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

Matheus Guedes Vilas Boas
Federal University of Ouro Preto

PhD thesis submitted to the Institute of Exact and Biological Sciences of the Federal University of Ouro Preto to obtain the title of PhD in Computer Science

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

V697d Vilas Boas, Matheus Guedes.
Decision trees for the algorithm selection problem [manuscrito]: integer programming based approaches. / Matheus Guedes Vilas Boas. - 2019.
70 f.: il.: color., tab..

Orientador: Prof. Dr. Haroldo Gambini Santos.
Tese (Doutorado). Universidade Federal de Ouro Preto. Departamento de Computação. Programa de Ciência da Computação.
Área de Concentração: Ciência da Computação.

1. Algoritmos de computador. 2. Mineração de dados (Computação). 3. Programação inteira. I. Santos, Haroldo Gambini. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.62

Bibliotecário(a) Responsável: Celina Brasil Luiz - CRB6 - 1589

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UFOP

ATA DE DEFESA DE DOUTORADO

Aos 13 dias do mês de dezembro do ano de 2019, às 09:00 horas, nas dependências do Departamento de Computação (Decom), foi instalada a sessão pública para a defesa de tese do doutorando Matheus Guedes Vilas Boas, sendo a banca examinadora composta pelo Prof. Dr. Haroldo Gambini Santos (Presidente - UFOP), pelo Prof. Dr. Christian Clemens Blum (Membro - Externo), pelo Prof. Dr. Luiz Henrique de Campos Merschmann (Membro - Externo), pelo Prof. Dr. Rodrigo Cesar Pedrosa Silva (Membro - UFOP), pelo Prof. Dr. Tulio Angelo Machado Toffolo (Membro - UFOP). Dando início aos trabalhos, o presidente, com base no regulamento do curso e nas normas que regem as sessões de defesa de tese, concedeu ao doutorando 60 minutos para apresentação do seu trabalho intitulado "Optimal Decision Trees for the Algorithm Selection Problem: Integer Programming Based Approaches". Terminada a exposição, o presidente da banca examinadora concedeu, a cada membro, um tempo máximo de 30 minutos para perguntas e respostas ao candidato sobre o conteúdo da tese, na seguinte ordem: Primeiro Prof. Dr. Christian Clemens Blum; segundo Prof. Dr. Luiz Henrique de Campos Merschmann; terceiro Prof. Dr. Rodrigo Cesar Pedrosa Silva; quarto Prof. Dr. Tulio Angelo Machado Toffolo; quinto Prof. Dr. Haroldo Gambini Santos. Dando continuidade, ainda de acordo com as normas que regem a sessão, o presidente solicitou aos presentes que se retirassem do recinto para que a banca examinadora procedesse à análise e decisão, anunciando, a seguir, publicamente, que o doutorando foi aprovado, sob a condição de que a versão definitiva da tese deva incorporar todas as exigências da banca, devendo o exemplar final ser entregue no prazo máximo de 60 (sessenta) dias à Coordenação do Programa. Para constar, foi lavrada a presente ata que, após aprovada, vai assinada pelos membros da banca examinadora e pelo doutorando. Ouro Preto, 13 de dezembro de 2019.

Haroldo G. Santos
____
Prof. Dr. Haroldo Gambini Santos

Presidente

XXXXXXXXXXXXXXXXXXXXXXXXXX
____
Prof. Dr. Christian Clemens Blum (Participação por Videoconferência)

[illegible]
____
Prof. Dr. Luiz Henrique de Campos Merschmann

Rodrigo Silva
____
Prof. Dr. Rodrigo Cesar Pedrosa Silva

[illegible]
____
Prof. Dr. Tulio Angelo Machado Toffolo

Matheus Guedes Vilas Boas
____
Doutorando

Certifico que a defesa realizou-se com a participação a distância do(s) membros(s) Prof. Dr. Christian Clemens Blum e que, depois das arguições e deliberações realizadas, cada participante a distância afirmou estar de acordo com o conteúdo do parecer da banca examinadora, redigido nesta ata.

Haroldo G. Santos
____
Prof. Dr. Haroldo Gambini Santos

Presidente

I dedicate to God, my son, my parents, my brother and my dogs.

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

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 thesis, we propose Integer Programming based approaches to build decision trees for the Algorithm Selection Problem. These techniques allow the automation of 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. We tested our approach from different perspectives: (*i*) univariate approach, where for each branch node, only one cutoff point of a feature is chosen and (*ii*) multivariate approach, where for each branch node, weights for multiple features are used (oblique decision trees). Considering the current scenario where the number of cores per machine has increased considerably, we also propose a new approach based on recommendation of concurrent algorithms. To evaluate our approaches, extensive computational experiments were executed using a dataset that considers the linear programming algorithms implemented in the COIN-OR Branch & Cut solver across a comprehensive set of instances, including all MIPLIB benchmark instances. We also conducted experiments with the scenarios/-datasets of the Open Algorithm Selection Challenge (OASC) held in 2017. Considering the first dataset and a 10-fold cross validation experiment, while selecting the single best solver across all instances decreased the total running time by 2%, our univariate approach decreased the total running time by 68% and using the multivariate approach, the total running time is decreased by 72%. An even greater performance gain can be obtained using concurrent algorithms, something not yet explored in the literature. For our experiments, using three algorithm configurations per leaf node, the total running time is decreased by 85%. These results indicate that our method generalizes quite well and does not overfit. Considering the results obtained using the scenarios of the OASC, the experimental results showed that our decision trees can produce better results than less interpretable models, such as random forest, which has been extensively used for algorithm recommendation.

Keywords: Algorithm Selection Problem, Feature Based Parameter Tuning, Data Mining, Integer Programming, Variable Neighborhood Search, COIN-OR Branch & Cut, Oblique Decision Trees, Recommendation of Concurrent Algorithms.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or processional qualification except as specified.

Matheus Guedes Vilas Boas

Acknowledgements

I thank God for always giving me strength to not give up and mainly, for giving me the pleasure of living!

I am grateful to Professor Haroldo Gambini Santos, adviser of the work, for dedication, commitment and above all for the trust placed in me.

I thank all the other teachers who have contributed to my education, always with pleasure and dedication to teaching.

I thank you for my son Pedro Henrique, everything for me.

I thank my mother Inês Alves Guedes and my father Luiz Carlos Vilas Boas for the love, care, and support for the teachings given to me.

I thank my brother Lucas Guedes Vilas Boas, for all the help offered in this work and around the doctorate. In addition to help, thank you for always believe in my success.

I thank my dogs Joe and Princesa, companions for life.

I thank everyone who contributed in any way that I could achieve this goal of great importance in my life.

Thank you all!

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

ASP	Algorithm Selection Problem
CART	Classification and Regression Trees
CBC	COIN-OR Branch & Cut
CLP	COIN-OR Linear Programming Solver
IP	Integer Programming
GRC	Greedy Randomized Constructive
LP	Linear Programming
MIP	Mixed-Integer Programming
MIPLIB	Mixed Integer Programming Library
VND	Variable Neighborhood Descent

Chapter 1

Introduction

In this thesis, we study the Algorithm Selection Problem (ASP). This problem is concerned with selecting the best algorithm to solve a given problem instance on a case-by-case basis (RICE, 1976; KOTTHOFF, 2012; BISCHL et al., 2016).

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. 1.1.

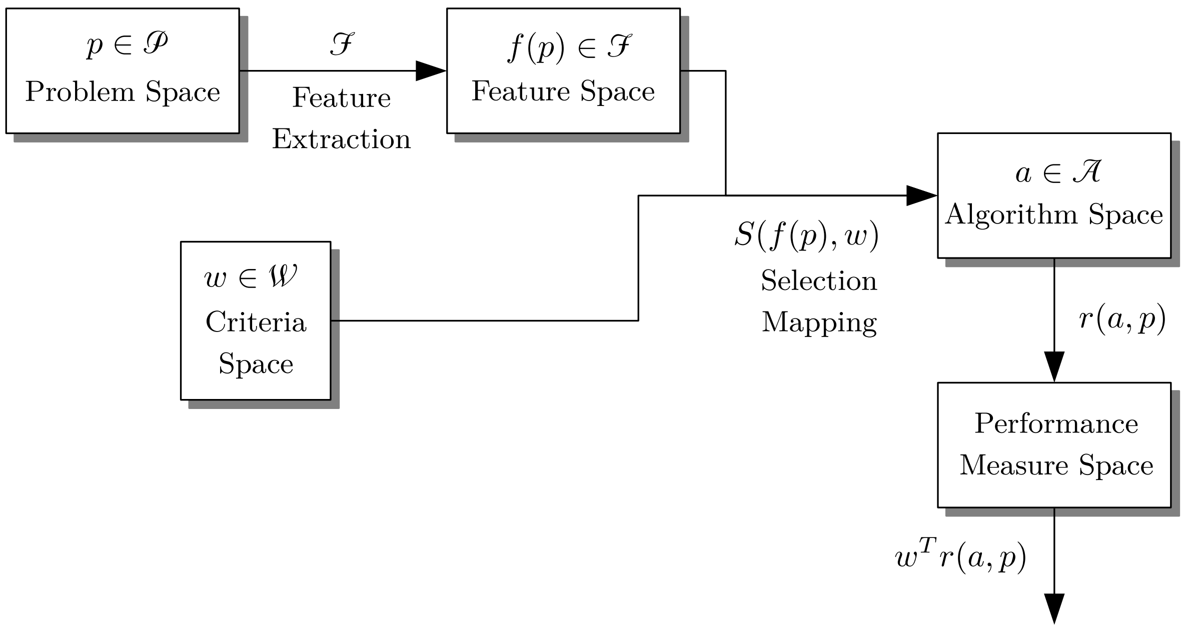


Figure 1.1: The Algorithm Selection Problem (RICE, 1976)

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 setting 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;

$\mathscr{W}$: 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 $w \in \mathscr{W} = [0, 1]^n$ 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:

$$\sum_{p \in \mathscr{P}} |w^T r(B(f(p), w)) - w^T r(S(f(p), w))| \tag{1.1}$$

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 thesis 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. Recent exceptions include 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. Also, the works of King et al. (2000) and Michie et al. (1995) used decision trees for recommending algorithms. In the training phase, the system marks each algorithm as applicable or not, depending on its similarity with the best algorithm. In the test phase, the decision tree is used to predict whether or not the algorithm is applicable to 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.

We tested our approach from two perspectives: (i) univariate approach, where for each branch node, only one cutoff point of a feature is chosen; (ii) Multivariate approach, where for each branch node, weights for multiple features are used (oblique decision trees). Considering the current scenario where the number of cores per machine has increased considerably, we also propose a new approach based on recommendation of concurrent algorithms. Thus, for each leaf node, we tested the recommendation of 1, 2 and 3 algorithm configurations.

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 (1.1). 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., 2003; 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.

As discussed in Kotthoff (2012), if an incorrect algorithm is chosen and our performance measure is processing time, the system can take more time to solve the problem instance. This additional time will vary depending on which algorithm was chosen — it may not matter much if the performance of the chosen algorithm is very close to the best algorithm, but it also may mean a large difference.

The present thesis therefore investigates the applicability of Integer Programming to build a mapping S with decision trees using the precise objective function (1.1) 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., 2014) 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 thesis we consider the more computationally expensive case, where for the training dataset all problem instances were evaluated on all algorithms.

This thesis 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 one of 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 (Subsection 6.1.5) but it also has the distinct advantages of recommending algorithms in constant time and producing easily interpretable results.

1.1 Text organization

The remainder of the thesis is organized as follows: Chapter 2 is dedicated to literature review. Chapter 3 presents the Integer Programming formulation for the construction of optimal decision trees for the ASP. Chapter 4 presents the model proposed for the multivariate choice of features to exactly one node of a decision tree. Chapter 5 presents a variable neighborhood descent based mathematical programming heuristic. Our extensive computational experiments and their results are presented in Chapter 6 and, finally, Chapter 7 presents the conclusions and provides some future research directions.

Chapter 2

Background

2.1 Algorithm Selection Problem

In the area of artificial intelligence, much time has been devoted to developing new algorithms that outperform previous approaches in solving a class of problem instances. The problem with this approach is that it surpasses the state of the art only in a specific class of problem instances. Improved algorithms for subclasses of problems may be developed since specific assumptions about the instance environment can be considered and these assumptions may not be valid for the all possible instances (KOTTHOFF, 2012). Considering a set of available algorithms, the Algorithm Portfolio, our objective is to recommend the best algorithm for new instance considering its characteristics and the experience acquired while solving other instances.

Researchers have long ago recognized that a single algorithm will not give the best performance across all problem instances one may want to solve (KOTTHOFF, 2012). This can be checked in the works of Aha (1992) and Wolpert and Macready (1997).

Some concepts are very important in this problem and described by Kerschke et al. (2019): an *oracle selector* or *virtual best solver* (VBS) is defined as a hypothetical algorithm selector with perfect performance for all problem instances. This VBS provides a lower bound on the performance of any realistic algorithm. In turn, the *single best solver* (SBS) is one algorithm with the best performance on average among all the available algorithms.

In the work of Salisu et al. (2017), the authors present different techniques developed to solve the Algorithm Selection Problem. The first work presented is StatLog (KING et al., 2000; MICHIE et al., 1995), where 19 data characteristics and 10 algorithms were used. In the training phase, the system marked the algorithms as applicable or not, based on similarity when compared to the best algorithm. A decision tree was built to predict whether or not the algorithm was applicable to a new data set. Subsequently, a tool called Data Mining Advisor (DMA) (GIRAUD-CARRIER, 2006) was proposed, where a ranking was provided for the classification algorithms. For this, the K-Nearest Neighbor (k-NN) algorithm was used to predict the performance of the algorithms in a new dataset.

The IDEA - Intelligent Discovery Electronic Assistant (BERNSTEIN et al., 2005) system is the first data-based planning analysis for data mining capable of building workflows. In turn, the e-Lico Intelligent Discovery Assistant (eIDA)[1] system creates data mining processes based on input data specification and user objectives. The Auto-WEKA tool (KOTTHOFF et al., 2017; THORNTON et al., 2012) is designed to help users automatically search through the learning algorithm set and hyper-parameter settings to maximize a certain measure of performance, such as instance accuracy. For this, the Bayesian optimization method is used. Finally, the Auto-sklearn (FEURER et al., 2015) system provides an automated machine learning toolkit, using recent improvements in Bayesian optimization, meta-learning, and ensemble construction.

Many of the current approaches to solve ASP use machine learning. In this context, a model may be constructed where instances are grouped according to their characteristics, so that similar instances belong to the same class and different instances belong to different classes. This model is used to predict performance on unknown instances. Machine learning applied to ASP involves a training phase, where the candidate algorithms are run on a sample of the problem space to experimentally evaluate their performance. This training data is used to create a performance model that can be used to predict the performance on new, unseen instances (KOTTHOFF, 2012).

2.1.1 Algorithm Selector And Pre-scheduler

Algorithm Selector and Pre-scheduler (ASAP) combines Algorithm Selection Problem (ASP) with Pre-scheduler. A scheduler is defined sequentially launching a few algorithms on a limited computational budget each. The ASAP system relies on the joint

[1]http://www.e-lico.eu/

optimization of a pre-scheduler and a per instance AS (Algorithm Selection) (GONARD et al. (2016)). Since some instances can be easily resolved by some algorithms, the pre-scheduler can allocate a portion of the computational budget to solve these instances. The pre-scheduler can be used to add new features to characterize the problem in question. These features are incorporated into the problem instances, indicating whether such an algorithm solves the instance. After this first phase, the second phase deals only with instances that have not yet been solve. As Xu et al. (2012), ASP and Pre-scheduler can be combined in a multistage process, where the scheduler resolves easy instances and the other instances are handled by ASP. In ASAP, we have two decisions to make: the first concerns how to split the available runtime between Pre-scheduler and ASP. The second decision defines how many algorithms (parameter k) will be involved in the pre-scheduler. Two versions of the ASAP system were considered: **ASAP.v1** and **ASAP.v2**, which will be discussed in the next paragraphs.

In ASAP.v1, the maximum execution time allocated to the pre-scheduler is fixed at 10% of the overall computational budget and the number of algorithms in the pre-scheduler is defined as 3 ($k = 3$). It is easy to see that the two phases are interdependent: the AS should focus on the instances that the pre-scheduler was unable to resolve while the pre-scheduler should focus on the instances where the AS does not behave well - select an inappropriate algorithm.

In the first phase (pre-scheduler), a mixed optimization problem is solved in order to balance the overall number of problems solved and the overall computational budget. As the experiments are carried out in scenarios with a maximum of 31 algorithms, the value of the parameter k and the maximum time allocated to this phase is determined by exhaustive search.

In the second phase (algorithm selector), two learning algorithms are considered: random forests and k-nearest neighbors. Based on preliminary experiments, 35 trees were used in the Random Forest algorithm and the number of neighbors defined as $k = 3$ for the k-nearest neighbors. In the k-nearest neighbors algorithm, the associated predicted value each instance of the problem is defined as the weighted sum of the performance of its closest neighbors, weighted by the relative distance to the instance. The features were normalized before selecting neighbors.

In ASAP.v2, the two phases are not considered sequentially. In other words, there is a loop for retrofitting the pre-scheduler to the new AS.

2.1.2 Algorithm Recommender System

In the work of Misir and Sebag (2017), the authors propose a collaborative filtering approach to the Algorithm Selection Problem which they give the name Algorithm Recommender System - ALORS. Such an approach is inspired by the challenge of Netflix: recommending items you will like based on previous items you like and that other users like. Two advantages of collaborative filtering (CF) over the algorithm selection problem are: ASP requires experimental results of all algorithms in all instances of the training set. This is not the case in CF, where only a portion of the experimental results may be available. The second advantage is related to the performance model: in CF, latent factors are created to describe algorithms and instances, and use the associated latent metrics to recommend algorithms for a particular instance. Still on the second advantage, the authors state that CF allows to independently analyze the following points: the representativeness of the set of problem instances in relation to the algorithm portfolio and; the quality of features used to describe problem instances.

2.1.3 Random Forest

Random forests consist of a set of decision trees. In work of Hutter et al. (2014), the authors use random forests to solve regression problems to predict algorithm runtimes. The prediction of algorithm runtimes allows the development of algorithm recommendation systems, since one can recommend the algorithm with the lowest expected runtime. Regression trees are extraordinarily flexible predictors, capable of capturing very complex and low bias interactions. Trees are designed to be different: Training takes place on different sub-samples of training data and/or allowing only a random subset of the variables as branch variables on each node. The average predictions for a new entry x are easy to calculate. For each tree, the answer to x is predicted and then the average of the predictions is calculated.

Still on this paper, the authors proposed new techniques for building predictive models and advanced state of the art in predicting the performance of algorithms for difficult combinatorial problems - namely SAT, MIP and TSP. Computational experiments were performed, predicting the performance of 11 algorithms in 35 instances of SAT, MIP and TSP, where the techniques proposed by the authors were compared with a set of literature methods. The results demonstrate the superiority of random forest based approaches when compared to other methods such as Ridge regression wih 2-phase forward selection; SPORE-FoBa (ridge regression with forward-backward selection); Feed-forward neural network with one hidden layer; Projected process (approximate Gaussian process) and; Regression tree with cost-complexity pruning. In this sense, we use Random Forest to compare performance with our approach to resolving ASP.

2.2 Decision Trees

Decision trees are commonly used to solve classification and regression problems. In this sense, the Subsections 2.2.1 and 2.2.2 discuss advances in regression tree algorithms and classification tree algorithms.

A classic example of classification using decision trees considers the Iris flower dataset. The dataset has 4 features: sepal length (SepalLength); sepal width (SepalWidth); petal length (PetalLength) and petal width (PetalWidth). Figure 2.1 presents the features of the Iris flower.

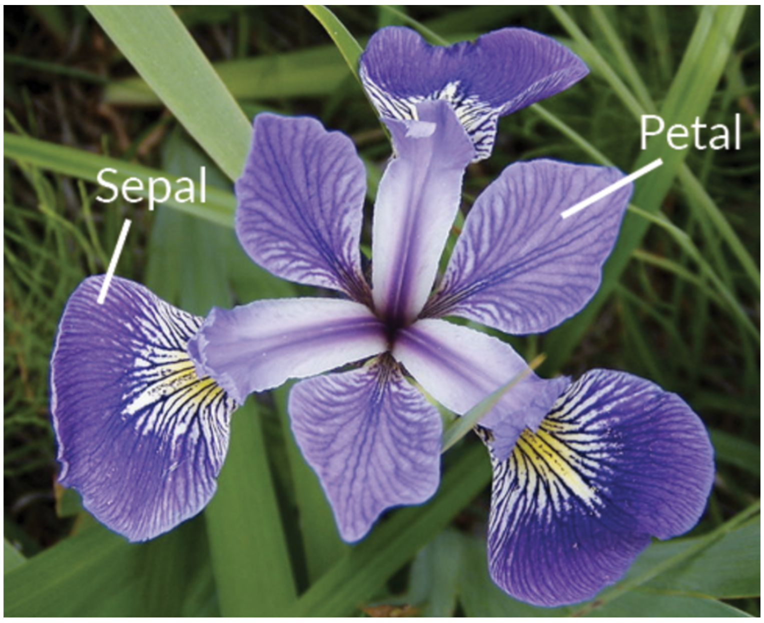


Figure 2.1: Features of the Iris flower. Source: `http://rafaelsakurai.github.io/classificacao-iris/`

The decision tree is then built, using the features to classify each flower/example into one of three possible classes: Versicolor, Setosa and Virginica. Figure 2.2 shows the three types/classes of the Iris flower.

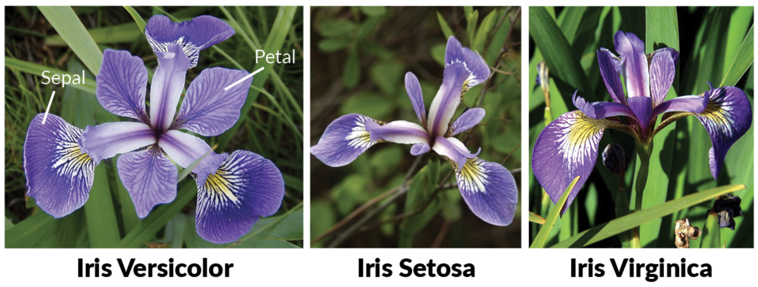


Figure 2.2: Types/Classes of of the Iris flower. Source:https://www.datacamp.com/community/tutorials/machine-learning-in-r

An example of a decision tree considering the Iris flower dataset is shown in Figure 2.3.

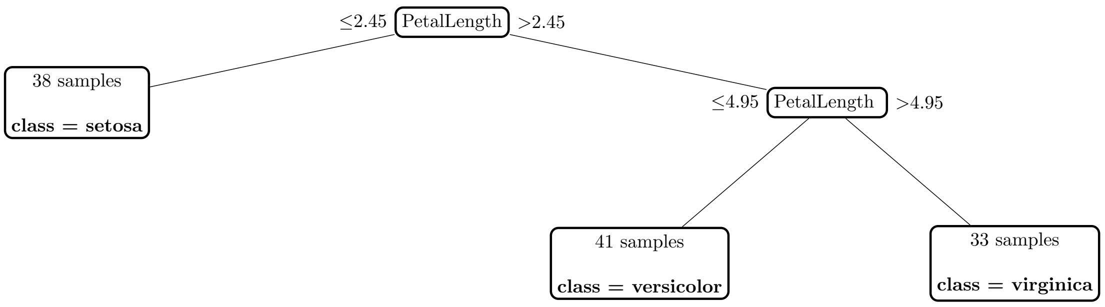


Figure 2.3: Decision Tree - Iris flower dataset.

Considering a new example/flower, we want to classify it in one of the 3 possible classes. For this, we verified that in the root node of the tree, the PetalLength feature was chosen with the cutoff point value equal to 2.45. In this sense, if the value of the PetalLength feature in the new example is less than or equal to 2.45, we should move to the left side of the tree. Otherwise, we go to the right side of the tree. Moving to the left side of the tree, we have already reached the leaf node, which gives the new example the setosa class. Moving to the right side of the tree, we have to re-evaluate the PetalLength feature, but now with a new cutoff point value (= 4.95). Thus, if the value of the PetalLength feature in the new example is less than or equal to 4.95, we should move to the left side of the tree (class = versicolor). Otherwise, we should move to the right side of the tree (class = virginica).

In Menickelly et al. (2016), a new formulation of mixed integer programming is proposed for constructing optimal binary classification trees of specific size. The input data are all binary and are considered small decision trees with up to three decision levels. A special structure of categorical characteristics were considered and thus combinatory decisions (based on subsets of values of such characteristic) were allowed in each node.

The computational results show that the small trees constructed have much better accuracy than the popular heuristic methods for building decision trees. The execution times were not compared, due to the considerable differences in the resolution strategy of the previously described methods. It can be seen that the classifiers *CART* and *Random Forest* overestimate all datasets used in the experiments. It is suggested as extensions of the work the treatment of characteristics of real values and the change in the objective of the resolution of the MIP model, considering the maximization of sensitivity or specificity. The work of Bertsimas and Dunn (2017) reinforces the conclusions reached in the work of Menickelly et al. (2016). Experiments with synthetic data were performed on the method proposed by Bertsimas and Dunn (2017) and when compared to the results obtained by the state of the art methods, the results are notably better. This contradicts the popular belief that such optimal methods will just overfit the training data at the expense of generalization ability.

2.2.1 Regression Problems

The first algorithm for regression trees was published by Morgan and Sonquist (1963) and named Automatic Interation Detection (AID). From the root node, this algorithm chooses the division that minimizes the sum of impurities on the two child nodes. Division is terminated when the reduction in impurity is less than an a determined fraction of impurity in the root node. Considering each node and the sample mean of the training dataset belonging to this node, impurity considers the sum of squared deviations. Studies in the literature showed that AID had serious problems: the first problem was overestimating the data, as proved by Einhorn (1972). In work of Doyle (1973), the author showed that if two or more variables are highly correlated, at most one of them can appear in the tree structure. This may lead to a hasty conclusion: the variable that did not appear in the tree structure may be considered unimportant.

Classification and Regression Trees (CART) were proposed by Breiman et al. (1984). Several improvements to the AID algorithms were considered. One of the improvements solved the problem of overstimating the data - as noted by Einhorn (1972): there are no rules to stop splitting a node into two child nodes. Instead, the tree grows and is then pruned to a size that has the lowest cross-validation estimate of error.

2.2.2 Classification Problems

The first algorithm proposed for classification is called THeta Automatica Interaction Detection (THAID) (MESSENGER and MANDELL, 1972). The algorithm is similar to the proposed AID algorithm for regression. The difference is in the prediction variable: in the THAID algorithm we consider the categorical variable and then the objective of the algorithm is chooses splits to maximize the sum of the number of correct observations in each modal category. The CART algorithm, already presented in the previous subsection, is also used in the construction of classification trees. CART (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets.

The CART algorithm has the following steps:

1. The algorithm starts at the root node of the tree and performs a division, creating two child nodes at the next level of the tree.
2. After splitting, the same procedure done in Step 1 is done for the two child nodes generated.
3. Divisions are made successively at the following levels.
4. The algorithm grows a tall tree and prunes some of its branches at the end of the process.

The division to be carried out is based on that which produces the greatest decrease in the classification error. The divisions are made until the final subsets are homogeneous in terms of the result variable (class). As a pruning criterion, all nodes below the lowest permitted level of the tree are removed. The decision class is the class with the highest number of cases in the respective node. Other criteria used for pruning are: definition of the minimum number of observations and the imposition of a minimum rate of diversity.

The algorithm called CHi-squared Automatic Interaction Detector (CHAID) (KASS, 1980) considers three types of variables: categorical, ordered without missing values (monotonic variables) and ordered with missing values (floating variables). The node is divided into two or more child nodes, depending on the type of the variable. Pairs of children nodes are then considered for merging by using Bonferroni-adjusted significance tests. The merged children nodes are then considered for division, again by means of Bonferroni-adjusted tests.

C4.5 is an algorithm proposed by Quinlan (1993). The algorithm assumes that if X has m distinct values in one node, then the node will be divided into m child nodes, with one child node for each value. The algorithm uses a node impurity-based entropy metric called the gain ratio. While the C4.5 algorithm accepts continuous and categorical data types, the CART algorithm accepts continuous data types and nominal attributes. C4.5 performs a pruning, while CART prunes the tree after processing the algorithm. Regarding missing values, only the C4.5 algorithm can handle this issue.

Classification Rule with Unbiased Interaction Selection and Estimation (CRUISE) (KIM and LOH, 2001) splits each node into multiple child nodes, this number being determined by the number of classes. For variable selection, CRUISE uses contingency table chi-squared tests for variable selection throughout, with the values of Y forming the rows and the (grouped, if X is ordered) values of X forming the columns of each table.

More recent works use a set of classifiers for predictions. In this sense, from the predicted values, the one with the largest number of votes is chosen, that is, the value that was predicted by the most classifiers. Among these works, Bagging (BREIMAN, 1996) uses an ensemble of unpruned CART trees constructed from bootstrap samples of the data. Random forest (BREIMAN, 2001) weakens the dependence among the CART trees by using a random subset of X variables for split selection at each node of a tree. In the work of Freund and Schapire (1997), the authors propose the Boosting algorithm. In it, the set of classifiers is constructed sequentially, so that for the incorrectly classified observations, greater weight is given in the next iterations.

2.2.3 Oblique Decision Trees

Splits considering the choice of only one feature and cut point may not work well for some problems (BROWN et al., 1996). For example, in the parity function problem (the n-dimensional generalization of the XOR function) the use of univariate approaches does not produce good results. Such a problem involves pattern recognition where the patterns associated with each class can come from separate regions in the feature space. In this sense, the authors of the work mentioned above presented a new approach to perform multi-variable splits within the recursive partitioning algorithm based on linear programming. Two objectives were considered in the proposed approach: the formulation that minimizes the maximum deviation of the input observations from the decision surface and the formulation that minimizes the sum of the deviations. The first objective considers, for each class, to minimize the number of input observations classified incorrectly. The second objective considers solving an LP for all classes, where the objective is to minimize the sum of the deviations (input observations classified incorrectly).

The new approach uses linear programming (LP) to find the locally optimal multivariate splits for classification problems. On each node of the tree is solved the optimal linear discriminant that separates one of the classes from all of the other classes. The solutions for each class are stored in addition to the univariate solution. The best solution of this set is chosen.

As described in Murthy et al. (1994), the approach described above (named as CART-LC) has severe limitations: CART-LC is deterministic. In this sense, there is no mechanism for escaping local optimum; CART-LC produces only a single tree; some adjustments to escape from some local optimum imply increased impurity of a split and there is no upper bound on the time spent at any node in the decision tree.

In work of Murthy et al. (1994), the system Oblique Classifier 1 (OC1) is proposed for the construction of oblique decision trees. The system combines deterministic hill-climbing with two forms of randomization to find a good oblique split (in the form of a hyperplane) at each node of a decision tree. The use of randomization improves the CART algorithm, originally proposed by Breiman et al. (1984), without significantly increasing the computational cost of the algorithm.

Chapter 3

Model

This chapter presents our integer programming model proposed for the construction of optimal decision trees for the ASP. The sets and parameters are described in Section 3.1. The corresponding decision variables are described in Section 3.2. The objective function associated with the problem and the constraints are described in Section 3.3.

3.1 Input data

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

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

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

$\boldsymbol{\mathscr{C}_f}$ 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;

$\boldsymbol{d}$ maximum tree depth;

$\boldsymbol{\tau}$ threshold indicating a minimum number of instances per leaf node;

$\boldsymbol{\beta}$ penalty incorporated into the objective function when a leaf node contains a number of problem instances smaller than threshold τ;

$\boldsymbol{r_{p,a}}$ cost of algorithm a for solving problem instance p;

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

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

$\boldsymbol{h_{l,n}}$ 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, $h_{l,n} = -1$. The id/number of a node is set from 1 to the number of nodes at a certain level in the tree. Thus, the second level of a tree has nodes with id = 1 ($n = 1$) and id = 2 ($n = 2$).

3.2 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.

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

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

3.3 Objective function and constraints

The objective of our model is to construct a tree of pre-defined maximum depth that minimizes the distance of the performance (cost) obtained using the recommended algorithm from the ideal performance (cost) 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. The objective function (3.1) and the set of constraints (3.2-3.17) of our model are presented below:

$$min \sum_{n=1}^{2^d} \sum_{p=1}^{\mathscr{P}} \sum_{a=1}^{\mathscr{A}} r_{p,a} \times w_{p,n,a} + \sum_{n=1}^{2^d} \beta \times m_n \tag{3.1}$$

subject to

$$\sum_{f \in \mathcal{F}} \sum_{c \in \mathcal{C}_f} x_{l,n,f,c} = 1 \qquad \forall\ l \in \{0, \dots, (d-1)\},\ n \in \{1, \dots, 2^l\} \quad (3.2)$$

$$\sum_{n=1}^{2^d} \sum_{a=1}^{\mathcal{A}} w_{p,n,a} = 1 \qquad \forall\ p \in \mathcal{P} \quad (3.3)$$

$$\sum_{a \in \mathcal{A}} z_{n,a} = 1 \qquad \forall\ n \in \{1, \dots, 2^d\} \quad (3.4)$$

$$w_{p,n,a} \leq z_{n,a} \qquad \forall\ p \in \mathcal{P},\ n \in \{1, \dots, 2^d\},\ a \in \mathcal{A} \quad (3.5)$$

$$w_{p,n,a} \leq y_{d,n,p} \qquad \forall\ p \in \mathcal{P},\ n \in \{1, \dots, 2^d\},\ a \in \mathcal{A} \quad (3.6)$$

$$u_n \geq y_{d,n,p} \qquad \forall\ n \in \{1, \dots, 2^d\},\ p \in \mathcal{P} \quad (3.7)$$

$$\sum_{p \in \mathcal{P}} y_{d,n,p} + m_n \geq \tau \times u_n \qquad \forall\ n \in \{1, \dots, 2^d\} \quad (3.8)$$

$$y_{l,n,p} \leq y_{(l-1),max(g_{l,n},h_{l,n}),p} \qquad \forall\ l \in \{2, \dots, d\},\ n \in \{1, \dots, 2^l\},\ p \in \mathcal{P} \quad (3.9)$$

$$y_{l,n,p} \leq 1 - x_{(l-1),g_{l,n},f,c} \qquad \forall\ l \in \{1, \dots, d\},\ n \in \{1, \dots, 2^l\}, \quad (3.10)$$
$$p \in \mathcal{P},\ f \in \mathcal{F},\ c \in \mathcal{C}_f : g_{l,n} \neq -1 \wedge v_{p,f} \leq c$$

$$y_{l,n,p} \leq 1 - x_{(l-1),h_{l,n},f,c} \qquad \forall\ l \in \{1, \dots, d\}, n \in \{1, \dots, 2^l\}, \quad (3.11)$$
$$p \in \mathcal{P},\ f \in \mathcal{F},\ c \in \mathcal{C}_f : h_{l,n} \neq -1 \wedge v_{p,f} > c$$

$$x_{l,n,f,c} \in \{0,1\} \qquad \forall\ l \in \{0, \dots, (d-1)\},\ n \in \{1, \dots, 2^l\},\ f \in \mathcal{F},\ c \in \mathcal{C} \quad (3.12)$$

$$y_{l,n,p} \in \{0,1\} \qquad \forall\ l \in \{1, \dots, d\},\ n \in \{1, \dots, 2^l\},\ p \in \mathcal{P} \quad (3.13)$$

$$z_{n,a} \in \{0,1\} \qquad \forall\ n \in \{1, \dots, 2^d\},\ a \in \mathcal{A} \quad (3.14)$$

$$u_n \in \{0,1\} \qquad \forall\ n \in \{1, \dots, 2^d\} \quad (3.15)$$

$$w_{p,n,a} \in \{0,1\} \qquad \forall\ p \in \mathcal{P},\ n \in \{1, \dots, 2^d\},\ a \in \mathcal{A} \quad (3.16)$$

$$m_n \in \mathbb{Z}^+ \qquad \forall n \in \{1, \dots, 2^d\} \quad (3.17)$$

Equations 3.2 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 3.3) and each leaf node must have exactly one associated algorithm (Equations 3.4). Inequalities 3.5 guarantee that the recommended algorithm for a leaf node is the same as the algorithm of the problem instances allocated to this node. Inequalities 3.6 guarantee that allocations of algorithms to problem instances are performed respecting the leaf node selection for each problem instance.

Constraint set 3.7 ensures that variables u_n are 1 if and only if there is at least one problem instance associated with leaf node n. Constraints 3.8 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 3.9 ensure that any problem instance allocated in a particular node must belong to the associated parent node. Finally, constraints 3.10 and 3.11 ensure that problem instances allocated in a particular node respect the feature and branching values selected at the parent node. Constraints 3.10 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 3.11 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 3.12-3.17 are related to the domain of the decision variables defined in the model.

Chapter 4

Multivariate Model

In this chapter an integer programming model is proposed to find locally optimal multivariate splits, i.e. to define feature weights so that the best possible multivariate split can be found for one node of the decision tree. This model is different from the model presented in the previous chapter for the following reasons: (i) in the new model, the cutoff value is not defined a priori (that is, the model itself will optimize this value); (ii) For each division of a node, several features with various weights can be used. Therefore, a given instance problem is attributed to the left child node if the sum of its feature values is multiplied by the features weights generated by the model is less than or equal to the cut-off point optimized by the model; otherwise, this instance problem is attributed to the right child node and; (iii) since we only optimize the division of one node into two nodes, we do not penalize leaf nodes with few instances.

The sets and parameters are described in Section 4.1. The corresponding decision variables are described in Section 4.2. The objective function associated with the problem and the constraints are described in Section 4.3.

4.1 Input data

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

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

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

$\boldsymbol{r_{p,a}}$ cost of algorithm a for solving problem instance p;

$\boldsymbol{v_{p,f}}$ normalized [0,1] numeric value for feature f for problem instance p;

Parameters $v_{p,f}$ indicate the value of each feature f for each problem instance p. Since the integer programming model decides the branching value (cut-off point) and this value is between 0 and 1, all features values must be normalized between 0 and 1.

4.2 Decision variables

The decision variables a_f are related to the weights of the features. The branch value (cut-off point) is associated with the decision variable b. Thus, the left leaf node (variables y_p^l) will group all problem instances where the sum of the weight of features (a_f) multiplied by the feature values $v_{p,f}$ is less than the cutoff point b. Similarly, the right leaf node (variables y_p^r) will group the remaining problem instances.

The model will determine the best algorithm for each group of problem instances placed on the two leaf nodes: left leaf node (variables z_p^l) and right leaf node (variables z_p^r).

$$y_p^l = \begin{cases} 1, \text{if problem instance } p \in \mathscr{P} \text{ is at left leaf node} \\ 0, \text{otherwise.} \end{cases}$$

$$y_p^r = \begin{cases} 1, \text{if problem instance } p \in \mathscr{P} \text{ is at right leaf node} \\ 0, \text{otherwise.} \end{cases}$$

$$z_a^l = \begin{cases} 1, \text{if algorithm } a \text{ selected is at left leaf node} \\ 0, \text{otherwise.} \end{cases}$$

$$z_a^r = \begin{cases} 1, \text{if algorithm } a \text{ selected is at right leaf node} \\ 0, \text{otherwise.} \end{cases}$$

$$w_{p,a} = \begin{cases} 1, \text{if algorithm } a \text{ is recommended for problem instance } p \\ 0, \text{otherwise.} \end{cases}$$

$$a_f = \begin{cases} \text{weight (real value between 0 and 1) of feature } f. \end{cases}$$

$$b = \begin{cases} \text{branching value (cut-off point).} \end{cases}$$

With the exception of variables a_f and variable b, all other variables are binary.

4.3 Objective function and constraints

The objective of our model is to divide problem instances into two groups 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. Follows the objective function (4.1) and the set of constraints

(4.2-4.11) of our model:

$$min \sum_{p \in \mathcal{P}} w_{p,a} \times r_{p,a} \tag{4.1}$$

subject to

$$y_p^l + y_p^r = 1 \qquad \forall\ p \in \mathcal{P} \tag{4.2}$$

$$\sum_{f \in \mathcal{F}} a_f \times v_{p,f} \leq b + 1 - y_p^l \qquad \forall\ p \in \mathcal{P} \tag{4.3}$$

$$\sum_{f \in \mathcal{F}} a_f \times v_{p,f} \geq b + \epsilon - 1 + y_p^r \qquad \forall\ p \in \mathcal{P} \tag{4.4}$$

$$\sum_{a \in \mathcal{A}} z_a^l = 1 \tag{4.5}$$

$$\sum_{a \in \mathcal{A}} z_a^r = 1 \tag{4.6}$$

$$\sum_{a \in \mathcal{A}} w_{p,a} = 1 \qquad \forall\ p \in \mathcal{P} \tag{4.7}$$

$$w_{p,a} \geq y_p^l + z_a^l - 1 \qquad \forall\ p \in \mathcal{P},\ a \in \mathcal{A} \tag{4.8}$$

$$w_{p,a} \leq -y_p^l + z_a^l + 1 \qquad \forall\ p \in \mathcal{P},\ a \in \mathcal{A} \tag{4.9}$$

$$w_{p,a} \geq y_p^r + z_a^r - 1 \qquad \forall\ p \in \mathcal{P},\ a \in \mathcal{A} \tag{4.10}$$

$$w_{p,a} \leq -y_p^r + z_a^r + 1 \qquad \forall\ p \in \mathcal{P},\ a \in \mathcal{A} \tag{4.11}$$

Equations 4.2 ensure that each problem instance must be allocated to exactly one leaf node. Constraints 4.3 and 4.4 ensure that problem instances allocated to the left and right leaf nodes respect the feature(s) and branching values selected at the parent node. The parameter ϵ is defined as smallest difference between two different values of features. Equation 4.5 ensures that the left leaf node of the tree must have exactly one associated algorithm. Similarly, Equation 4.6 ensures that the right leaf node of the tree must have exactly one associated algorithm. Equations 4.7 ensure that each problem instance must be allocated to exactly one algorithm.

Inequalities 4.8 and 4.9 guarantee that the recommended algorithm for the left leaf node is the same as the algorithm of the problem instances allocated to this node and that allocations of algorithms to problem instances are performed respecting the left leaf node selection for each problem instance. Similarly, Inequalities 4.10 and 4.11 guarantee that the recommended algorithm for the right leaf node is the same as the algorithm of the problem instances allocated to this node and that allocations of algorithms to problem instances are performed respecting the right leaf node selection for each problem instance.

Chapter 5

VND to accelerate the discovery of better solutions

The model proposed in Chapter 3 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 scalability tests (Section 6.1.3) to check how practical it is the use of the 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 up 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.

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:    $\mathcal{T} \leftarrow \{\}$
 7:    GRC-ASP ($\mathcal{P}$, $\mathcal{A}$, $r$, $\mathcal{T}$, $i$, $D$, $d$, rand (0.1, 1.0), $m$, $\mathcal{E}$, $\mathcal{Q}$)
 8:    if (PerformanceDegradation ($\mathcal{T}$) < HigherPerformanceDegradation ($\mathcal{E}$)) then
 9:       if (PerformanceDegradation ($\mathcal{T}$) < LowerPerformanceDegradation ($\mathcal{E}$)) then
10:          $i \leftarrow$ -1
11:       end if
12:       $i \leftarrow i + 1$
13:       if ($\mathcal{T} \notin \mathcal{E}$) then
14:          if ($|\mathcal{E}| < m$) then
15:             $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathcal{T}\}$
16:          else
17:             $\jmath \leftarrow$ SimilarityTrees ($\mathcal{E}$, $\mathcal{T}$); $\mathcal{E}_\jmath \leftarrow \mathcal{T}$
18:          end if
19:          $\mathcal{Q} \leftarrow \mathcal{Q} \cup$ AlgorithmsLeafNodes($\mathcal{T}$)
20:       end if
21:    else
22:       $i \leftarrow i + 1$
23:    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:    $s' \leftarrow$ MIPSearch ($s$, $k$, $\ell$, $\mathcal{Q}$. $\mathcal{G}$)
30:    if ($f(s') < f(s)$) then
31:       $s \leftarrow s'$; $k \leftarrow 1$; $\mathcal{G} \leftarrow$ Shuffle($\mathcal{G}$)
32:    else
33:       $k \leftarrow k + 1$
34:    end if
35:    $ft \leftarrow$ time(); $et \leftarrow ft - st$
36: end while
37: Return $s$;
```

Line 1 initializes set $\mathcal{E}$ and structure tree $\mathcal{T}$ as empty. elite set $\mathcal{E}$ represents the set of trees/solutions. Structure tree $\mathcal{T}$ represents the tree that will be built at each iteration of the algorithm. In *Line 2*, the function **algsubset** $(\boldsymbol{r}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{E}}, \boldsymbol{q}, \boldsymbol{q'})$ is called. This function adds a subset of promising algorithms to the set $\mathcal{Q}$ as follows. The algorithms are selected according to the following MIP model: 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. The function **GRC-ASP** $(\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{A}}, \boldsymbol{r}, \boldsymbol{\mathcal{T}}, \mathbf{0}, \boldsymbol{D}, \boldsymbol{d}, \mathbf{1.0}, \boldsymbol{m}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Q}})$ is called in *Line 3*, where a Greedy Randomized algorithm is employed to generate initial feasible solution (GRC-ASP - Algorithm 2). A initial feasible solution is a decision tree that respects the constraints imposed on the integer programming model presented in Chapter 3. As we can see in the function call in *Line 3*, the first initial feasible solution built is totally greedy ($\alpha = 1.0$).

Multiple runs of this constructive algorithm are used to construct an elite set $\mathcal{E}$ of solutions (*Lines 5-24*). The repetition structure ($i < n$) is conditioned to at most n iterations without updates in the elite set. The function **GRC-ASP** $(\boldsymbol{\mathcal{P}}, \boldsymbol{\mathcal{A}}, \boldsymbol{r}, \boldsymbol{\mathcal{T}}, \boldsymbol{i}, \boldsymbol{D}, \boldsymbol{d}, \textbf{rand}\ (\mathbf{0.1}, \mathbf{1.0}), \boldsymbol{m}, \boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{Q}})$ is called in *Line 7*. For the elite set $\mathcal{E}$ to have different solutions, the parameter α is used. This parameter controls the randomization of the greedy algorithm, varying between 0.1 and 1.0 for each function call.

In *Line 8* we check if the solution obtained $\mathcal{T}$ is better than the worst solution in the elite set $\mathcal{E}$ of solutions. If yes, this solution will be included in the elite set $\mathcal{E}$. In *Line 9*, we check if the solution $\mathcal{T}$ is better than the best solution of the elite set $\mathcal{E}$. If yes, then we update the counter i. To prevent the elite set $\mathcal{E}$ from containing the same solutions, the current solution $\mathcal{T}$ is only included in the set if it is not already in it (*Line 13*). If the elite set $\mathcal{E}$ is not yet complete ($|\mathcal{E}| < m$), then we add the solution $\mathcal{T}$. If we already have m solutions in the elite set $\mathcal{E}$, then we call the function **SimilarityTrees** $(\boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{T}})$ which looks for the solution most similar to the solution $\mathcal{T}$ and then removes it and adds solution $\mathcal{T}$ to the elite set $\mathcal{E}$. The similarity between two trees is verified by calculating for each internal node how many features and cutoff point values are equal in the two trees. The subset $\mathcal{Q}$ includes all algorithms that appear in the elite set $\mathcal{E}$ (*line 19*).

The best solution of elite set $\mathcal{E}$ is used in our VND local search (MIPSearch) - *Line 25*, where parts of this solution are fixed and the remaining parts are optimized with the MIP model presented previously. The list $\mathcal{G}$ contains all sub-problems that can be obtained by fixing solution components in all neighborhoods (*Line 26*). We shuffle these sub-problems (*Line 27*) 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. In *Line 29*, the function **MIPSearch** $(\boldsymbol{s}, \boldsymbol{k}, \boldsymbol{\ell}, \boldsymbol{\mathcal{Q}}. \boldsymbol{\mathcal{G}})$ is called where our MIP based neighborhoods (Section 5.2) are explored. The parameter ℓ indicates the time limit for a single exploration of a sub-problem in the MIP based neighborhood search. Whenever the incumbent solution is updated, the list $\mathcal{G}$ is shuffled again (*Line 31*) and the algorithm starts to explore it from the beginning.

Additionally, our matheuristic has the following parameters: $\hbar$ indicates the time limit for running the entire algorithm; d is the maximum tree depth; Set D represents the different features and cut-off points of the problem instances. To control the execution time of the algorithm, we use the function **time()** which returns the current time and also the variables st and ft that represent the *start time* and the *final time* respectively.

In the following sections we describe in more details the algorithm used to generate initial feasible solutions (Section 5.1) and the MIP based neighborhoods employed in our algorithm (Section 5.2).

5.1 Constructive Algorithm

The initial solution s is obtained from the best solution of the elite set of trees $\mathcal{E}$. These trees 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) search (RESENDE and RIBEIRO, 2014) . 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. Information gain measures how much information a feature gives us about the class. Thus, the feature with the highest information gain will be tested first. 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 instance p.

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

```
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 ($\mathscr{T}$, $r$, $f$, $c$)
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 $\mathscr{T}$ 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: $\mathscr{T}_l$ = GRC-ASP ($\mathscr{P}$, $\mathscr{A}$, $r$, $\mathscr{T}$, $i$, $D_l$, $d$, $\alpha$, $m$, $n$, $\mathscr{E}$, $\mathscr{Q}$)
21: Attach $\mathscr{T}_l$ to the corresponding branch of $t$
22: $\mathscr{T}_r$ = GRC-ASP ($\mathscr{P}$, $\mathscr{A}$, $r$, $\mathscr{T}$, $i$, $D_r$, $d$, $\alpha$, $m$, $n$, $\mathscr{E}$, $\mathscr{Q}$)
23: Attach $\mathscr{T}_r$ to the corresponding branch of $t$
```

Each tree generated is limited by parameter d, which indicates the maximum depth of the tree. In this sense, the condition imposed on *Line 1*, verifies that the current depth is equal to d and if yes, terminates the algorithm. The repetition structure included in *Line 4* calculates the performance degradation of each feature f and cut-off point value c of set D and this value is assigned to matrix $\kappa_{f,c}$.

Variables $\underline{a}$ and $\overline{a}$ receive the worst and best performance degradation value from matrix κ respectively (*Lines 8 and 9*). As already mentioned, the parameter α controls the randomization of the greedy algorithm, varying between 0.1 and 1.0. Therefore, when $\alpha = 1.0$, we consider adding to the list RCL (Restricted Candidate List), initially empty - *Line 7*, only the best solution, that is, the solution with the least performance degradation ($pdt = \overline{a}$). We consider varying the value of the parameter α between 0.1 and 1.0 so that it would not be possible for all solutions to be included in the list RCL. This case occurs when $\alpha = 0.0$ and then $pdt = \underline{a}$. The value of the variable pdt is set on *Line 10* and the list RCL is updated on *Lines 11-15*.

In *Line 16*, a pair of feature and cutoff value from the list RCL is chosen at random. From there, the decision tree $\mathcal{T}$ is built and two subsets are induced (*Lines 17-19*). The subset D_l contains only problem instances whose feature value f is less than or equal the value of the cutoff point a_{rcl}. The subset D_r contains the remaining problem instances, that is, those in which the value of feature f is greater than the value of the cutoff point a_{rcl}. *Lines 20 and 22* call the algorithm recursively, where the branch of the tree on the left ($\mathcal{T}_l$) and the branch of the tree on the right ($\mathcal{T}_r$) are considered.

We also included the multivariate model presented in Chapter 4 within the constructive algorithm presented in Algorithm 2. Thus, for each node of the decision tree we use the model to consider multivariate divisions, that is, for a given node, several features with different weights can be chosen from the augmented list of features that includes hybrid features created with our MIP model. The value of this new feature for each problem instance p is calculated as follows: sum of the multiplication of the weight of each feature f by the value of each feature f for problem p.

We consider running the model for only one decision tree node. In this sense, we first consider the root node with all instances of the data set. We run the model for the root node and so now we have two subproblems related to the left child node of the root node and the right child node of the root node. We run the model again for each of the subproblems and so on, considering only a subset of the problem instances according to the cutoff values of the parent nodes. The multivariate model receives as input parameters the matrix r of performance algorithms, the set of problems $\mathcal{P}$, the set of algorithms $\mathcal{A}$, the set of features $\mathcal{F}$ and the set of all different branch values for all features D. All values of this last set are normalized between 0 and 1 to make the execution of the model viable. In normalization, we ensure that the shortest nonzero distance between two values of a given feature is at least 0.001.

Algorithm 3 (MGRC-ASP) shows the adaptations made in the constructive algorithm. As we can see, *Lines 3 and 4* are added when we compare with Algorithm

2.

Algorithm 3 MGRC-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: $(f, c) \leftarrow$ modelMultivariate($r$, $\mathcal{P}$, $\mathcal{A}$, $\mathcal{F}$, $D$)
 5: D $\leftarrow$ D $\cup$ $(f, c)$
 6: for all $(f, c)$ in $D$ do
 7:     $\kappa_{f,c} \leftarrow$ PerformanceDegradationTree ($\mathcal{T}$, $r$, $f$, $c$)
 8: end for
 9: RCL $\leftarrow \emptyset$
10: $\underline{a} \leftarrow \max(\kappa_{f,c})$
11: $\overline{a} \leftarrow \min(\kappa_{f,c})$
12: $pdt \leftarrow \underline{a} + \alpha \times (\overline{a}$ - $\underline{a})$
13: for all $(f, c)$ in $D$ do
14:     if $\kappa_{f,c} \leq pdt$ then
15:         RCL $\leftarrow$ RCL $\cup$ $(f, c)$
16:     end if
17: end for
18: $a_{rcl}$ = Randomly select a $(f, c)$ from the RCL list.
19: $t$ = Create a decision node in $\mathcal{T}$ that tests $a_{rcl}$ in the root
20: $D_l$ = Induced sub-dataset of $r$ whose value of feature $f$ are less than or equal to the
    cut-off point $a_{rcl}$
21: $D_r$ = Induced sub-dataset of $r$ whose value of feature $f$ are greater cut-off point $a_{rcl}$

22: $\mathcal{T}_l$ = GRC-ASP ($\mathcal{P}$, $\mathcal{A}$, $r$, $\mathcal{T}$, $i$, $D_l$, $d$, $\alpha$, $m$, $n$, $\mathcal{E}$, $\mathcal{Q}$)
23: Attach $\mathcal{T}_l$ to the corresponding branch of $t$
24: $\mathcal{T}_r$ = GRC-ASP ($\mathcal{P}$, $\mathcal{A}$, $r$, $\mathcal{T}$, $i$, $D_r$, $d$, $\alpha$, $m$, $n$, $\mathcal{E}$, $\mathcal{Q}$)
25: Attach $\mathcal{T}_r$ to the corresponding branch of $t$
```

5.2 Neighborhoods

Since optimizing the complete MIP model may be too expensive (according to experiments that will be presented in the next chapter), five neighborhoods have been designed to be employed in a fix-and-optimize context. Each neighborhood defines a set of sub-problems to be optimized. These neighborhoods are explained in what follows, together with examples shown in Figs. 5.1-5.5. 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 5.1, 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 5.2, 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 5.3, 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 5.4, 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 5.5, 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.

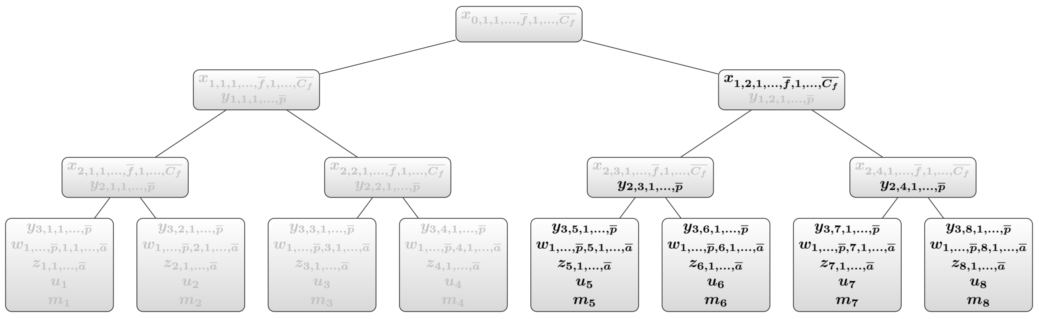

$x_{0,1,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$
$x_{1,1,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{1,1,1,\ldots,\overline{p}}$
$x_{1,2,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{1,2,1,\ldots,\overline{p}}$
$x_{2,1,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{2,1,1,\ldots,\overline{p}}$
$x_{2,2,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{2,2,1,\ldots,\overline{p}}$
$x_{2,3,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{2,3,1,\ldots,\overline{p}}$
$x_{2,4,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{2,4,1,\ldots,\overline{p}}$
$y_{3,1,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},1,1,\ldots,\overline{a}}$ $z_{1,1,\ldots,\overline{a}}$ u_1 m_1
$y_{3,2,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},2,1,\ldots,\overline{a}}$ $z_{2,1,\ldots,\overline{a}}$ u_2 m_2
$y_{3,3,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},3,1,\ldots,\overline{a}}$ $z_{3,1,\ldots,\overline{a}}$ u_3 m_3
$y_{3,4,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},4,1,\ldots,\overline{a}}$ $z_{4,1,\ldots,\overline{a}}$ u_4 m_4
$y_{3,5,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},5,1,\ldots,\overline{a}}$ $z_{5,1,\ldots,\overline{a}}$ u_5 m_5
$y_{3,6,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},6,1,\ldots,\overline{a}}$ $z_{6,1,\ldots,\overline{a}}$ u_6 m_6
$y_{3,7,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},7,1,\ldots,\overline{a}}$ $z_{7,1,\ldots,\overline{a}}$ u_7 m_7
$y_{3,8,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},8,1,\ldots,\overline{a}}$ $z_{8,1,\ldots,\overline{a}}$ u_8 m_8

Figure 5.1: Example of neighborhood $\mathcal{N}_1$ variables: variables highlighted in gray are fixed and variables highlighted in black will be optimized.

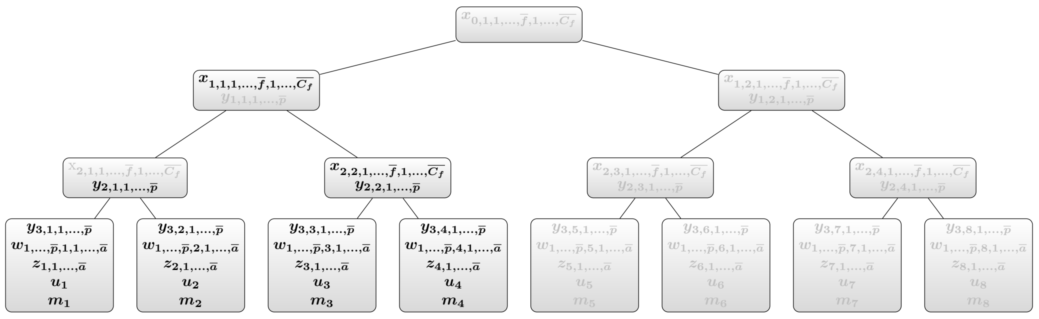

$x_{0,1,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$
$x_{1,1,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{1,1,1,\ldots,\overline{p}}$
$x_{1,2,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{1,2,1,\ldots,\overline{p}}$
$x_{2,1,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{2,1,1,\ldots,\overline{p}}$
$x_{2,2,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{2,2,1,\ldots,\overline{p}}$
$x_{2,3,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{2,3,1,\ldots,\overline{p}}$
$x_{2,4,1,\ldots,\overline{f},1,\ldots,\overline{C_f}}$ $y_{2,4,1,\ldots,\overline{p}}$
$y_{3,1,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},1,1,\ldots,\overline{a}}$ $z_{1,1,\ldots,\overline{a}}$ u_1 m_1
$y_{3,2,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},2,1,\ldots,\overline{a}}$ $z_{2,1,\ldots,\overline{a}}$ u_2 m_2
$y_{3,3,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},3,1,\ldots,\overline{a}}$ $z_{3,1,\ldots,\overline{a}}$ u_3 m_3
$y_{3,4,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},4,1,\ldots,\overline{a}}$ $z_{4,1,\ldots,\overline{a}}$ u_4 m_4
$y_{3,5,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},5,1,\ldots,\overline{a}}$ $z_{5,1,\ldots,\overline{a}}$ u_5 m_5
$y_{3,6,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},6,1,\ldots,\overline{a}}$ $z_{6,1,\ldots,\overline{a}}$ u_6 m_6
$y_{3,7,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},7,1,\ldots,\overline{a}}$ $z_{7,1,\ldots,\overline{a}}$ u_7 m_7
$y_{3,8,1,\ldots,\overline{p}}$ $w_{1,\ldots,\overline{p},8,1,\ldots,\overline{a}}$ $z_{8,1,\ldots,\overline{a}}$ u_8 m_8

Figure 5.2: Example of neighborhood $\mathcal{N}_2$ variables: variables highlighted in gray are fixed and variables highlighted in black will be optimized.

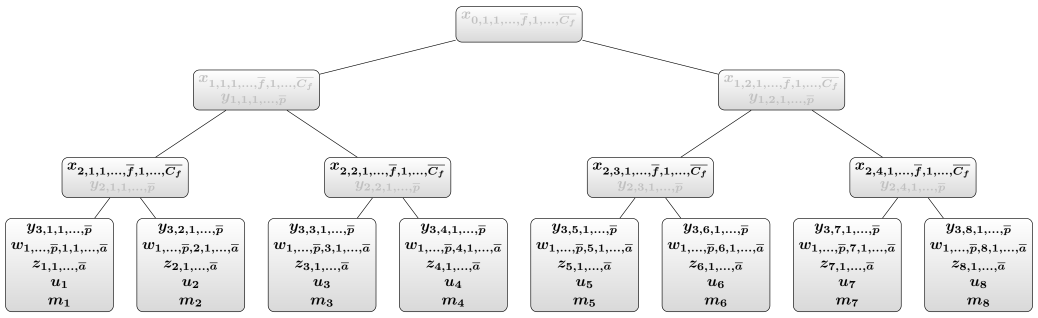


Figure 5.3: Example of neighborhood $\mathcal{N}_3$ variables: variables highlighted in gray are fixed and variables highlighted in black will be optimized.

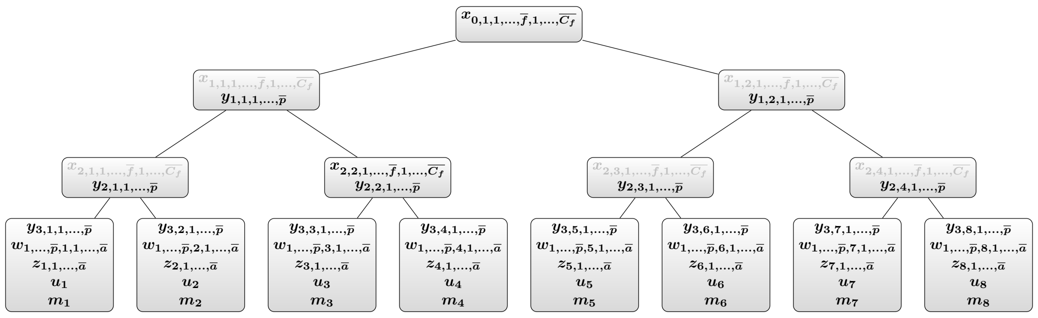


Figure 5.4: Example of neighborhood $\mathcal{N}_4$ variables: variables highlighted in gray are fixed and variables highlighted in black will be optimized.

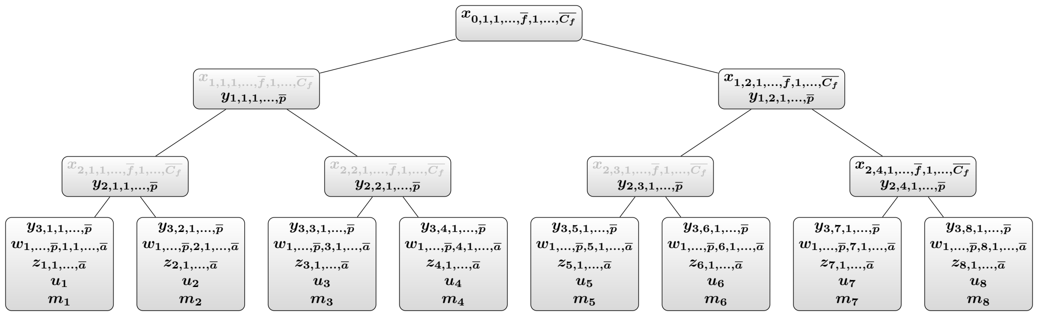


Figure 5.5: Example of neighborhood $\mathcal{N}_5$ variables: variables highlighted in gray are fixed and variables highlighted in black will be optimized.

Chapter 6

Experiments

To evaluate our approach, we built a dataset with the runtime of a set of exact algorithms in linear programming. These algorithms will be detailed in the next section. The experiments and details on this dataset, such as problem instances, features and algorithms, are presented in Section 6.1. After that, the experiments are presented considering the complete dataset and also considering use cross-validation and then divide the complete dataset into 10 partitions, where training and test partitions are used. These experiments consider only our mathematical programming heuristic.

In Section 6.2, we used the scenarios of the Open Algorithm Selection Challenge - which took place in 2017 - to compare our approach with the Random Forest algorithm. The Random Forest algorithm has been the most used technique for recommending algorithms, including in the winner of the last algorithm selection competition, as detailed in Subsections 2.1.1 and 2.1.3. Thus, we describe the performance metrics used by the competition to evaluate the algorithms, as well as presenting the 8 scenarios/datasets used.

6.1 First dataset: linear programming algorithms from the COIN-OR Linear Programming solver

6.1.1 Problem instances

Computational experiments were performed for a diverse set of 905 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 36 features 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. (2014) 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 6.1 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 $(\frac{nz}{rows}) \times 100$. The feature nz represents the number of non-zeros in the constraints and feature $rows$ represents the number of constraints of the problem instance.

Fig. 6.1 summarizes the 36 features for ASP, grouped by features related to variables, constraints and coefficients in the constraint matrix.

Table 6.1: Distribution of problem instances according to features

feature	**min**	**max**	**avg**	**sd**
cols	18	2277736	37481.15	126378.81
bin	0	2277736	26875.62	109002.76
int	0	440899	2519.47	17174.05
cont	0	799416	8086.07	49595.67
objMin	-172440000000	5084550	-191047068.15	5732090405.53
objMax	-400.07	1125210000	4877773.15	62464466.42
objAv	-25388000000	50035400	-28135134.91	843951825.78
objMed	-155250000	40212300	-213435.87	7520569.32
objAllInt	0	1	0.72	0.45
objRatioLSA	-1	2241420000000	2726084829.79	74729265604.54
rows	4	2897380	38935.69	152421.32
rpart	0	18431	275.87	1136.30
rpack	0	773664	4378.65	42031.43
rcov	0	88452	417.72	3667.38
rcard	0	430	10.87	35.36
rknp	0	103041	202.93	3541.62
riknp	0	547200	2240.30	31213.63
rflowbin	0	381806	2453.88	20335.27
density	0.00018	99.18	4.33	15.14
rflowint	0	120201	474.07	4274.16
rflowmx	0	410733	1926.49	18136.89
rother	0	2365080	26554.92	115161.44
rhsMin	-29989100	1324	-46520.02	1035978.83
rhsMax	-6.47	1E+100	1.55E+98	1.23E+99
rhsAv	-28274700	6.24E+95	7.78E+93	6.31E+94
rhsMed	-29961800	121388	-41940.55	1032770.66
rhsAllInt	-1	1E+100	1.55E+98	1.23E+99
rhsRatioLSA	-1	1.01E+103	2.02E+100	4.50E+101
equalities	0	416449	6254.27	28930.67
nz	46	27329856	414018.01	1647890.27
aMin	-4208540000	1	-5031709.32	140038859.24
aMax	-1	370795000	2102736	16366322.98
aAv	-107447000	1148410	-116070.46	3577572.84
aMed	-5590.14	10000	14.79	423.70
aAllInt	0	1	0.71	0.45
aRatioLSA	1	5786940000000	6773500440.57	192475581841.59

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 – 24.	**Number of constraints of type**: rpart, rpack, rcov, rcard, rknp, riknp, rflowbin, rflowint, rflowmx, rother and equalities.
25 – 30	**Right-hand Side Features**: rhsMin, rhsMax, rhsAv, rhsMed, rhsAllInt and rhsRatioLSA.

Coefficients in the constraint matrix features:

31 – 36.	**Variation of the coefficients**: aMin, aMax, aAv, aMed, aAllInt and aRatioLSA.

Figure 6.1: Features of problem instances of Algorithm Selection Problem: variables, constraints and coefficients in the constraint matrix.

6.1.2 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 550 different algorithm configurations were evaluated for each one of the 905 problem instances. A timeout $T = 4000$ seconds 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 $r_{pa} = 8000$ seconds. 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). Tables 6.2-6.4 show the algorithms and parameter values evaluated. The default value of each parameter is shown in parentheses right after the parameter name. Following each table, we put details about the primal, dual and barrier algorithms. 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 26% in average, a remarkable improvement considering that CLP is already the fastest open source linear programming solver.

Table 6.2: Algorithm primal simplex and parameters values evaluated

parameters	**values**
idiot (-1)	{3, 4, 5, 6, 7, 9, 10, 11, 15, 20, 25, 30, 35, 40, 50, 60, 80, 100}
crash (off)	{idiot1, idiot2, idiot3, idiot4, idiot5, idiot6, idiot7, lots, on, so}
pertv (50)	{-3500, -3157, -3000, -2395, -2000, -1483, -1000, 61}
psi (-0.5)	{-0.84, -0.62, -0.35, 0.62, 0.66, 0.84, 0.91}
sprint (-1)	{1217, 1557, 1804, 3384, 4826}
subs (3)	{1, 57, 251, 270, 294}
passp (5)	{-138, 22, 40, 80, 138}
dualize (3)	{0, 1, 2, 4}
primalp (auto)	{change, exa}
presolve (on)	{off, more}
scal (auto)	{geo, off}
perturb (on)	{off}
spars (on)	{off}

The primal simplex method exploits objective linearity and viable region convexity (PLP) to move efficiently along a sequence of extreme points until it reaches an optimal extreme point (DANTZIG, 1963). The method is generally implemented in two phases. In the first phase, an augmented system is initialized with an easily identifiable extreme-point solution using artificial variables to measure infeasibilities, and then optimized using the simplex algorithm with a view to obtaining an extreme-point solution to the augmented system that is feasible for the original system. If a solution without artificial variables cannot be found, the original linear program is infeasible. Otherwise, the second phase of the method uses the original problem formulation (without artificial variables) and the feasible extreme-point solution from the first phase and moves from that solution to a neighboring, or adjacent, solution.

The dual simplex method works implicitly on the dual problem (DLP) while operating on the constraints associated with the primal problem (PLP) (LEMKE, 1954). It does so by constructing a dual basic feasible solution, and then working to remove the primal infeasibilities. In that sense, the two algorithms are symmetric. By contrast, one can also explicitly solve the dual problem (DLP) by operating on the dual constraint set with either the primal or dual simplex method. In all cases, the algorithm moves from one adjacent extreme point to another to improve the objective function value (assuming

Table 6.3: Algorithm duals simplex and parameters values evaluated

parameters	**values**
crash (off)	{idiot1, idiot2, idiot3, idiot4, idiot5, idiot6, idiot7, lots, on, so}
dualize (3)	{0, 1}
dualp (auto)	{pesteep, steep}
passp (5)	{-167, -81, -67, -33, 0, 36, 66, 67, 93}
perturb (on)	{off}
pertv (50)	{-4900, . . . , 820}
presolve (on)	{more, off}
psi (-0.5)	{-1.1, . . . , 1.1}
scal (auto)	{equi, geo, rows}
spars (on)	{off}
sprint (-1)	{0, 468, 620, 1612, 2228}
subs (3)	{3, 37, 40, 41, 297, 4288, 4354, 4932}

non-degeneracy) at each iteration.

By default, CPLEX uses the dual simplex method. Experiments on a wide variety of linear programs have shown that this method provides the best overall performance.

However, as IBM support[1], the primal simplex method may outperform the dual simplex method on problems where the number of variables is dramatically larger than the number of constraints. This can occur because the dual simplex method requires full pricing, while the primal does not.

Table 6.4: Algorithm barrier and parameters values evaluated

parameters	**values**
cholesky (native)	{dense, univ}
passp (5)	{83}
pertv (50)	{-208, 208}
scal (auto)	{geo, off}
subs (3)	{132}

[1]https://www.ibm.com/support/pages/deciding-which-cplexs-numerous-linear-programming-algorithms-fastest-performance

The idea of the barrier algorithm is to generate a sequence of positive primal and dual solutions to a problem. Barrier method performance frequently depends upon the structure of the constraint matrix. The algorithm is very efficient for large and sparse problems. In particular, if the product of the constraint matrix and its transpose is sparse, the barrier method will probably outperform the simplex method.

According to CPLEX, the barrier method currently makes better use of multiple threads then any of CPLEX's other linear programming algorithms. So, when moving from serial machine to a parallel machine, the barrier method should be considered even if the simplex method (primal and dual) outperformed it running on only one thread.

The differences between primal, dual and barrier algorithms are as follows:

- In cases where there are several global optimals, the nature of the solutions may differ greatly when considering barrier and primal and dual methods.

- The non-crossover barrier method prevents problem optimization based on advanced start information.

- Problems that are numerically difficult for one method may be easier to solve by the other.

- The barrier optimizer works well on problems where the AAT remains sparse. In contrast, the simplex optimizers will probably perform better on problems where the AAT and the resulting Cholesky factor are relatively dense.

Gurobi Optimizer[2] provides two main algorithms to solve continuous models and the continuous relaxations of mixed-integer models: barrier and simplex. The barrier algorithm is usually faster for large, difficult models. However, it is also more numerically sensitive. And even when the barrier algorithm converges, the crossover algorithm that usually follows can stall due to numerical issues. The simplex method is often a good alternative, since it is generally less sensitive to numerical issues.

[2]https://www.gurobi.com/documentation/8.1/refman/numerics_choosing_the_righ.html

6.1.3 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. Additionally, we use our mathematical programming heuristic in order to compare the performance obtained by the heuristic in relation to the performance of the integer programming model. 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. Our experiments considered datasets with 50 algorithms and problem instances ranging from 50 to 500. Considering $d = 1$, we found the optimal solution for all experiments. Considering $d = 2$, Fig. 6.2 shows the performance of our integer programming model. Considering $d = 3$, a feasible solution was not found in the time limit.

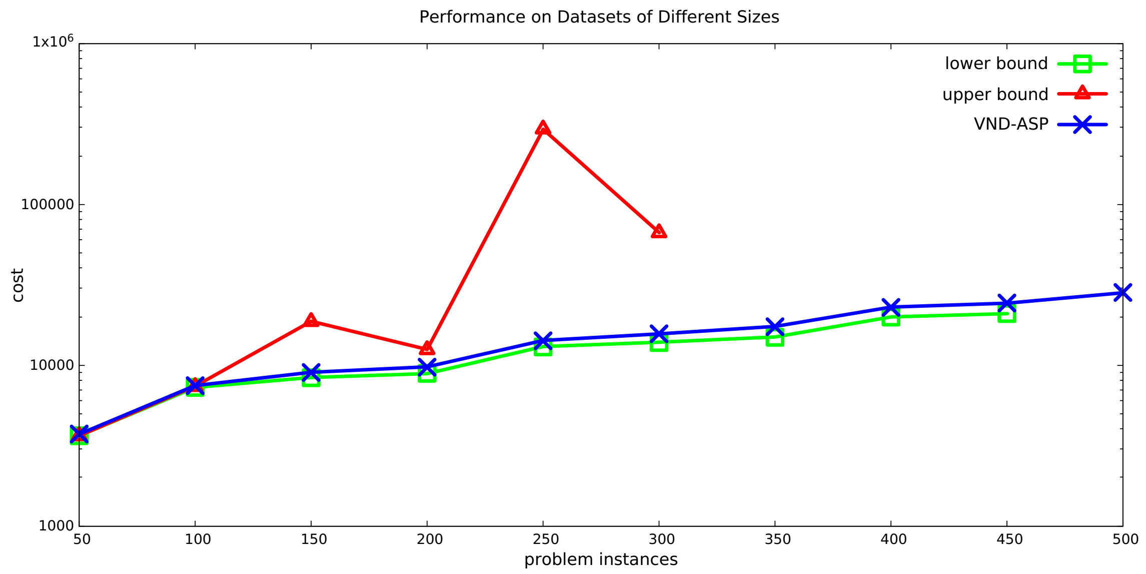


Figure 6.2: Performance of the integer programming model and the VND-ASP mathematical programming heuristic over sets of problem instances of different sizes and 50 algorithms.

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 within 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. From another point of view, we can see that the proposed heuristic obtains solutions close to the optimal solution even considering the scenario with the highest number of instances ($\mathscr{P} = 500$). Thus, for the complete dataset, experiments in the next sections were performed only with the proposed VND-ASP heuristic.

6.1.4 Experiments with the complete dataset

To create optimized decision trees considering the entire experimental results dataset is quite challenging: there are approximately half million observations[3], far beyond the limits indicated in the previous section. Thus, only experiments with our mathematical programming heuristics were conducted for this dataset.

Fig. 6.3 presents the decision tree constructed with VND-ASP using the following parameters: maximum tree depth $d = 3$, total time limit $\hbar = 72000$ seconds, MIP search timeout $l = 4000$ seconds, elite set size $m = 20$, initial algorithms subsetsize $q = 100$, $q' = 20$, minimum number of instances per leaf node $\tau = 50$ and penalty cost $\beta = 1$. The estimated performance improvement with this decision tree is 68%, a remarkable improvement.

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 Duals simplex algorithm setting the “pertv” parameter to value 50 considering 105 LP problems allocated to this node.

[3] 497750 execution results produced by solving 905 LP problems with 550 different algorithm configurations each

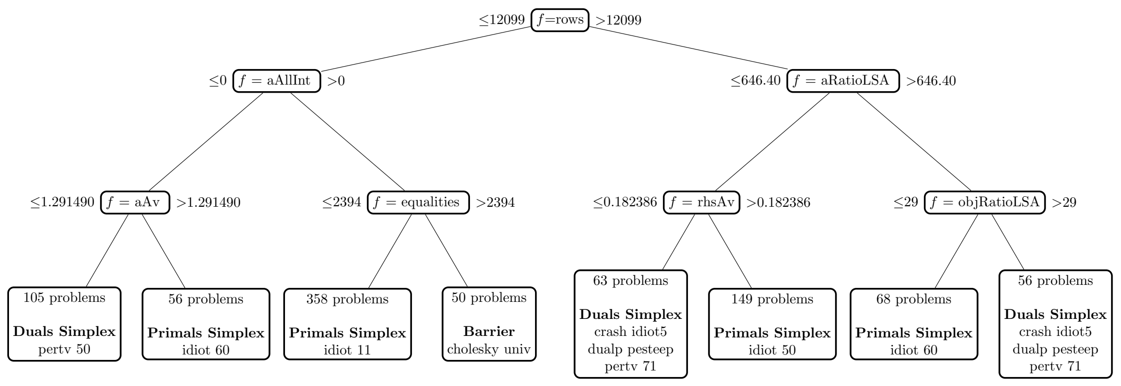
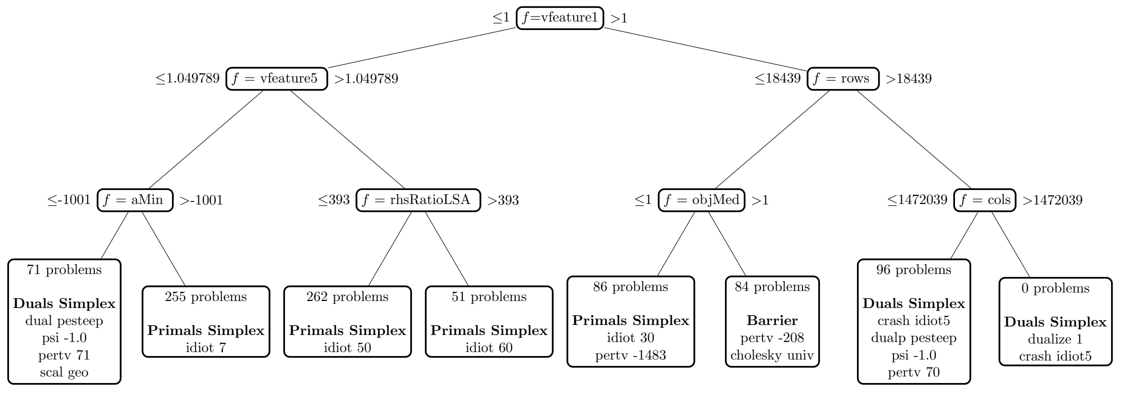

≤12099 f=rows >12099
≤0 f = aAllInt >0
≤646.40 f = aRatioLSA >646.40
≤1.291490 f = aAv >1.291490
≤2394 f = equalities >2394
≤0.182386 f = rhsAv >0.182386
≤29 f = objRatioLSA >29
105 problems Duals Simplex pertv 50
56 problems Primals Simplex idiot 60
358 problems Primals Simplex idiot 11
50 problems Barrier cholesky univ
63 problems Duals Simplex crash idiot5 dualp pesteep pertv 71
149 problems Primals Simplex idiot 50
68 problems Primals Simplex idiot 60
56 problems Duals Simplex crash idiot5 dualp pesteep pertv 71

Figure 6.3: Decision tree (univariate features) with maximum depth = 3

Fig. 6.4 presents the decision tree constructed with MVND-ASP (oblique decision trees). The estimated performance improvement with this decision tree is 72%. The parameter values used were the same as those described above.

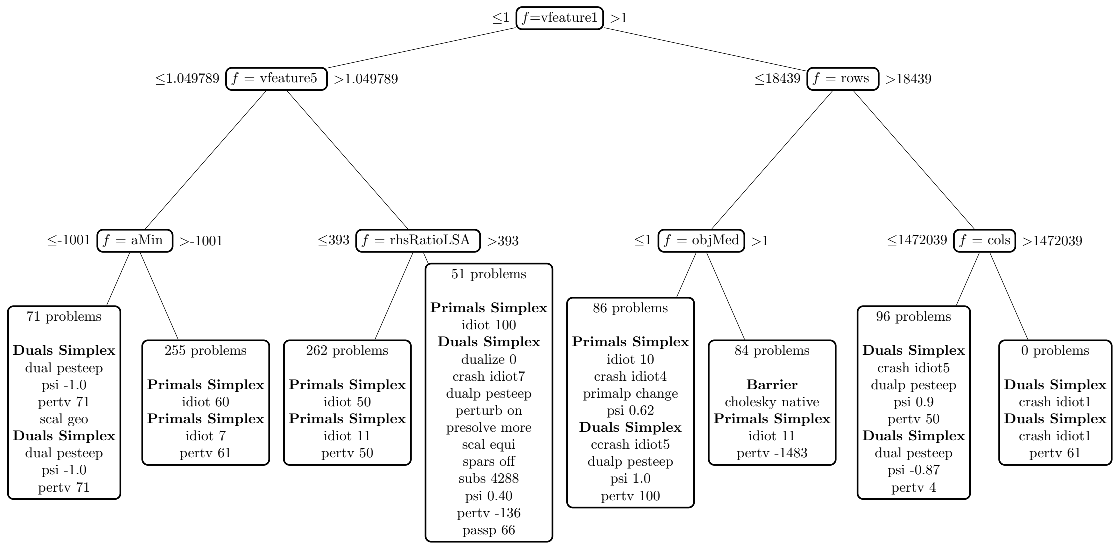

≤1 f=vfeature1 >1
≤1.049789 f = vfeature5 >1.049789
≤18439 f = rows >18439
≤-1001 f = aMin >-1001
≤393 f = rhsRatioLSA >393
≤1 f = objMed >1
≤1472039 f = cols >1472039
71 problems Duals Simplex dual pesteep psi -1.0 pertv 71 scal geo
255 problems Primals Simplex idiot 7
262 problems Primals Simplex idiot 50
51 problems Primals Simplex idiot 60
86 problems Primals Simplex idiot 30 pertv -1483
84 problems Barrier pertv -208 cholesky univ
96 problems Duals Simplex crash idiot5 dualp pesteep psi -1.0 pertv 70
0 problems Duals Simplex dualize 1 crash idiot5

feature/weights	**rows**	**nz**	**density**	**int**	**cont**	**rpart**	**rknp**	**rflowmx**
vfeature1	0.145568	0.059260	0.028605	0.045253	0.018778	0.063069	0.034485	0.018388
	rother	**objMin**	**objMax**	**objAllInt**	**objRatioLSA**	**rhsMin**	**aAv**	**aRatioLSA**
	0.105879	0.096289	0.096512	0.017031	0.150023	0.641356	0.248940	0.175072

feature/weights	**cols**	**rows**	**equalities**	**bin**	**rpart**	**rother**	**objMin**
vfeature5	0.028135	0.013005	0.104403	0.100460	0.000411	0.226801	0.166909
	objMax	**objAv**	**rhsMin**	**aMin**	**aMed**	**aAllInt**	
	0.553704	0.016188	1.000000	0.048103	0.324919	0.039324	

Figure 6.4: Decision tree (multivariate features) with maximum depth = 3

Fig. 6.5 presents the decision tree constructed with MVND-ASP (oblique decision trees) considering two recommended algorithms. The estimated performance improvement with this decision tree is 83%.



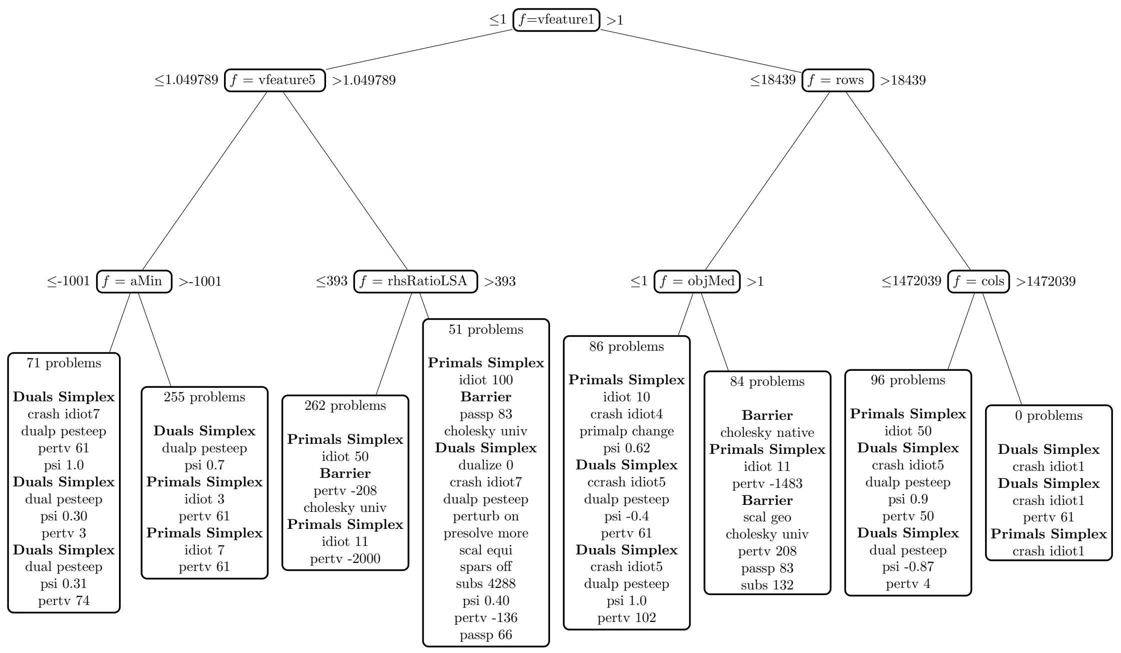
feature/weights	rows	nz	density	int	cont	rpart	rknp	rflowmx
vfeature1	0.145568	0.059260	0.028605	0.045253	0.018778	0.063069	0.034485	0.018388
	rother	**objMin**	**objMax**	**objAllInt**	**objRatioLSA**	**rhsMin**	**aAv**	**aRatioLSA**
	0.105879	0.096289	0.096512	0.017031	0.150023	0.641356	0.248940	0.175072

feature/weights	cols	rows	equalities	bin	rpart	rother	objMin
vfeature5	0.028135	0.013005	0.104403	0.100460	0.000411	0.226801	0.166909
	objMax	**objAv**	**rhsMin**	**aMin**	**aMed**	**aAllInt**	
	0.553704	0.016188	1.000000	0.048103	0.324919	0.039324	

Figure 6.5: Decision tree (multivariate features) with maximum depth = 3 and two recommended algorithms per leaf node

Fig. 6.6 presents the decision tree constructed with MVND-ASP (oblique decision trees) considering three recommended algorithms. The estimated performance improvement with this decision tree is 85%.

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.


≤1 f=vfeature1 >1
≤1.049789 f = vfeature5 >1.049789
≤18439 f = rows >18439
≤-1001 f = aMin >-1001
≤393 f = rhsRatioLSA >393
≤1 f = objMed >1
≤1472039 f = cols >1472039
71 problems
Duals Simplex crash idiot7 dualp pesteep pertv 61 psi 1.0
Duals Simplex dual pesteep psi 0.30 pertv 3
Duals Simplex dual pesteep psi 0.31 pertv 74
255 problems
Duals Simplex dualp pesteep psi 0.7
Primals Simplex idiot 3 pertv 61
Primals Simplex idiot 7 pertv 61
262 problems
Primals Simplex idiot 50
Barrier pertv -208 cholesky univ
Primals Simplex idiot 11 pertv -2000
51 problems
Primals Simplex idiot 100
Barrier passp 83 cholesky univ
Duals Simplex dualize 0 crash idiot7 dualp pesteep perturb on presolve more scal equi spars off subs 4288 psi 0.40 pertv -136 passp 66
86 problems
Primals Simplex idiot 10 crash idiot4 primalp change psi 0.62
Duals Simplex ccrash idiot5 dualp pesteep psi -0.4 pertv 61
Duals Simplex crash idiot5 dualp pesteep psi 1.0 pertv 102
84 problems
Barrier cholesky native
Primals Simplex idiot 11 pertv -1483
Barrier scal geo cholesky univ pertv 208 passp 83 subs 132
96 problems
Primals Simplex idiot 50
Duals Simplex crash idiot5 dualp pesteep psi 0.9 pertv 50
Duals Simplex dual pesteep psi -0.87 pertv 4
0 problems
Duals Simplex crash idiot1
Duals Simplex crash idiot1 pertv 61
Primals Simplex crash idiot1

feature/weights	**rows**	**nz**	**density**	**int**	**cont**	**rpart**	**rknp**	**rflowmx**
vfeature1	0.145568	0.059260	0.028605	0.045253	0.018778	0.063069	0.034485	0.018388
	rother	**objMin**	**objMax**	**objAllInt**	**objRatioLSA**	**rhsMin**	**aAv**	**aRatioLSA**
	0.105879	0.096289	0.096512	0.017031	0.150023	0.641356	0.248940	0.175072

feature/weights	**cols**	**rows**	**equalities**	**bin**	**rpart**	**rother**	**objMin**
vfeature5	0.028135	0.013005	0.104403	0.100460	0.000411	0.226801	0.166909
	objMax	**objAv**	**rhsMin**	**aMin**	**aMed**	**aAllInt**	
	0.553704	0.016188	1.000000	0.048103	0.324919	0.039324	

Figure 6.6: Decision tree (multivariate features) with maximum depth = 3 and three recommended algorithms per leaf node

6.1.5 Experiment using cross-validation on the complete dataset 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 448250 examples (815 problem instances × 550 available algorithms), with the exception of the last five partitions that contained 447700 examples (814 problem instances × 550 available algorithms). The results of the cross-validation are given in Fig. 6.7. This figure shows the average performance degradation considering the ideal performance to solve the LP relaxation of all problem instances (the virtual best solver). Results of UVND-ASP (univariate features) and MVND-ASP (multivariate features/oblique decision trees) with maximum tree depth 3 (UVND-ASP(D=3) and MVND-ASP(D=3)), maximum tree depth 4 (UVND-ASP(D=4) and MVND-ASP(D=4)) and maximum tree depth 5 (UVND-ASP(D=5) and MVND-ASP(D=5)) are included. The remaining parameters of UVND-ASP and MVND-ASP are the same described in the previous subsection. We also compare our results with the results produced by the best configuration of the Random Forest (RF) algorithm implemented in Weka (HALL et al., 2009) (RF(T=40), where T is the number of trees)). We tested parameter T at the following values: 1, 3, 5, 10, 20, 30, 40, 50, 100, 120 and 150. Results obtained selecting a single best algorithm (SBA) are also included. Default CBC settings (Default) and results obtained selecting the virtual best algorithm (vba) are also included.

We also consider testing our model, using not only the decision tree recommended algorithm for each leaf node, but the two best or three best algorithms for instance problems allocated on that leaf node. Thus, in the decision tree built for each partition, we choose the 2 (or 3) best configurations for each leaf node and evaluate them in the test partition. The results are presented in *MVND-ASP(D=4;RA=2)* and *MVND-ASP(D=4;RA=3)*.

As can be seen, our results indicate performance gains of 72.5% compared against default settings. This value is obtained by the formula $(1 - (\frac{27954.4}{101816.8})) \times 100\%$. Moreover, they are noticeably better than those obtained when selecting only the single best algorithm (2%). UVND-ASP and MVND-ASP results are also mostly superior to the ones obtained with RF with 40 trees. 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 depend 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.

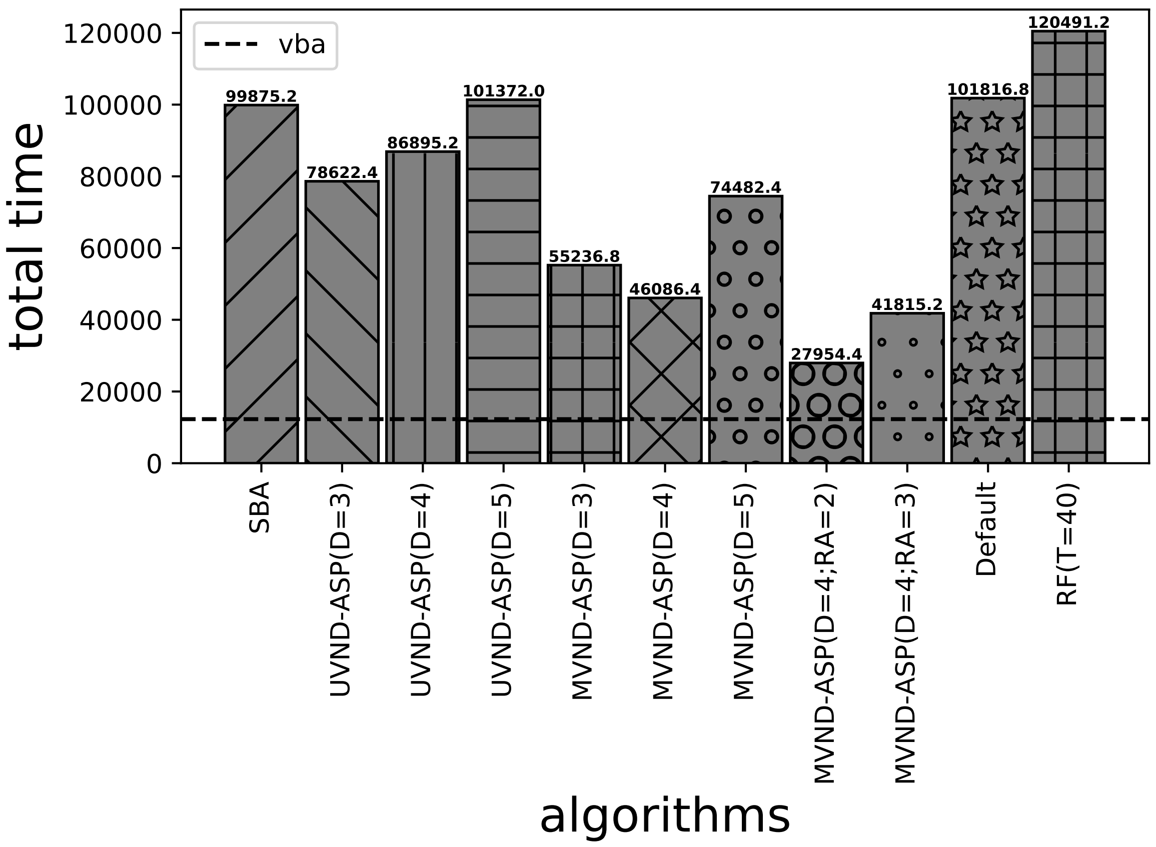


Figure 6.7: Cross-validation results for all partitions of First Dataset

6.2 The ICON challenge

In this section, we use the Algorithm Selection Benchmark Library - ASlib (BISCHL et al., 2016) to compare our method with the Random Forest algorithm. Aslib consists of many scenarios for which performance data for all algorithms in all instances is available. Two competitions were held based on the Aslib: the ICON Challenge on Algorithm Selection (2015) and the Open Algorithm Selection Challenge - OASC (2017). We consider our experiments only with the scenarios of the Competition held in 2017. We were motivated by being the last competition, which suggests improvements in the algorithms that competed in 2015 and also the inclusion of new algorithms.

In the 2017 competition, participants sent only predictions made by the system to new test instances. In this case, 2/3 of the instances were assigned to the training partition and 1/3 of the instances were assigned to the test partition. Only the values of the features of the test instances were provided. In addition, there is no limit on the computational resources that can be used.

In the competition, benchmarks for the selection of algorithms from different domains were used, in a total of 11 scenarios: 8 of the 11 scenarios were completely new and were not disclosed to the participants before the competition. In our experiments, we considered only 8 out of 11 scenarios. The 3 scenarios not considered deal with the objective of the quality of the solution - we are interested in the runtime. Table 6.5 describes the 8 scenarios, showing the name, the number of instances, features and algorithms.

Table 6.5: Scenarios of the OASC (2017) - Runtime Objective

scenario	**instances**	**features**	**algorithms**
BNSL-2016 (Bado)	1179	87	8
CSP-Minizine-Obj-2016 (Caren)	100	95	8
MAXSAT-PMS-2016 (Magnus)	601	37	19
MAXSAT-WPMS-2016 (Monty)	630	37	18
MIP-2016 (Mira)	218	143	5
QBF-2016 (Quill)	825	46	24
SAT12-ALL (Svea)	1614	115	31
SAT03-16_INDU (Sora)	2000	483	10

It is important to note that the competition allowed different types of algorithm selection systems, namely: selection of a single algorithm; selection of a schedule of algorithms; use of pre-solving schedules and; use of different instance feature sets (e.g., on a per-instance base). Participants in the competition sent systems considering the selection of a schedule of algorithms, which makes a fair comparison with our algorithm, which recommends only one algorithm for each instance (**selection of a single algorithm**).

We performed experiments with our method UVND-ASP considers just univariate features. We decided not to test the approach MVND-ASP that considers multivariate features. As the participants of the Competition OASC-2017 did not add new features to existing ones, we think it is fair not to include new features. We tested trees with maximum depth varying between 3 and 5 ($d = 3, 4, 5$). We compared our method with the 8 methods used in the competition. The results are shown in the Table 6.6.

In results, the score measures the proportion of gap closed in terms of PAR10 defined as (1 - $PAR10$) where $PAR10$ is defined as in equation $(F(S) - F(oracle))/(F(SB) - F(oracle))$. $F(S)$ represents the performance of the method. $F(SB)$ represents the performance of the algorithm with the best performance on average when running the instances. $F(oracle)$ represents the performance of the optimal case, that is, the case where for each instance, is chosen algorithm with better performance. The gap closed measure is 1 if the system reaches the oracle performance and 0 if performance is the same as SB.

Table 6.6: Results of the OASC (2017) - Runtime Objective

scenario	**ASAP.v2**	**ASAP.v3**	**UVND-ASP(d=3)**	**UVND-ASP(d=4)**	**UVND-ASP(d=5)**	**Sunny-fkvar**	**Sunny-autok**	**star-zilla_dyn_sched**	**star-zilla**	**AS-RF**	**AS-ASL**
Bado	0.761	0.808	0.833	0.890	0.911	0.847	0.748	0.484	0.707	0.836	0.681
Caren	0.588	0.590	0.372	0.372	0.180	0.945	0.783	0.777	-0.001	-0.659	-1.068
Magnus	0.508	0.506	0.248	0.722	0.263	0.581	0.502	0.590	0.583	-1.012	-1.013
Monty	0.833	0.763	0.628	0.622	0.622	-0.910	0.632	0.487	0.173	-7.482	-6.973
Mira	0.505	0.509	0.029	0.029	0.029	0.432	-0.014	-1.337	0.033	0.495	-0.407
Quill	0.698	0.580	0.334	0.246	0.456	0.569	0.850	0.459	0.308	-0.328	-0.299
Svea	0.677	0.688	0.315	0.393	0.416	0.658	0.579	0.171	0.171	0.457	0.439
Sora	0.350	0.225	0.371	0.363	0.354	0.179	0.173	0.313	0.313	-0.135	-0.383
Average	**0.615**	**0.584**	**0.391**	**0.455**	**0.404**	**0.413**	**0.532**	**0.243**	**0.286**	**-0.979**	**-1.128**

The ASAP.v2 and ASAP.v3 techniques achieved the best results, followed by the Sunny-autok technique. We can see that our method, considering $d = 4$, ranked fourth in the overall ranking. An adjustment in the values of parameters used in our method, such as a minimum number of problems per leaf node (parameter τ) and penalty (parameter β) for leaf nodes with few instances could contribute to a better result, which we certainly place as future work on this thesis.

For a better evaluation of our method, we consider experiments with Random Forest algorithm on 8 scenarios of the competition. We considered the selection of the best algorithm for each instance and tested 3 different values for the parameter (named as parameter t) related to the number of decision trees generated by the method. The results of our method and the Random Forest method are shown in Table 6.7.

Table 6.7: Results on the scenarios of the OASC (2017) - Our method vs Random Forest

scenario	**RF (t = 50)**	**RF (t = 100)**	**RF (t = 150)**	**UVND-ASP(d=3)**	**UVND-ASP(d=4)**	**UVND-ASP(d=5)**
Bado	0.854	0.910	0.874	0.833	0.890	0.911
Caren	0.387	0.173	-0.224	0.372	0.372	0.180
Magnus	0.160	0.177	0.159	0.248	0.722	0.263
Monty	-0.488	-0.347	-0.276	0.628	0.622	0.622
Mira	-0.433	0.953	0.521	0.029	0.029	0.029
Quill	0.427	0.577	0.484	0.334	0.246	0.456
Svea	0.671	0.685	0.701	0.315	0.393	0.416
Sora	0.217	0.308	0.313	0.371	0.363	0.354
Average	**0.224**	**0.429**	**0.319**	**0.391**	**0.455**	**0.404**

The results presented indicate that our algorithm has a better performance than the Random Forest algorithm. In addition, our method provides better interpretability, since we limit the maximum depth of the tree. Another positive point is that in all scenarios our results are better than the single best algorithm - which is not the case with Random Forest.

Chapter 7

Discussion and Closing Remarks

This thesis 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 26% faster than default CLP settings.

Scalability tests were performed in increasingly larger datasets to check up to which size it was possible to optimally solve this problem until 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 multivariate tree using three algorithms recommendations per leaf node were 85% faster than CLP default settings, almost doubling the improvement that could be obtained using a single best parameter setting. Considering the first dataset, our results are much better than those obtained after tuning the Random Forest algorithm, with the 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.

We performed additional experiments using 8 scenarios that were used in the Open Algorithm Selection Challenge 2017. The competition allowed different types of algorithm selection systems, so that the participants sent systems considering the selection of a schedule of algorithms, which makes a fair comparison impossible with our method (selection of a single algorithm). Still, our results are comparable to the competition's winning system. For a better evaluation of our method, we consider experiments with Random Forest algorithm on the scenarios of the competition. The results indicate that our method has a better performance than the RF algorithm and in all scenarios our results are better than the single best algorithm.

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. In addition, we will conduct further experiments on the bases studied in this thesis to verify the impacts of parameter values, such as the penalty value for poorly representative leaf nodes and/or the threshold value that indicates whether a leaf node is with little representation of problem instances.

Appendix A

Appendix

This appendix lists the papers developed during the course of the doctoral thesis.

Vilas Boas, M. G., Santos, H. G., Merschmann, L. H. and Vanden Berghe, G. (2019). **Optimal Decision Trees for the Algorithm Selection Problem: Integer Programming based Approaches**. *International Transactions in Operational Research.*

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. International Conference on Computational Science. ICCS 2017, 12-14 June 2017. Zurich, Switzerland.

Bibliography

Aha, D. W.: 1992, Generalizing from case studies: A case study, *In Proceedings of the Ninth International Conference on Machine Learning*, Morgan Kaufmann, pp. 1–10.

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

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

Bernstein, A., Provost, F. and Hill, S.: 2005, Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification, *Knowledge and Data Engineering, IEEE Transactions on* **17**, 503 – 518.

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

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K. and Vanschoren, J.: 2016, Aslib: A benchmark library for algorithm selection, *Artificial Intelligence* **237**, 41 – 58.

Breiman, L.: 1996, Bagging predictors, *Machine Learning* **24**(2), 123–140.

Breiman, L.: 2001, Random forests, *Machine Learning* **45**(1), 5–32.

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

Brown, D. E., Pittard, C. L. and Park, H.: 1996, Classification trees with optimal multivariate decision nodes, *Pattern Recognition Letters* **17**(7), 699 – 703.

Dantzig, G.: 1963, *Linear programming and extensions*, Rand Corporation Research Study, Princeton Univ. Press, Princeton, NJ.

Doyle, P.: 1973, The use of automatic interaction detector and similar search procedures, *Journal of The Operational Research Society - J OPER RES SOC* **24**, 465–467.

Einhorn, H.: 1972, Alchemy in the behavioral sciences, *Public Opinion Quarterly - PUBLIC OPIN QUART* **36**.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M. and Hutter, F.: 2015, Efficient and robust automated machine learning, *in* C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett (eds), *Advances in Neural Information Processing Systems 28*, Curran Associates, Inc., pp. 2962–2970.

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

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

Freund, Y. and Schapire, R. E.: 1995, A Decision Theoretic Generalization of On-Line Learning and an Application to Boosting, *in* P. M. B. Vitányi (ed.), *Second European Conference on Computational Learning Theory (EuroCOLT-95)*, pp. 23–37.

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

Gearhart, J. L., Adair, K. L., Detry, R. J., Durfee, J. D., Jones, K. A. and Martin, N.: 2013, Comparison of open-source linear programming solvers, *Technical report*, Sandia National Laboratories.

Giraud-Carrier, C.: 2006, The data mining advisor: meta-learning at the service of practitioners, Vol. 2005, p. 7 pp.

Gonard, F., Schoenauer, M. and Sebag, M.: 2017, Algorithm Selector and Prescheduler in the ICON challenge, *in* E.-G. Talbi and A. Nakib (eds), *Bioinspired heuristic optimization* , Computational Intelligence, Springer Verlag.

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

Hutter, F., Hoos, H., Leyton-Brown, K. and Stützle, T.: 2009, Paramils: An automatic algorithm configuration framework, *J. Artif. Intell. Res. (JAIR)* **36**, 267–306.

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

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

Johnson, E., Nemhauser, G. and 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. and Tierney, K.: 2010, Isac –instance-specific algorithm configuration, *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.

Kerschke, P., Hoos, H. H., Neumann, F. and Trautmann, H.: 2019, Automated algorithm selection: Survey and perspectives, *Evolutionary Computation* **27**(1), 3–45. PMID: 30475672.

Kim, H. and Loh, W.-Y.: 2001, Classification trees with unbiased multiway splits, *Journal of the American Statistical Association* **96**(454), 589–604.

King, R., Feng, C. and Sutherl, A.: 2000, Statlog: Comparison of classification algorithms on large real-world problems, *Applied Artificial Intelligence* **9**.

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.

Kotthoff, L.: 2012, Algorithm selection for combinatorial search problems: A survey, *AI Magazine* **35**.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F. and Leyton-Brown, K.: 2017, Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka, *Journal of Machine Learning Research* **18**(25), 1–5.

Lemke, C. E.: 1954, The dual method of solving the linear programming problem, *Naval Research Logistics Quarterly* **1**(1), 36–47.

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

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

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. and Stützle, T.: 2014, An Analysis of Parameter Adaptation in Reactive Tabu Search, *International Transactions in Operational Research* **21**(1), 127–152.

Menickelly, M., Gunluk, O., Kalagnanam, J. and Scheinberg, K.: 2016, Optimal generalized decision trees via integer programming.

Messenger, R. and Mandell, L.: 1972, A modal search technique for predictive nominal scale multivariate analysis, *Journal of the American Statistical Association* **67**(340), 768–772.

Michie, D., Spiegelhalter, D. J., Taylor, C. C. and Campbell, J. (eds): 1995, *Machine Learning, Neural and Statistical Classification*, Ellis Horwood, USA.

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

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

Morgan, J. N. and Sonquist, J. A.: 1963, Problems in the analysis of survey data, and a proposal.

Murthy, S. K., Kasif, S. and Salzberg, S.: 1994, A system for induction of oblique decision trees, *J. Artif. Int. Res.* **2**(1), 1–32.

Mısır, M. and Sebag, M.: 2017, Alors: An algorithm recommender system, *Artificial Intelligence* **244**, 291 – 314. Combining Constraint Solving with Mining and Learning.

Pochet, Y. and 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. and Neumann, F.: 2014, A comprehensive benchmark set and heuristics for the traveling thief problem, *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. and Ribeiro, C.: 2014, *GRASP: Greedy randomized adaptive search procedures*, Springer US, pp. 287–312.

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

Salisu, M., Abdulrahman, Adamu, A., Ado, Y. and Rilwan, A.: 2017, An overview of the algorithm selection problem, *International Journal of Computer (IJC)* .

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

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

Souza, M., Coelho, I., Ribas, S., Santos, H. and 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.

Thornton, C., Hutter, F., Hoos, H. and Leyton-Brown, K.: 2012, Auto-weka: Combined selection and hyperparameter optimization of classification algorithms, *KDD* .

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

Wolpert, D. H. and Macready, W. G.: 1997, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation* **1**(1), 67–82.

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

Xu, L., Hutter, F., Shen, J., Hoos, H. H. and Leyton-brown, K.: n.d., Satzilla2012: Improved algorithm selection based on cost-sensitive classification models.

Zhang, H., Wang, S., Xu, X., Chow, T. W. and 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.