This paper introduced a new mathematical programming formulation to solve the Algorithm Selection Problem (ASP). This formulation produces globally optimal decision trees with limited depth. The main advantage of this approach is that, despite the construction of the tree itself potentially being computationally expensive, once the tree has been constructed, algorithm recommendations can be made in constant time. A dataset containing the experimental results of many linear programming solver configurations of the COIN-OR Branch-&-Cut linear programming solver (CLP) was built solving a comprehensive set of instances from various applications. This initial batch of experiments itself already revealed improved parameter settings for the LP solver, including the discovery of a new algorithm configuration which was 22% faster than default CLP settings.

Scalability tests were performed to check how large subsets could become before it was no longer possible to generate provably optimal decision trees with a state of the art standalone MIP solver. Given that, at a certain point, the resulting MIP model becomes too difficult to optimize exactly, a mathematical programming-based VND local search heuristic was also proposed to handle larger datasets.

To evaluate the predictive power of our method, a 10-fold cross validation experiment was conducted. The results were very promising: executions with the recommended parameter settings were 40% faster than CLP default settings, almost doubling the improvement that could be obtained using a single best parameter setting. Our results are comparable with those obtained after tuning the Random Forest algorithm, with the added advantage that the predictive model produced by our method (a single tree) is easily interpretable and,

more importantly, the cost of recommending an algorithm is not dependent upon the number of available algorithms.

Future directions include evaluating stronger alternative integer programming formulations for this problem given that, as the scalability test showed, there is still a significant gap between the lower and upper bounds produced for the larger datasets. The positive results for the ASP are also a good indicator that the application of our methodology to classification and regression problems represents a promising future research path.

Acknowledgements

The authors would like to thank the Brazilian agencies CNPq and FAPEMIG for the financial support. The authors acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources of the SDumont supercomputer, which have contributed to the research results reported within this paper. URL: http://sdumont.lncc.br. The research was partially supported by Data-driven logistics (FWO-S007318N). Editorial consultation provided by Luke Connolly (KU Leuven).

References

Achard, P., De Schutter, E., 2006. Complex Parameter Landscape for a Complex Neuron Model. *PLOS Computational Biology* 2, 7, 1–11.

Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J., 2006. *The traveling salesman problem: a computational study.* Princeton university press.

Atamtürk, A., Savelsbergh, M.W., 2005. Integer-programming software systems. *Annals of operations research* 140, 1, 67–124.

Atkeson, C.G., Moore, A.W., Schaal, S., 1997. Locally Weighted Learning. *Artificial Intelligence Review* 11, 1-5, 11–73.

Audet, C., Orban, D., 2006. Finding Optimal Algorithmic Parameters Using Derivative-Free Optimization. *SIAM Journal on Optimization* 17, 3, 642–664.

Battistutta, M., Schaerf, A., Urli, T., 2017. Feature-based tuning of single-stage simulated annealing for examination timetabling. *Annals of Operations Research* 252, 2, 239–254.

Baz, M., Hunsaker, B., Brooks, J.P., Gosavi, A., 2007. Automated Tuning of Optimization Software Parameters. Technical report, Technical Report TR2007-7, University of Pittsburgh, Department of Industrial Engineering.

Bellio, R., Ceschia, S., Gaspero, L.D., Schaerf, A., Urli, T., 2016. Feature-Based Tuning of Simulated Annealing applied to the Curriculum-Based Course Timetabling Problem. *Computers & Operations Research* 65, 83 – 92.

Bertsimas, D., Dunn, J., 2017. Optimal Classification Trees. *Machine Learning* 106, 7, 1039–1082.

Bixby, R.E., Fenelon, M., Gu, Z., Rothberg, E., Wunderling, R., 2004. Mixed Integer Programming: A Progress Report. In Grötschel, M. (ed.), *The Sharpest Cut: The Impact of Manfred Padberg and His Work.* SIAM, chapter 18, pp. 309–324.

Bolme, D.S., Beveridge, J.R., Draper, B.A., Phillips, P.J., Lui, Y.M., 2011. Automatically Searching for Optimal Parameter Settings Using a Genetic Algorithm. In Crowley, J.L., Draper, B.A. and Thonnat, M. (eds), *Computer Vision Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 213–222.

Breiman, L., 2001. Random Forests. *Machine Learning* 45, 1, 5–32.

Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J., 1984. *Classification and Regression Trees.* Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A.

Eiben, A.E., Hinterding, R., Michalewicz, Z., 1999. Parameter Control in Evolutionary Algorithms. *Transactions on Evolutionary Computation* 3, 2, 124–141.

Fischetti, M., Fischetti, M., 2016. Matheuristics. *Handbook of Heuristics* pp. 1–33.

Fonseca, G.H., Santos, H.G., Carrano, E.G., Stidsen, T.J., 2017. Integer programming techniques for educational timetabling. *European Journal of Operational Research* 262, 28–39.

Forrest, J., Lougee-Heimer, R., 2005. CBC User Guide. In *Emerging Theory, Methods, and Applications*. INFORMS, pp. 257–277.

Gamrath, G., Koch, T., Martin, A., Miltenberger, M., Weninger, D., 2015. Progress in Presolving for Mixed Integer Programming. *Mathematical Programming Computation* 7, 367–398.

Garey, M.R., Johnson, D.S., 1979. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Gearhart, J.L., Adair, K.L., Detry, R.J., Durfee, J.D., Jones, K.A., Martin, N., 2013. Comparison of Open-Source Linear Programming Solvers. Technical report, Sandia National Laboratories.

Haas, J., Peysakhov, M., Mancoridis, S., 2005. GA-Based Parameter Tuning for Multi-Agent Systems. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, ACM, New York, NY, USA, pp. 1085–1086.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H., 2009. The weka data mining software: an update. *ACM SIGKDD explorations newsletter* 11, 1, 10–18.

Hutter, F., Stützle, T., Leyton-Brown, K., Hoos, H.H., 2014a. Paramils: An automatic algorithm configuration framework. *CoRR* abs/1401.3492. 1401.3492.

Hutter, F., Xu, L., H., H.H., Leyton-Brown, K., 2014b. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206, 79 – 111.

Hyafil, L., Rivest, R.L., 1976. Constructing Optimal Binary Decision Trees is NP-Complete. *Information Processing Letters* 5, 1, 15 – 17.

Johnson, E., Nemhauser, G., Savelsbergh, W., 2000. Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition. *INFORMS Journal on Computing* 12.

Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K., 2010. Isac –instance-specific algorithm configuration. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, IOS Press, Amsterdam, The Netherlands, The Netherlands, pp. 751–756.

Kass, G.V., 1980. An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 29, 2, 119–127.

Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R.E., Danna, E., Gamrath, G., Gleixner, A.M., Heinz, S., et al., 2011. MIPLIB 2010. *Mathematical Programming Computation* 3, 2, 103.

Kohavi, R., John, G.H., 1995. Automatic Parameter Selection by Minimizing Estimated Error. In *In Proceedings of the Twelfth International Conference on Machine Learning*, Morgan Kaufmann, pp. 304–312.

Land, A.H., Doig, A.G., 1960. An Automatic Method for Solving Discrete Programming Problems. *Econometrica* 28, 3, 497–520.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., Shoham, Y., 2003a. Boosting as a metaphor for algorithm design. In Rossi, F. (ed.), *Principles and Practice of Constraint Programming – CP 2003*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 899–903.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., Shoham, Y., 2003b. A portfolio approach to algorithm selection. In *IJCAI*, Vol. 3, pp. 1542–1543.

Loh, W.Y., Shih, Y.S., 1997. Split selection methods for classification trees. *Statistica Sinica* 7, 4, 815–840.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M., 2016. The irace Package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives* 3, 43–58.

López-Ibáñez, M., Stützle, T., 2014. Automatically Improving the Anytime Behaviour of Optimisation Algorithms. *European Journal of Operational Research* 235, 3, 569 – 582.

Lougee-Heimer, R., 2003. The Common Optimization Interface for Operations Research: Promoting Open-Source Software in the Operations Research Community. *IBM Journal of Research and Development* 47, 1, 57–66.

Mascia, F., Pellegrini, P., Birattari, M., Sttzle, T., 2014. An Analysis of Parameter Adaptation in Reactive Tabu Search. *International Transactions in Operational Research* 21, 1, 127–152.

Menickelly, M., Günlük, O., Kalagnanam, J., Scheinberg, K., 2016. Optimal Generalized Decision Trees via Integer Programming. *CoRR* abs/1612.03225.

Misir, M., Sebag, M., 2017. Alors: An algorithm recommender system. *Artificial Intelligence* 244, 291 – 314.

Mittelmann, H., 2018. Benchmark of simplex LP solvers. http://plato.asu.edu/ftp/lpsimp.html. Accessed: 2018-10-03.

Mladenović, N., Hansen, P., 1997. Variable Neighborhood Search. *Computers and Operations Research* 24, 11, 1097–1100.

Pochet, Y., Wolsey, L.A., 2006. *Production Planning by Mixed Integer Programming (Springer Series in Operations Research and Financial Engineering)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Polyakovskiy, S., Bonyadi, M.R., Wagner, M., Michalewicz, Z., Neumann, F., 2014. A comprehensive benchmark set and heuristics for the traveling thief problem. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ACM, pp. 477–484.

Quinlan, J.R., 1986. Induction of Decision Trees. *Machine Learning* 1, 1, 81–106.

Quinlan, J.R., 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Resende, M., Ribeiro, C., 2014. GRASP: Greedy randomized adaptive search procedures, Springer US. pp. 287–312.

Rice, J.R., 1976. The algorithm selection problem. *Advances in Computers*. Vol. 15. Elsevier, pp. 65 – 118.

Santos, H.G., Toffolo, T.A., Gomes, R.A., Ribas, S., 2016. Integer programming techniques for the nurse rostering problem. *Annals of Operations Research* 239, 225–251.

Silva, C., Santos, H., 2017. Drawing graphs with mathematical programming and variable neighborhood search. *Electronic Notes in Discrete Mathematics* 58, 207 – 214.

Song, Y.Y., Lu, Y., 2015. Decision tree methods: applications for classification and prediction, 27, 2, 130–135.

Souza, M., Coelho, I., Ribas, S., Santos, H., Merschmann, L., 2010. A hybrid heuristic algorithm for the open-pit-mining operational planning problem. *European Journal of Operational Research* 207, 2, 1041 – 1051.

Tange, O., 2011. Gnu parallel-the command-line power tool. *The USENIX Magazine* 36, 1, 42–47.

Tsoumakas, G., Katakis, I., 2007. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)* 3, 3, 1–13.

Vapnik, V.N., 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

Vilas Boas, M.G., Santos, H.G., Martins, R.S.O., Merschmann, L.H.C., 2017. Data Mining Approach for Feature Based Parameter Tunning for Mixed-Integer Programming Solvers. *Procedia Computer Science* 108, 715 – 724.

Witten, I.H., Frank, E., Hall, M.A., 2011. *Data Mining: Practical Machine Learning Tools and Techniques* (3rd edn.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Wolsey, L.A., 2007. Mixed Integer Programming, John Wiley & Sons, Inc.

Xu, L., Hoos, H., Leyton-Brown, K., 2010. Hydra: Automatically configuring algorithms for portfolio-based selection.

Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K., 2008. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research* 32, 565–606.

Zhang, H., Wang, S., Xu, X., Chow, T.W., Wu, Q.J., 2018. Tree2vector: learning a vectorial representation for tree-structured data. *IEEE transactions on neural networks and learning systems* , 99, 1–15.

Zhu, Y., 2007. Mixed-Integer Linear Programming Algorithm for a Computational Protein Design Problem. *Industrial & Engineering Chemistry Research* 46, 3, 839–845.