UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA

ANYTIME AUTOMATIC ALGORITHM SELECTION FOR THE PSEUDO-BOOLEAN OPTIMIZATION PROBLEM

Catalina Pezo Vergara

Thesis presented in "Departamento de Ingeniería Informática y Ciencias de la Computación, Facultad de Ingeniería, Universidad de Concepción" to obtain the title of "Magíster en Ciencias de la Computación"

October 2023
Concepción, Chile

Supervisor: Julio Godoy del Campo, Roberto Asín Achá
Evaluating Committee: Pedro Pinacho, Albert Oliveras

Abstract

Machine learning (ML) techniques have been proposed to automatically select the best solver from a portfolio of solvers, based on predicted performance. These techniques have been applied to various problems, such as Boolean Satisfiability, Traveling Salesperson, Graph Coloring, and others. These methods, known as meta-solvers, take an instance of a problem and a portfolio of solvers as input, then predict the best-performing solver and execute it to deliver a solution. Typically, the quality of the solution improves with a longer computational time. This has led to the development of *anytime* selectors, which consider both the instance and a user-prescribed computational time limit. *Anytime meta-solvers* predict the best-performing solver within the specified time limit.

In this study, we focus on the task of designing anytime meta-solvers for the NP-hard optimization problem of *Pseudo-Boolean Optimization* (PBO). The effectiveness of our approach is demonstrated via extensive empirical study in which our anytime meta-solver improves dramatically on the performance of Mixed Integer Programming solver Gurobi, the best-performing single solver in the portfolio.

Keywords – Meta-Algorithm, Pseudo-Boolean Optimization, Anytime Algorithm Selection

Contents

List of Tables

List of Figures

1 Introduction

Per-instance Automatic Algorithm Selection (AAS), first proposed in [1], consists of, for a given instance of a known problem and a portfolio of algorithms for the problem, a prediction of an algorithm in the portfolio that best solves the given instance. The prediction is done by Machine Learning models that are trained on a set of problem instances. This is of particular interest for NP-hard optimization problems since, for such problems, there is no single algorithm that dominates the others on every instance in every possible scenario. The *anytime behavior* of an algorithm, when a feasible solution is available, is its profile of improvement in the objective function value at each successive time step. *Anytime Automatic Algorithm Selection* aims to choose the algorithm which is expected to find the best possible solution, within the given time limit, for a specific instance. Previously, Anytime Automatic Algorithm Selection meta-solvers were proposed for the Knapsack [2] and Traveling Salesperson [3] problems.

We devise here an Anytime Automatic Algorithm Selection for the NP-hard Pseudo-Boolean Optimization problem (PBO) [4], which generalizes Satisfiability and Maximum Satisfiability problems [5]. PBO is an optimization problem with an objective function that is a Pseudo-Boolean function and subject to constraints that are (in)equalities over Boolean variables. Many problems are typically modeled as PBO, including hardware and software verification [6, 7], software dependency [8], planning [9], scheduling problems [10], among others. As such, improving the ability to deliver high-quality solutions for PBO can impact the solvability of a broad range of problems. Indeed, a number of commercial and publicly available algorithms (solvers) have been proposed for PBO and the SAT community maintains the Pseudo-Boolean Competition [11] in which the performance of state-of-the-art solvers is assessed.

This work describes a meta-solver that, for a given instance and time limit, i) predicts, using a Machine Learning model, which solver, among a portfolio of solvers, will deliver a best quality (smallest objective value) feasible solution and ii) executes such a solver. For this, we propose different machine learning models and assess their performance. Our experiments demonstrate that our meta-solver outperforms all the individual solvers in the portfolio by a wide margin. In particular, our meta-solver outperforms Gurobi – which is the dominant solver in the portfolio – in achieving better quality solutions, for a portion of the instances and time limits where Gurobi finds feasible solutions. In 47% out of the cases where Gurobi does not identify feasible solutions, our meta-solver does find feasible solutions. A major contribution of our meta-solver is that identifies with great precision when feasibility is *not* expected to be attained for the given instance, within the specified time limit.

Beyond achieving improved results, our study provides insights into the most important features that determine the choice of the best solver. We identify the fraction of the number of terms that appear on the objective function, out of the total number of terms in the objective and the constraints, as a major feature. This feature has not appeared previously in algorithm selection studies on SAT and MaxSAT. Another major feature is the prescribed time limit, which appears to be more important than other characteristics of the instances in determining the solver selection.

This thesis is organized as follows: Section 2 presents essential concepts and terminology. In Section 3 we discuss related work. Section 4 describes the meta-solver construction and Section 5 presents and analyzes experimental results. Finally, Section 6 discusses future work and conclusions.

This project was done with the Meta-Algorithms Research Group of the University of California, Berkeley.

1.1 Hypothesis

The use of Machine Learning techniques will allow to generate a meta-solver for the Pseudo-Boolean Optimization Problem that will adequately select a solver from a portfolio of solvers, given an instance and a user specified time limit. This meta-solver will outperform the best-performing single solver, in regards to a performance metric.

1.2 Objectives

1.2.1 General Objective

To implement an Anytime Automatic Algorithm Selection meta-solver based on a Machine Learning model that improves the state-of-the-art for the Pseudo-Boolean Optimization Problem.

1.2.2 Specific Objectives

- To design an efficient experimentation environment for evaluating the anytime behavior of the state-of-the-art solvers for the Pseudo-Boolean Optimization problem.
- To collect Pseudo-Boolean Optimization algorithms and adapt them to observe their anytime behavior.
- To identify patterns that characterize and differentiate instances of the Pseudo-Boolean Optimization problem, for the purpose of finding an indicator of the best way each instance can be solved. In order to be useful, these characteristics must be easy to compute.
- To create a Machine Learning model that recommends a solver which will give the best feasible solution for a specific instance of the Pseudo-Boolean Optimization problem, given a time limit restriction.
- To evaluate the created meta-solver and compare its performance against the best-performing single solver of the Pseudo-Boolean Optimization problem.

1.3 Limitations

- For the present work, only time will be considered as a computational resource in order to make the selection of the best Pseudo-Boolean Optimization solver. Space will not be considered.

2 Preliminaries

In this Section, we provide formal definitions of the Pseudo-Boolean Optimization problem, Machine Learning and the Automatic Algorithm Selection problem. In addition, we give an overview of the Machine Learning models that we use, and we also present a performance metric, called the $\hat{m}$, that is used in addition to accuracy and confusion matrix to assess the performance of the proposed meta-solver.

To facilitate the reading of the paper we provide, in Table 2.1, a list of acronyms used throughout.

Acronym	**Definition**
AAS	Automatic Algorithm Selection
AAAS	Anytime Automatic Algorithm Selection
ASLib	Algorithm Selection Library
ASP	Answer Set Programming
BCS	Boolean Constraint Satisfaction
BDD	Binary Decision Diagram
CDCL	Conflict Driven Clause Learning
GB	Gradient Boosting
CNF	Conjunctive Normal Form
CNN	Convolutional Neural Network
KNN	K-Nearest Neighbors
LP	Linear Programming
LS	Local Search
LSU	Linear SAT-UNSAT algorithm
MaxSAT	Maximum Boolean Satisfiability problem
ML	Machine Learning
MIP	Mixed Integer Programming
NaPS	Nagoya Pseudo-Boolean Solver
PB	Pseudo-Boolean
PBO	Pseudo-Boolean Optimization
RF	Random Forest
SAT	Boolean Satisfiability problem
SBS	Single Best Solver
TSP	Traveling Salesperson problem
VBS	Virtual Best Solver
WBO	Weighted Boolean Maximization
WPM	Weighted Partial MaxSAT

Table 2.1: Acronyms used in this paper

2.1 Pseudo-Boolean Optimization (PBO)

A Pseudo-Boolean function is a mapping $f : \{0,1\}^n \to \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers, [4]. A Pseudo-Boolean Optimization Problem (PBO) is formulated for an array of Boolean variables x as follows:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_1(x) \geq a_1 \\ & \vdots \\ & g_n(x) \geq a_n \\ & x \in \{0,1\}^n. \end{aligned}$$

Without loss of generality, the constraints are of the form $g_i(x) = b_1t_1 + b_2t_2 + \ldots + b_nt_m$, where b_i are integers, and t_j, called a *term*, is a product of the variables in a subset $S_j \in \{1, 2, \ldots, n\}$, $t_j = \prod_{k \in S_j} x_k$.

2.1.1 Pseudo-Boolean Competition

The Pseudo-Boolean Competition [11] is a special event, part of the International Conference on Theory and Applications of Satisfiability Testing (SAT), organized by the SAT Association. The goal of this evaluation is to assess the state of the art in the field of Pseudo Boolean solvers. Several state-of-the-art solvers have come up from that competition, such as solver NaPS [12], solver Open-WBO [13], solver Minisat+ [14], among others.

2.2 Machine Learning

Machine Learning, as stated in [15], is a sub-field of Artificial Intelligence that enables software to use raw data to extract patters (learn) and improve from experience in order to acquire new knowledge, it aims for computer systems to imitate the way humans learn.

Over the past decade, the field of Machine Learning (ML) has undergone significant development, according to [16]. ML has become a powerful tool for processing and analyzing large volumes of data, as algorithms developed for various ML models aim to uncover hidden patterns within the data. These models learn from a given set of data, called a training set, to create a function f that maps an input instance to a corresponding scalar or vector output, referred to as labels.

The process by which f is learned determines the classification of the ML model:

- **Supervised models:** their learning process relies on ground truth labels, consisting of input instances and their corresponding output labels. Examples of supervised models can be found in Burkart's survey [17].
- **Unsupervised models:** if the model finds patterns independently without access to ground truth labels, it is considered unsupervised, as seen in Alloghani's work [18].
- **Semi-supervised models:** these models combine ground truth labels with pattern analysis of input data to learn, as described in [19].

Supervised and unsupervised machine learning can both perform automatic algorithm selection/configuration, as demonstrated by the work of [20]. However, the focus of this text is on supervised Machine Learning.

ML models can also be classified based on the nature of the output produced by f:

- **Classification model:** its output consists of discrete values used to categorize inputs into different classes.
- **Regression model:** its output corresponds to real values.

The supervised ML algorithms for classification used here are:

Random Forest: The Random Forest (RF) method, as described by Breiman [21], is an ensemble technique [22] that constructs a specified number of Decision Trees (controlled by a parameter $n_{estimators}$).

A Decision Tree [23] is a supervised machine learning algorithm used for classification and regression. The algorithm takes data and creates a tree model which explicitly represents decision making. In this model, leaves represent labels and branches represent conjunctions of features that lead to those labels.

In RF, each tree is trained on a different subset of instances within the training set and proposes a result to compute the output label. The final output label is determined through a consensus scheme that differs depending on whether the model is a regression or classification model. In the case of regression, the consensus is reached by averaging the outputs of all the Decision Trees. In contrast, for classification models, the output label corresponds to the most frequently repeated label (voted) among the Decision Trees' outputs. A representation of the above is presented in Figure 2.1.

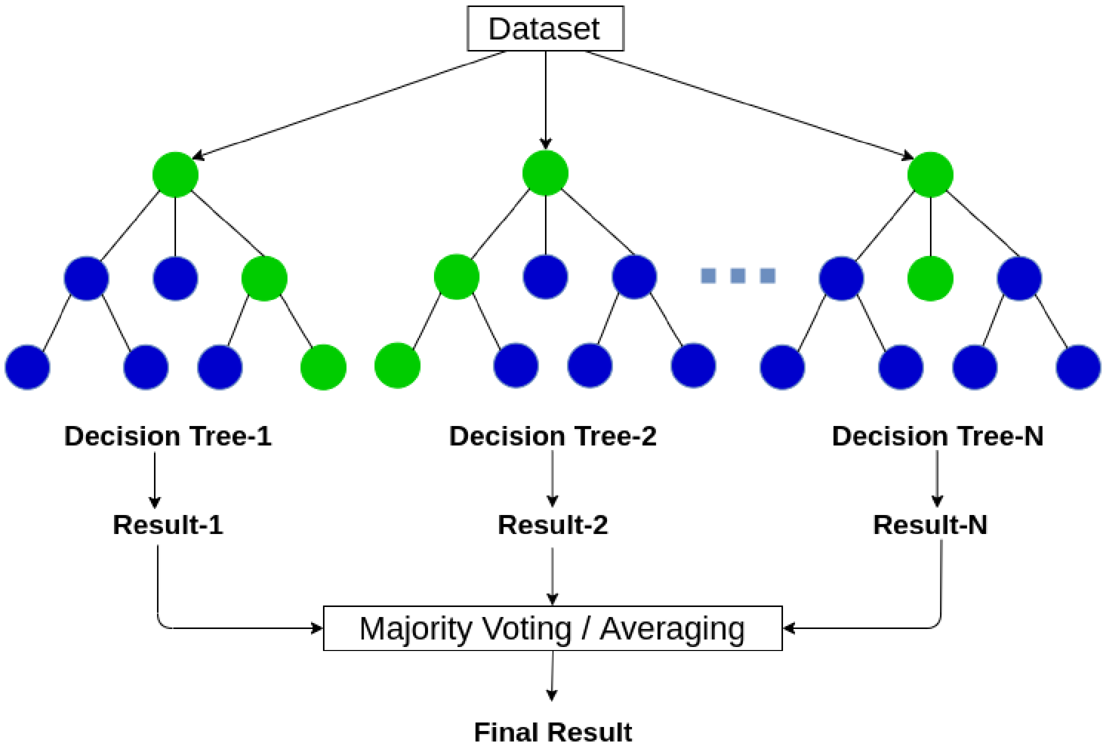


Figure 2.1: Random Forest

***k*-Nearest Neighbors:** The k-Nearest Neighbors (KNN) algorithm, introduced by Fix and Hodges [24], is a Machine Learning (ML) method that determines the output label based on the labels of the k closest training examples to the input point being labeled. The distance between the feature vectors of the input point and the training examples can be calculated using various metrics, but the most commonly used is the Euclidean distance.

For classification tasks, KNN assigns the output label as the most frequently occurring label among the k neighbors. In the case of regression tasks, the output label corresponds to the average of the labels of the k neighbors.

Gradient Boosting: Gradient Boosting (GB) is an ML method, proposed in [25], that builds upon the ideas behind Ada Boost [26]. GB allows for different parameterized loss functions to be defined. The learning process involves consecutively training a parameterized number ($n_{estimators}$) of new "weak´´ models, with each new model being given as input to the next iteration. In a manner similar to gradient descent, a negative gradient is computed based on the past model, which is weighted according to a parameterized scheme (*learning_rate*). A move in the opposite direction is then taken to reduce the loss. This process is repeated to improve the performance of the model.

2.2.1 Deep Learning

Deep Learning is a specific type of ML, which is in turn a type of AI. Figure 2.2 represents the relationship between the different AI disciplines.

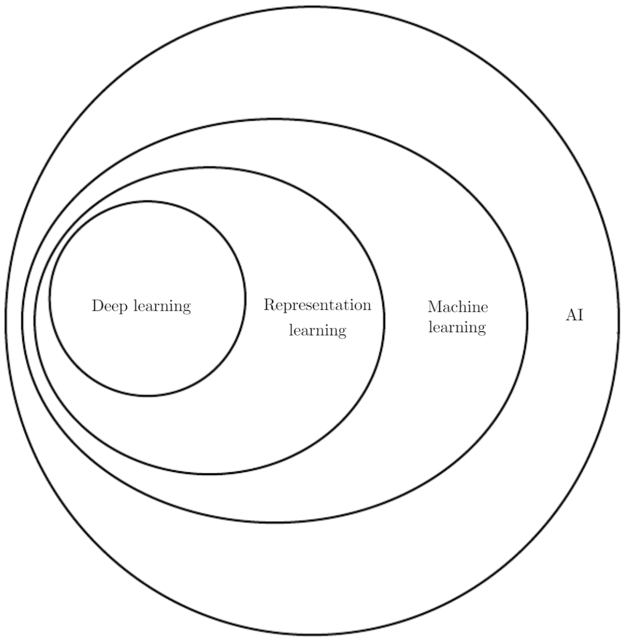


Figure 2.2: Venn Diagram that shows the relationship between Artificial Intelligence, Machine Learning and Deep Learning.

As described in [15], Deep Learning is a Representation Learning method that allows computers learn from experience and represent the world through a hierarchy of concepts, building complex concepts out of simpler ones. That is, DL has multiple levels of representation, obtained from simple non-linear modules that transform the representation from one level into a representation at a higher, more abstract level, starting from the raw input on the first level.

The most typical example of a DL model is the feed-forward deep network or multilayer perceptron, which is a mathematical function that maps an input value to an output value. To go from one layer to the next one, a set of units compute the weighted sum of the inputs received from the previous layer and then apply a non-linear activation function.

One particular type of feed-forward deep network that specializes in processing data with a grid-like topology is the Convolutional Neural Network. Figure 2.3, shows a basic Convolutional Neural Network and its main components:

- **Convolutional Layer:** Convolution is a mathematical operation that works as a filter. In this layer, a convolution kernel filters the feature map (input data) to obtain specific information.
- **Pooling Layer:** Pooling consists of taking input and reducing it to a single value (subsampling).
- **Fully Connected Layer:** Layers in which all the inputs from the previous layer are connected to every activation unit of the next layer.
- **Output Layer:** Last layer, which outputs the results.

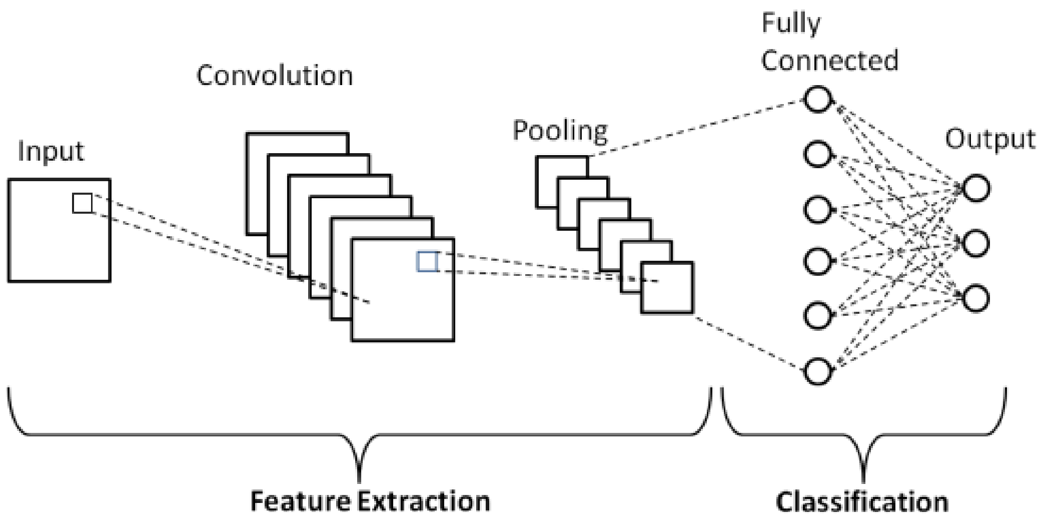


Figure 2.3: Simple Convolutional Neural Network

2.2.2 Evaluation Metrics

There are several evaluation metrics for Machine Learning models. Some metrics to evaluate specifically a classification model are:

- **Accuracy:** number of correct predictions divided by the total number of predictions.
- **Precision:** ratio of true positives and total positives predicted.
- **Recall:** ratio of true positives to all the positives in ground truth.

- **Confusion Matrix:** a tabular summary of the number of correct and incorrect predictions made by a classifier. It is a $N \times N$ table, being N the number of classes. In each cell, the matrix compares the actual target values with what the model predicted.

2.3 Algorithm Selection

The Algorithm Selection problem [1] consists on selecting an algorithm from a portfolio of algorithms for a specific problem, based on its efficiency to solve a given instance. The motivation of such problem is funded on the observation that for practically any computational problem, different instances are best solved using different algorithms. This is specially important for hard problems, for which there is no single algorithm that defines the state-of-the-art, but a set of them that work best in different scenarios.

A specific type of Algorithm Selection is *per-instance Automatic Algorithm Selection (AAS)*: given a problem P, with I a set of instances of P, $A = \{A_1, A_2, ..., A_n\}$ a set of algorithms for P and a general given metric pm that measures the performance of any algorithm $A_j \in A$ for I, AAS consists of a selector S that maps any instance $i \in I$ to an algorithm $S(i) \in A$ such that the overall performance of S on I is optimal according to metric pm.

This predicament has been applied for a variety of problems and studied for several years. An important examination on the subject was made in [27], this work provides an overview on the research in the area, explaining the basis for the subject and successful applications of Algorithm Selection in discrete and continuous optimization.

The state-of-the-art in Algorithm Selection has been boosted by the Algorithm Selection Competitions [28], which has also standardized the way oracles are measured, defining a metric $\hat{m}$ to evaluate the performance of different Algorithm Selectors.

2.3.1 Performance metric for Anytime Automatic Algorithm Selection

In order to measure the performance of a solver $s \in A$ over time, we discretize the time-space into *timesteps*. Let I be a set of instances, and T a set of timesteps. For the instance-timestep pair $(i,t) \in I \times T$, let $o_s(i,t)$ be the objective value of s on instance i, at timestep t. Since the value for $o_s(i,t)$ can greatly vary across instances and timesteps, in order for each data point to weigh equally in a cumulative metric, a normalization function $n(o_s(i,t),i,t)$ is used to map $o_s(i,t)$ values to a uniform range. For PBO, we use the normalization given in (5.2). The cumulative metric m_s we use, also considered in [29], is defined as:

$$m_s = \sum_{(i,t) \in I \times T} n(o_s(i,t),i,t) \tag{2.1}$$

and corresponds to the normalized cumulative performance of solver s across all pairs $(i,t) \in I \times T$. For a meta-solver ms that for each (i,t) instance-timestep pair selects solver $s'_{i,t}$ its cumulative performance metric is defined as:

$$m_{ms} = \sum_{(i,t) \in I \times T} n(o_{s'_{i,t}}(i,t),i,t) \tag{2.2}$$

The evaluation of meta-solvers is usually done in comparison to the performance of Single Best Solver and Virtual Best Solver, defined as:

- **Single Best Solver (SBS):** The single algorithm that performs best (on average) on *all* instances.
- **Virtual Best Solver (VBS):** A solver that makes perfect decisions and matches the best-performing algorithm for each problem instance, without overhead.

For an algorithm selector meta-solver ms, the $\hat{m}_{ms}$ metric was proposed for the Algorithm Selection Competitions [28], using, for each solver s, the performance metric $m_s = \sum_{i \in I} n(o_s(i),i)$. Here we generalize it for Anytime Algorithm Selector meta-solvers as follows.

$$\hat{m}_{ms} = \frac{m_{ms} - m_{VBS}}{m_{SBS} - m_{VBS}} \tag{2.3}$$

where m_{ms} is the normalized cumulative performance of meta-solver ms, m_{VBS} is the normalized cumulative performance of the VBS, and m_{SBS} is the normalized cumulative performance of the SBS.

We observe that:

- The closer $\hat{m}_{ms}$ to 0, the more similar the meta-solver is to the VBS.
- If $\hat{m}_{ms} > 1$, then the meta-solver is worse than the SBS and, hence, is not useful.

3 Related Work

This section provides an overview of related work on Pseudo-Boolean Optimization solvers and on Automatic Algorithm Selection for the SAT and MaxSAT problems, which are special cases of PBO. This section also discusses recent work on Anytime Automatic Algorithm Selection.

3.1 PBO solvers

Most PBO solvers are based on making calls to a program subroutine, based on the Conflict-Driven Clause-Learning (CDCL) algorithm [30], that solves a decision problem on whether the input formula is feasible or not. The optimization problem is translated into a feasibility problem by adding to the constraints the *objective function constraint*, which is an inequality specifying that the objective function is less than or equal (for minimization) to a specified upper bound. This translates the PBO problem into a Boolean Constraint Satisfaction (BCS) problem. Many solvers (e.g. [14, 12, 31]) further encode the BCS problem as a CNF Satisfiable (SAT) formula. Another family of solvers, e.g. [32, 33], implement a Branch & Bound search strategy on a search tree that, at each node, solves the linear relaxation of the problem. In addition to these, a third family of solvers uses local search procedures [34].

Next, we list the PBO solvers considered for inclusion in the portfolio of the meta-solver. These solvers were chosen due to their good performance in the PBO competitions [11].

NaPS: The Nagoya Pseudo-Boolean Solver [12] won the 2016 Pseudo-Boolean Competition in 4 categories. This solver is a MaxSAT solver, based in Minisat+[14]. The main difference between NaPS and other PBO solvers that translate the formula to MaxSAT, is that NaPS uses Binary Decision Diagrams (BDD) to translate the PB constraint to a SAT formula.

OpenWBO: Open-WBO [13] is a weighted partial MaxSAT solver that won second place in two categories in the 2016 PBO Competition. PBO instances are easily translated into weighted partial MaxSAT instances where the PBO's constraints are translated into hard clauses (that must be satisfied), and the objective function is translated into a set of weighted soft clauses.

Open-WBO implements five different search algorithms, of which we only consider two since the other three were dominated by other algorithms in our portfolio. The two search algorithms are:

Linear-su: This algorithm translates the PBO instance to Weighted Partial MaxSAT and uses the LSU search strategy as explained in [35]. We will refer to this option as *OpenWBO-lsu*.

oll: This algorithm translates the PBO instance into Weighted Partial MaxSAT and uses a search strategy similar to WPM1, as explained in [36]. This option will be referred to as *OpenWBO-oll*.

Clasp: Clasp [37] is part of the PosTdam Answer Set Solving COllection, POTASSCO. It is a CDCL solver for Answer Set Programming. Answer Set Programming (ASP) is a form of declarative programming oriented towards difficult (primarily NP-hard) search problems that is more expressive and subsumes PBO. It uses different semantics than other CDCL solvers and, as such, it has superior performance for certain subsets of instances.

LS-PBO: The local search LS-PBO solver achieved good performance in instances from the PB competition. It features a transformation of the objective function into objective constraints, a constraint weighting scheme for the Pseudo-Boolean constraints, and a scoring function to guide the local search [34].

Gurobi: Gurobi [33] is a Mixed Integer Programming (MIP) commercial solver that is able to handle mixed linear, quadratic and second-order cone constraints. When solving a PBO instance, Gurobi uses a Branch & Bound search procedure powered by advanced preprocessing techniques, intelligent generation of cutting planes, specialized heuristics, and parallel processing. Here we used version 9.5.0.

RoundingSAT: The RoundingSAT solver, originally introduced in [38], is a CDCL solver that includes faster propagation routines for PB constraints. Unlike other solvers, it does not translate the PB constraints into a SAT formula but executes conflict analysis directly on the PB constraints. It also allows for incorporating a Linear Programming (LP) solver into its pipeline.

3.2 Algorithm Selection for SAT and MaxSAT

A thorough review of Automatic Algorithm Selection (AAS) is provided in [27]. The performance of AAS meta-solvers has been improved over time due to the influence of Algorithm Selection Competitions [28] and the maintenance and updating of the Algorithm Selection Library (ASLib) [39].

In particular, for SAT and MaxSAT (which are closely related to PBO), many successful meta-solving approaches were proposed in [40], [41], [42],[43], [44], [45], [46]. For example, the SATzilla solver [40] has been quite influential in the SAT community and won several categories in different versions of the SAT competition and SAT evaluation. SATzilla is a Portfolio-Based Algorithm Selection system that chooses the appropriate solver in the portfolio, based on the computation of a number of features from the input instance and other features it collects from probing procedures. For MaxSAT, an improved instance-specific algorithm configuration, also based on different formula and probing features, was proposed in [42]. This solver won the majority of the categories of the MaxSAT competition in 2016.

SATzilla's first version [47] proposed 84 features for characterizing SAT instances, classified into 9 categories: problem size, variable-clause graph, variable graph, clause graph, balance, proximity to Horn formulae, LP-based, CDCL probing and local search probing features. Later on, to build the Satzilla Algorithm Selector [40] 48 of those proposed features were used, excluding the computationally expensive ones. These features are listed below:

- Problem size features:
  - Number of clauses: denoted c.
  - Number of variables: denoted v.
  - Ratio: c/v.
- Variable-clause graph features:
  - Variable nodes degree statistics: mean, variation coefficient, minimum, maximum and entropy.
  - Clause nodes degree statistics: mean, variation coefficient, minimum, maximum and entropy.

- Variable graph features:
  - Nodes degree statistics: mean, variation coefficient, minimum and maximum.
- Balance features:
  - Ratio of positive and negative literals in each clause: mean, variation coefficient and entropy.
  - Ratio of positive and negative occurrences of each variable: mean, variation coefficient, minimum, maximum and entropy.
  - Fraction of binary and ternary clauses.
- Proximity to Horn formula:
  - Number of occurrences in a Horn clause for each variable: mean, variation coefficient, minimum, maximum and entropy.
- CDCL probing features:
  - Number of unit propagations: computed at depths 1, 4, 16, 64 and 256.
  - Search space size estimate: mean depth to contradiction and estimate of the log of number of nodes.
- Local search probing features:
  - Fraction of improvement due to first local minimum: mean for local search algorithms SAPS [48] and GSAT [49].
  - Number of steps to the best local minimum in a run: mean, median, 10^{th} and 90^{th} percentiles for SAPS .
  - Average improvement to best in a run: mean improvement per step to best solution for SAPS.
  - Coefficient of variation of the number of unsatisfied clauses in each local minimum: mean over all runs for SAPS.

In MaxSAT by improved instance-specific algorithm configuration [42], 32 of the standard SAT features were selected, such as the number of variables, number of clauses, proportion of positive to negative literals, and average number of clauses in which a variable appears, among others. In addition, for the specific MaxSAT problem, they also computed the percentage of clauses that are soft and the statistics of the distribution of weights.

A new approach to AAS was proposed in [50], following the philosophy of deep learning models that replace domain-specific features with generic raw data, from which they learn the important features automatically. For this, the authors propose to use as raw data the input text file of any combinatorial problem and convert it to a fixed-size image, that will be used as input for a Convolutional Neural Network (CNN). Specifically, they first create a vector from the input file, replacing each character with its ASCII code, they then reshape the vector as a matrix of $\sqrt{N} \times \sqrt{N}$, where N is the number of total characters in the input text file. Finally, this new "image" of ASCII values is re-scaled to a predefined size, to work with a set of images of the same size. With this input, the selector is a trained CNN multi-label classification model that encodes the input instance and outputs the most promising solver for the instance. This approach is tested with SAT and Constraint Satisfaction (CSP) instances, obtaining a meta-solver that is able to outperform the Single Best Solver, but underperforms in comparison with methods based on domain-specific features. As a baseline for our work, we will use a straightforward adaptation to this approach to anytime scenarios, since no specific work on Anytime Automatic Algorithm Selection for PBO has been proposed until now.

3.3 Anytime Automatic Algorithm Selection

Coping with hard-optimization problems, there is a choice between using efficient heuristics, that work fast but do not necessarily provide high-quality solutions, and exact methods, that may require prohibitive computational time but on average tend to provide better quality solutions. As such, there is a trade-off between running time and the quality of the solution. This trade-off was discussed in [29].

The Anytime Automatic Algorithm Selection (AAAS) framework maps a computational time limit provided by the user to the solver that is predicted to deliver the best-quality solution within this time limit. Meta-solvers based on AAAS were proposed in the literature, for example, for the Knapsack [2] and Traveling Salesperson (TSP) [3] problems.

In [2], the authors propose an AAAS-based meta-solver for the Knapsack problem.

They show that, when taking the time into account, their meta-solver was able to predict the best algorithm among a set of 9 solvers for most of the problem instances. In [3], the authors present a new AAAS meta-heuristic for the TSP, using a model that selects among 5 state-of-the-art TSP algorithms, obtaining results with a precision of 79.8%.

For data collection in the previous two works, the authors utilized an instance generator that emulates the existing public datasets on Knapsack and TSP. For the PBO problem, however, a high-quality instance generator (that generates highly realistic and diverse problem instances) does not currently exist, which prevents us from synthetically augmenting the data used for training and evaluation.

Anytime approaches distinguish themselves from all the approaches in the AS Competition in the sense that the model is trained taking into account the anytime behavior, which is used not only for evaluation. The training includes one data point for each pair $< instance, time >$ which is associated with a label corresponding to the best solver for the pair. Hence, since the setup is different, AS Systems do not fit the anytime scenario.

4 Designing the Machine Learning Oracle for AAAS for PBO

In this section we describe the workflow we carried on for designing and implementing AAAS Machine Learning oracles for PBO. In Subsection 4.1 we elaborate on the characteristics of the instance benchmarks used for our work. In Subsection 4.2 we give details on how we recorded the anytime behavior of the solvers on the chosen portfolio. Subsection 4.3 presents the dataset we used for the training and testing of our meta-solver, and Subsection 4.4 describes the possibilities we considered for characterizing the instances to use as input to our models. Further details on the ML models and algorithms tested can be found in Subsection 4.5. Finally, Subsection 4.6 presents the evaluation of the implemented ML models.

4.1 PBO instances

The dataset of PBO instances was obtained from the 2006, 2007, 2009, 2010, 2011, 2012, 2015, and 2016 Pseudo-Boolean Competitions [11]. These instances were collected from different domain applications such as Bio-informatics, Timetabling, and Hardware verification, among others. The instances with similar origins are organized in benchmarks or "families", and differ from one another in the type of constraints (linear or nonlinear) and the magnitudes of the constraints' coefficients (normal integers or arbitrary precision integers). Our experimental study includes 118 benchmarks, for a total of 3128 *feasible* instances.

4.2 PBO solvers

The solvers used for the construction of the meta-solver, described in detail in Section 3.1, are: NaPS, two variants of OpenWBO, LS-PBO, RoundingSAT, Gurobi and Clasp. We only considered solvers that either have their codes available so as to modify them to record their anytime behavior, or already provide this capability by default.

In order to evaluate the anytime behavior of the solvers, we discretize a time interval of one hour into 500 timesteps following a logarithmic scale, analogous to

[3]. We keep track, whenever the value of the objective function improves, of the corresponding timestep and the updated new solution.

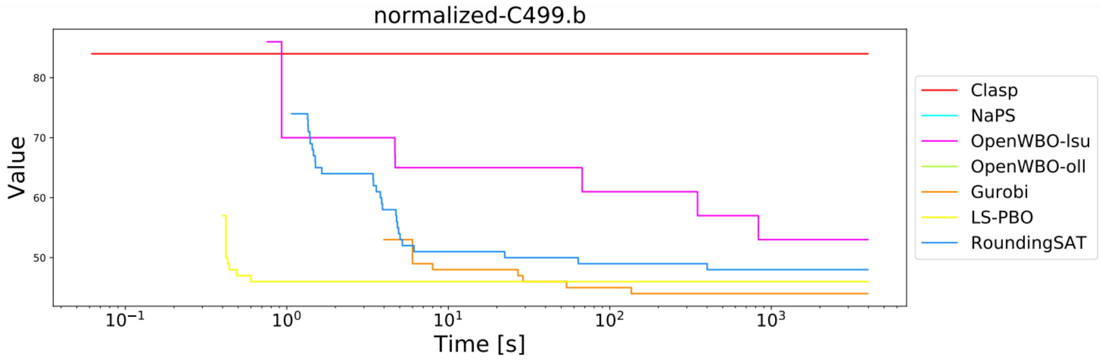


Figure 4.1: Anytime behavior of the solvers for the instance "normalized-C499_b" from Benchmark 106.

The anytime behavior of each solver is recorded as the updated best objective value (incumbent) for each of the 500 timesteps. Figure 4.1 shows the anytime behaviors of the solvers for the instance "normalized-C499_b", which corresponds to a logic-synthesis application. Note that the solver that outputs the solution with the smallest value at a given timestep ts is considered the best option for any specified time limit between the time corresponding to ts and the next timestep $ts + 1$. In Figure 4.1 we observe the change in the best solver across the timeline. Initially, for small time limits, *Clasp* is the best solver. Then, after a few milliseconds, *LS-PBO* becomes the best solver, but it is finally outperformed by *Gurobi*.

A solver is said to *win* an instance-timestep pair if it computes the best-found solution (i.e. a feasible solution with the best objective value) for that instance in that timestep. Ties are broken in favor of the solver that achieved such best incumbent first.

Figure 4.2 presents in a different way the same information from Figure 4.1, indicating the ranking the solvers from 1st (*winner*) to 7th place across time. If a solver does not get a solution, it is not shown. As mentioned before, for the instance "normalized-C499_b" *Clasp* obtained first place for the first few seconds, then being replaced as the best by *LS-PBO* and finally *Gurobi* remains the winner.

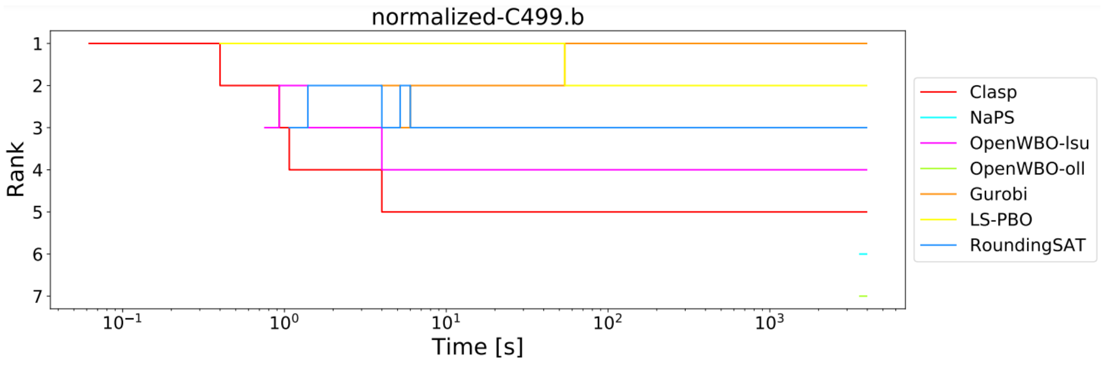


Figure 4.2: Rank of solvers across time for instance "normalized-C499_b" from Benchmark 106.

4.3 Training and testing dataset generation

The dataset we used for building our ML oracles was generated from running all the solvers on the portfolio over all the instances we collected. Figure 4.3 summarizes the number of wins for each solver across the time horizon, for each of the 3128 instances. For each solver, on the horizontal axis, there is a bar consisting of 500 vertical lines, colored from light blue (for the small timesteps) all the way to purple (for the large timesteps). We include a "no solution" entry for instance-timestep pairs where no feasible solution was identified by any of the solvers. Throughout various instances and time intervals, four dominant solvers emerge: Gurobi, RoundingSAT, OpenWBO-oll and LS-PBO. RoundingSAT and LS-PBO exhibit a greater share of wins for smaller timesteps, in comparison with larger ones. Conversely, Gurobi's success rate grows as the timesteps become larger. Although OpenWBO-lsu, NaPS, and Clasp do not command a significant portion of victories, they complement the behavior of the more dominant solvers within the portfolio.

Figure 4.4 summarizes the information on the best solver for each instance and for each timestep. In this figure, it is apparent that the best solver performance depends on the benchmark as well as the timestep. The clear implication is that there is no single best solver for all the instances and timesteps. Since instances belonging to the same family are plotted together, we can also observe that the behavior of the solvers in the portfolio seem to depend on the family of the instances. Most of the instances for which no feasible solution is found by any of the solvers across 500 timesteps belong to benchmarks "mps-v2-20-10" and

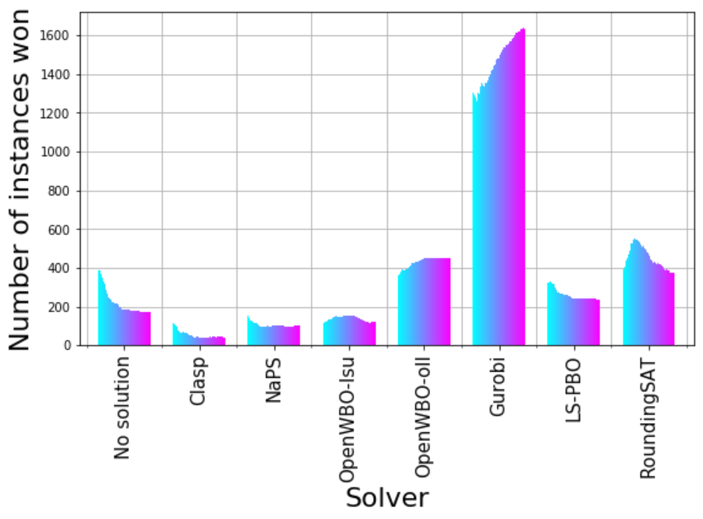


Figure 4.3: Number of wins for each solver across the time horizon (see explanation in text).

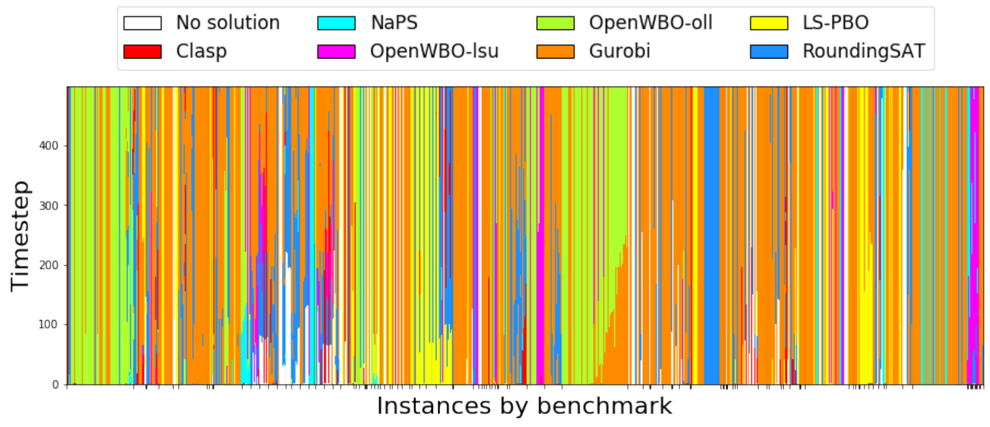


Figure 4.4: Best solver per instance for each timestep. The horizontal axis represents the 3128 instances arranged in the 118 benchmarks. Each vertical bar displays, for one instance, the change in the best solver over the timesteps.

"market-split" from the 2006 version of the PB Competition, benchmarks "opb-trendy" and "opb-paranoid" from the PB Competition 2010 and PB Competition 2012. These benchmarks correspond to the competition's category called BIGINT, which means that the coefficients can be arbitrary precision (i.e. not bounded) integer numbers.

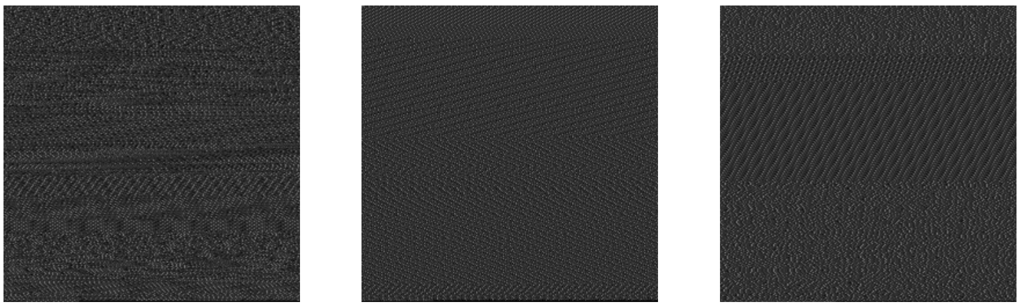
To train and test the ML models, the instances were partitioned into training and testing sets. This was done by partitioning the instances of the 118 benchmarks into 70% for training and 30% for testing, resulting in 2054 instances for training and 1074 instances for testing. The partition was done by randomly picking the instances from each benchmark, maintaining the same ratio.

4.4 Characterization and labeling

4.4.1 Loreggia's representation

An adaptation of [50] is used as a baseline for our work. Loreggia's representation was further explained previously.

For this project, each one of the 3128 PBO instances was transformed into a $\sqrt{N} \times \sqrt{N}$ matrix containing the corresponding ASCII values, where N is the number of total characters in each original input text file. Then, this gray-scale image was re-scaled into 256×256 size. Figure 4.5 shows three different instances transformed to images by implementing the previous procedure.



(a) Benchmark 1, instance "normalized-single-obj-f4-DataDisplay"

(b) Benchmark 106, instance "normalized-C499_b"

(c) Benchmark 118, instance "normalized-30_30_4"

Figure 4.5: Instances represented as images.

4.4.2 Domain-specific features for PBO

Based on previous work on SAT [40] and MaxSAT [42], we defined a set of features for our problem. For this selection, considering the anytime nature of our meta-solver, we focus on informative fast-to-compute features. Since our problem has its own characteristics, we also test some other features that are specific to non-linear PBO instances. Therefore, here we use the following 8 sets of *domain-specific features*:

- **Number of constraints:** Number of constraints in the instance. An equivalent feature for SAT and MaxSAT was used by [40] and [42].
- **Number of variables:** Number of Boolean variables present in the instance. This feature was used by [40] and [42].
- **Linearity:** Identifies if the formula contains non-linear constraints. No similar feature was proposed before.
- **Distribution of the number of terms per constraint:** We partition the constraints into four classes according to the number of terms they contain: 1, 2, 3, or 4 or more terms. The four percentages of the number of constraints in each class out of the total are four features in this set. A similar set of features was used by [40].
- **Term degree:** Percentage of unary, binary, ternary and quaternary-or-more terms. This is the number of terms with 1, 2, 3, or 4 or more variables out of the total number of terms in the instance. No similar feature was proposed before.
- **Objective function size:** Percentage of terms that are present in the objective function, out of the total number of terms. No similar feature was proposed before.
- **Positive terms (Constraints):** Percentage of positive terms in the constraints. An equivalent feature was used by [40] and [42].
- **Positive terms (Objective):** Percentage of positive terms in the objective function. Inspired by the above, we extend the feature for the objective function.

4.4.3 Ground truth labeling for the models

As mentioned in Subsection 4.2, 7 different solvers were used to create the meta-solver. We then use the solvers as labels to identify which solver is the best for a given instance-time pair. We include a “no solution” label to indicate the cases where no solver obtains a feasible solution at a given timestep. This can be useful, especially, for hard instances where solvers require a long time to compute the first feasible solution. Also, in practice, a “no solution” label may indicate to the user the need for allocating more computational resources for solving the instance.

Nevertheless, this feature of our model was not used for the final evaluation of the meta-solver, since this kind of prediction is not usually considered in the $\hat{m}$ metric.

4.5 Machine Learning Models

A generic ML approach that uses Convolutional Neural Networks (CNN) to design an automatic algorithm selector that is able to work without the need of handcrafted domain-specific features is proposed in [50]. We test variants of this method against variants of Random Forest, Gradient Boosting and k-Nearest Neighbors based on the domain-specific characterization of Subsection 4.4.2.

Despite the potential misalignment between accuracy and $\hat{m}$ metrics, we have chosen to train multi-label classification models that prioritize accuracy. This decision stems from the requirement of having simple and fast ML models for anytime scenarios. By employing a multi-label classification model, our approach offers the advantage of considering all solvers simultaneously and making a single call to the oracle to make a decision. This stands in contrast to more complex Algorithm Selection Systems that typically involve multiple ML oracles, such as multiple binary classification models for each pair of alternatives or regression models for individual solvers. The use of such complex systems would result in prohibitively long prediction times, which are not suitable for our anytime scenario.

4.5.1 CNN for Loreggia's representation

As a baseline method, we characterize the instances as images, following the proposal of [50] (described in Subsection 4.4.1). These images are given as input to a Convolutional Neural Network (CNN), which outputs the best solver for every timestep. Hence, we adapt the method to handle anytime scenarios by learning a label for each of the possible 500 timesteps. That way, when a prediction for a particular time is needed, we have to inspect the output of the network that corresponds to the closest (smaller or equal) timestep output.

For the implementation of the CNN, three different architectures were tested: **VGG16** [51], **AlexNet** [52] and **GoogLeNet** [53]. These architectures are shown in Figures 4.6, 4.7 and 4.8, respectively.

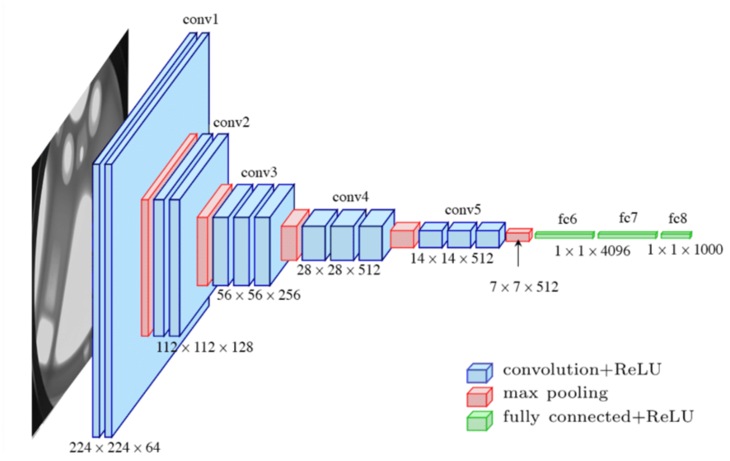


Figure 4.6: VGG16 Architecture

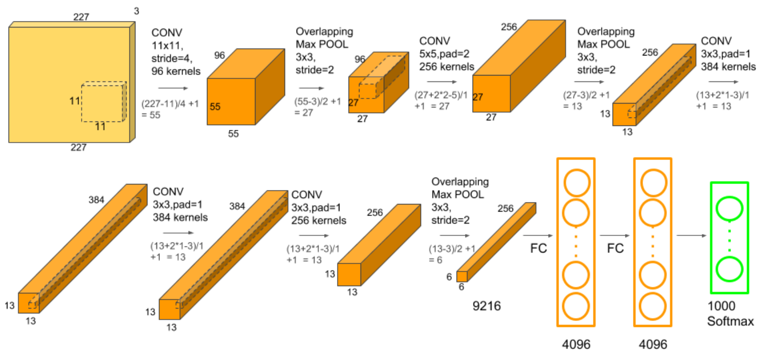


Figure 4.7: AlexNet Architecture

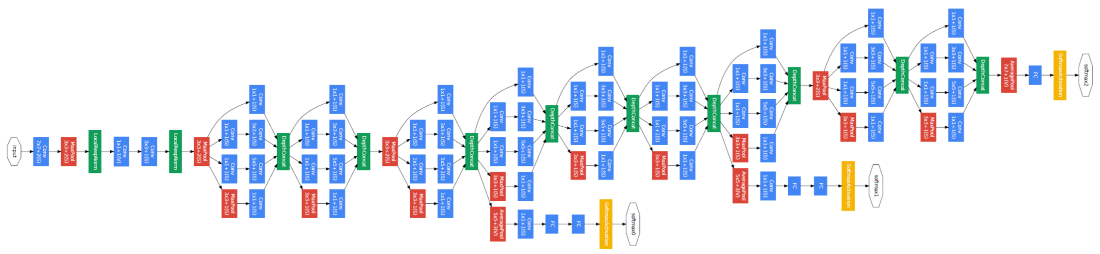


Figure 4.8: GoogLeNet Architecture. Different colors denotes different layers: Convolutional layers are colored blue, pooling layers are shown in red, Softmax activation function is colored yellow and green corresponds to others.

4.5.2 Models using domain-specific features

In our study, we conducted experiments using three ML algorithms, as outlined in Subsection 2.2. These algorithms were implemented using the Scikit-learn library [54]. We utilized various subsets of the 14 domain-specific features described in Subsection 4.4.2. Additionally, hyperparameter tuning was performed to determine the optimal architecture for each model, as well as weighting strategies to compensate for the natural bias induced by the dominating classes of the portfolio.

RF_basic: The Random Forest classifier uses only two features: the number of constraints and the number of variables. The hyperparameters used were: `n_estimators = 100`, `max_features = "sqrt"`, `criterion = "gini"`.

RF_nonlinear: The Random Forest classifier uses all the 8 sets of domain-specific features. The hyperparameters used were: `n_estimators = 100`, `max_features = "sqrt"`, `criterion="gini"`.

RF_linear: The Random Forest classifier uses features of the linearized version of the PBO instance. Therefore, the features related to non-linearity and term degree are redundant and removed. The hyperparameters used were: `n_estimators=100`, `max_features = "sqrt"`, `criterion = "gini"`.

GB_basic: The Gradient Boosting classifier uses only two features: the number of constraints and the number of variables. The hyperparameters used were: `n_estimators = 100`, `learning_rate = 0.5`, `max_depth = 3`, `max_features="sqrt"`.

GB_nonlinear: The Gradient Boosting classifier uses all the 8 sets of domain-specific features. The hyperparameters used were: `n_estimators = 100`, `learning_rate = 0.25`, `max_depth = 3`, `max_features = "sqrt"`.

GB_linear: The Gradient Boosting classifier uses features of the linearized version of the PBO instance. Therefore, the features related to non-linearity and term degree are redundant and removed. The hyperparameters used were: `n_estimators = 100`, `learning_rate = 0.1`, `max_depth = 3`, `max_features = "sqrt"`.

KNN_basic: The $k-$Nearest Neighbors classifier uses only two features: the number of constraints and the number of variables. The hyperparameter used was: `n_neighbors = 13`.

KNN_nonlinear: The $k-$Nearest Neighbors classifier uses all the 8 sets of domain-specific features. The hyperparameter used was: `n_neighbors = 21`.

KNN_linear: The $k-$Nearest Neighbors classifier uses features of the linearized version of the PBO instance. Therefore, the features related to non-linearity and term degree are redundant and removed. The hyperparameter used was: `n_neighbors = 21`.

For all the variants, the set of features is augmented with the feature of timestep, which increments the number of the model's input features in one. We note that for the `linear` versions, we first need to linearize the input instance in order to compute the purely linear features, which is not the case for the `nonlinear` versions, for which we don't incur in such overhead for the computing of the non-linear features.

4.6 Evaluation

ML Oracle	**Accuracy**	**Metric $\hat{m}$**
Loreggia's w VGG	0.4712	1.00
Loreggia's w AlexNet	0.5775	0.7157
Loreggia's w GoogLeNet	0.4931	1.3565
RF_basic	0.6580	0.7108
RF_nonlinear	0.7106	**0.5250**
RF_linear	**0.7159**	0.5729
GB_basic	0.6379	0.8252
GB_nonlinear	0.7046	0.6198
GB_linear	**0.7184**	0.6501
KNN_basic	0.6225	0.8481
KNN_nonlinear	0.6407	0.8589
KNN_linear	0.6621	0.7971

Table 4.1: Accuracy and $\hat{m}$ values for different Machine Learning Models and subsets of characteristics.

Table 4.1 compares the accuracy and $\hat{m}$ values (calculated as described in 5.1) of the different combinations of ML models and subsets of features as explained in the previous subsection. As can be seen, GB_linear provides the best performance in accuracy and RF_nonlinear the best performance in the $\hat{m}$ metric.

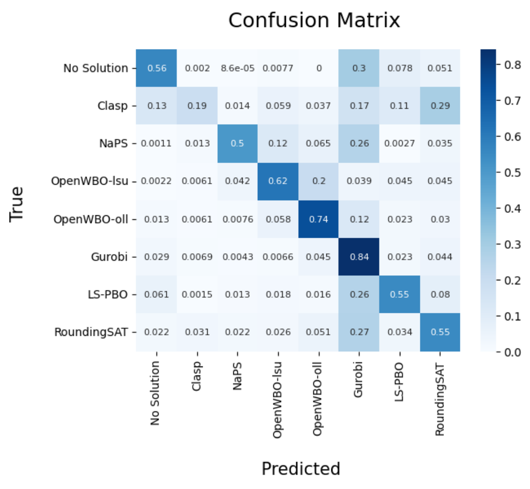
The ML methods relying on domain-specific features, regardless of the subsets of features considered, outperform in accuracy the deep-learning networks based on the generic representation of [50], which we take as a baseline. The Deep Learning Network that provides the best accuracy and $\hat{m}$ values is the one based on the AlexNet architecture.

Although not perfect, Table 4.1 demonstrates an inverse correlation relation between the accuracy and $\hat{m}$ metrics. This is with the noticeable exception of the best-performing Deep Learning Network, AlexNet, which, in comparison with GB_basic and all KNN models, with a worse accuracy value achieves a better $\hat{m}$ score.

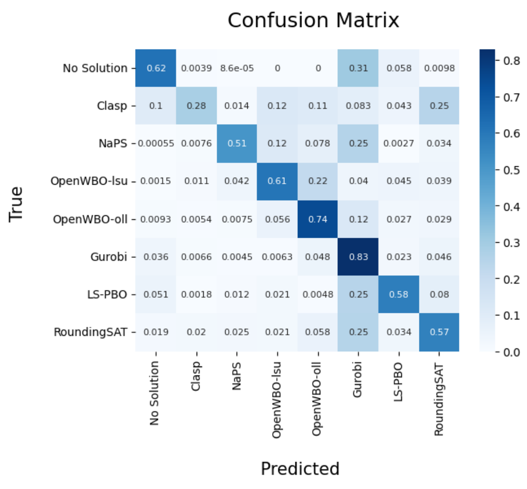
It is important to note that this table does not account for the overhead associated with computing the features or the time required for the models to generate predictions. These factors can significantly impact the practical performance of using these models to build an anytime meta-solver. Therefore, we will further analyze and present results considering the four best-performing combinations of models and sets of features: RF_nonlinear, RF_linear, GB_nonlinear, and GB_linear.

Figure 4.9 depicts the Confusion Matrix for our best models. It is evident that all matrices demonstrate a similar pattern in the behavior of the models. Generally, it can be inferred that the classes were learnable, except for the Clasp class, which has a smaller representation in the dataset. It is natural for these models that the higher the class representation in the dataset, the higher the accuracy for that class. Similarly, a higher class representation increases the likelihood of the model over-predicting that class. To mitigate this issue, we implemented methods to address the bias introduced by the dominant classes. These methods involved assigning, during the training of the models, bigger weights to miss-classifications of less frequent classes, compared to the more dominant ones.

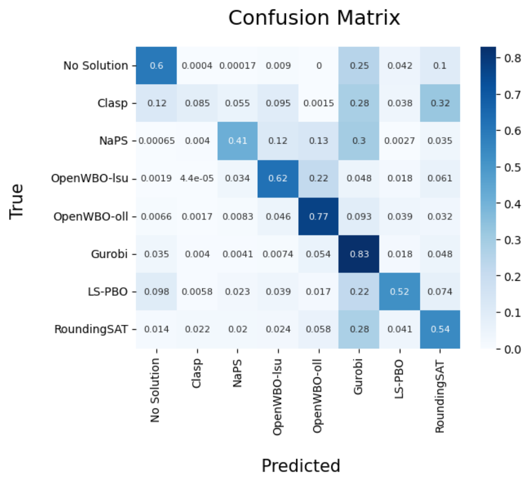
The output of Random Forest and Gradient Boosting includes the MDI (Mean Decrease in Impurity) for each feature, which is a proxy for feature importance. The higher the value of the MDI, the more important the feature is. Figure 4.10 shows the MDI values for the 10 features of the linearized instances for both models. It is evident that the importance of features varies depending on the model used. In particular, for the GB model, the timestep feature appears to



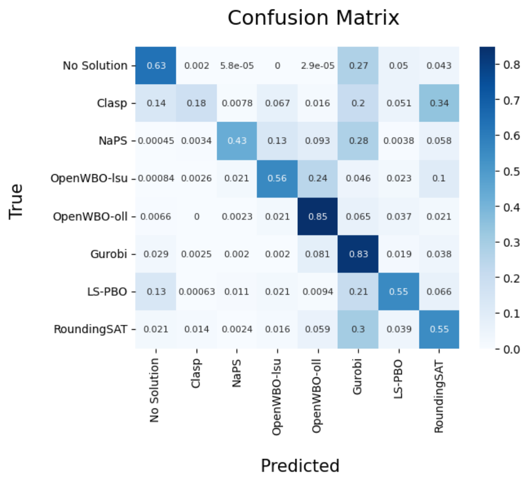
(a) RF_nonlinear



(b) RF_linear



(c) GB_nonlinear

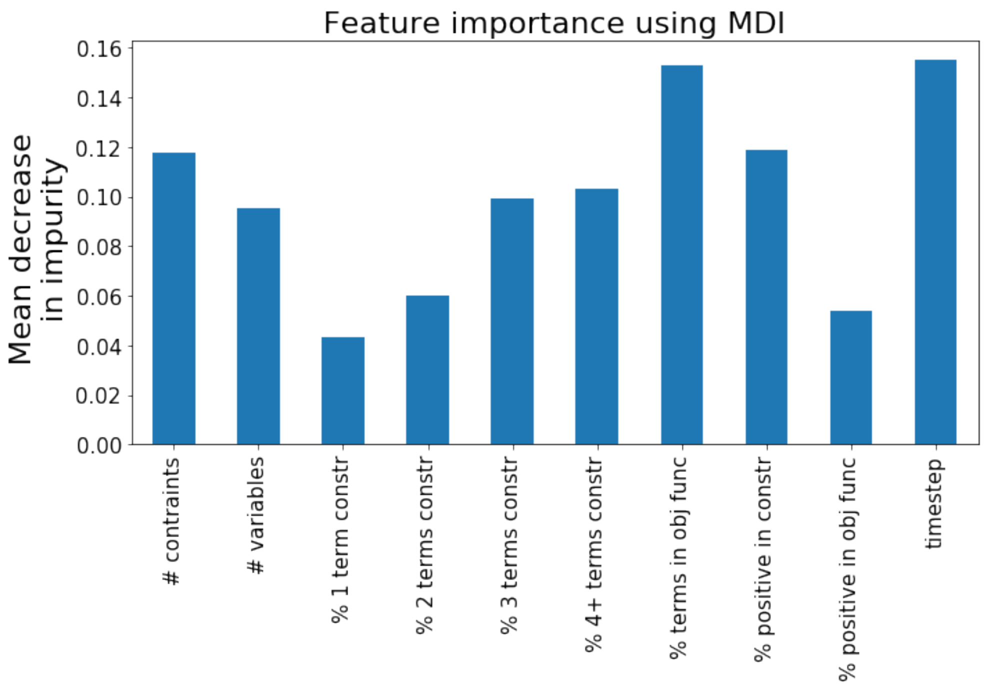


(d) GB_linear

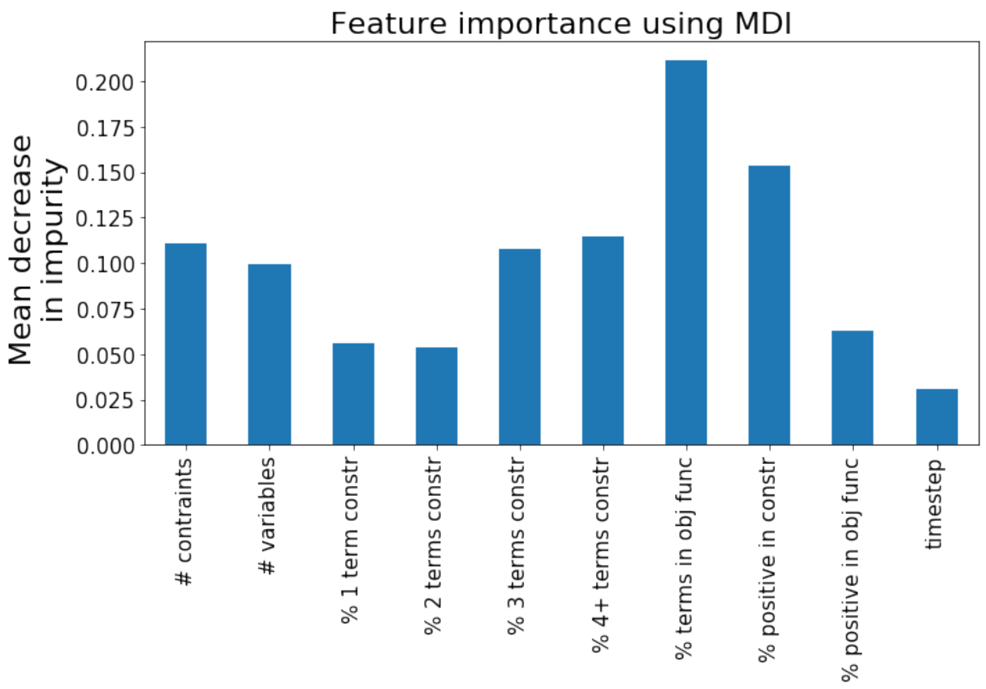
Figure 4.9: Confusion matrices of PBO meta-solvers based on RF_nonlinear, RF_linear, GB_nonlinear and GB_linear.

be less significant compared to other features related to the composition of the PBO formula in making predictions. On the other hand, the RF model heavily relies on the timestep feature to make recommendations. Both models consider the *percentage of terms that are present in the objective function*, first proposed here, as a very important feature.

Figure 4.11 provides a visual representation of how our two most accurate models, RF_linear and GB_linear, behave. By examining this figure in conjunction with Figures 4.9, 4.10 and Table 4.1, we can draw some conclusions. Although GB_linear achieves higher accuracy, this is primarily due to the bias introduced by the four most dominant classes, for which it exhibits superior performance



(a) Feature importance for the RF_linear model.



(b) Feature importance for the GB_linear model.

Figure 4.10

compared to the RF model. Furthermore, it is evident that the GB model places less emphasis on the anytime behavior of the solvers and tends to select the

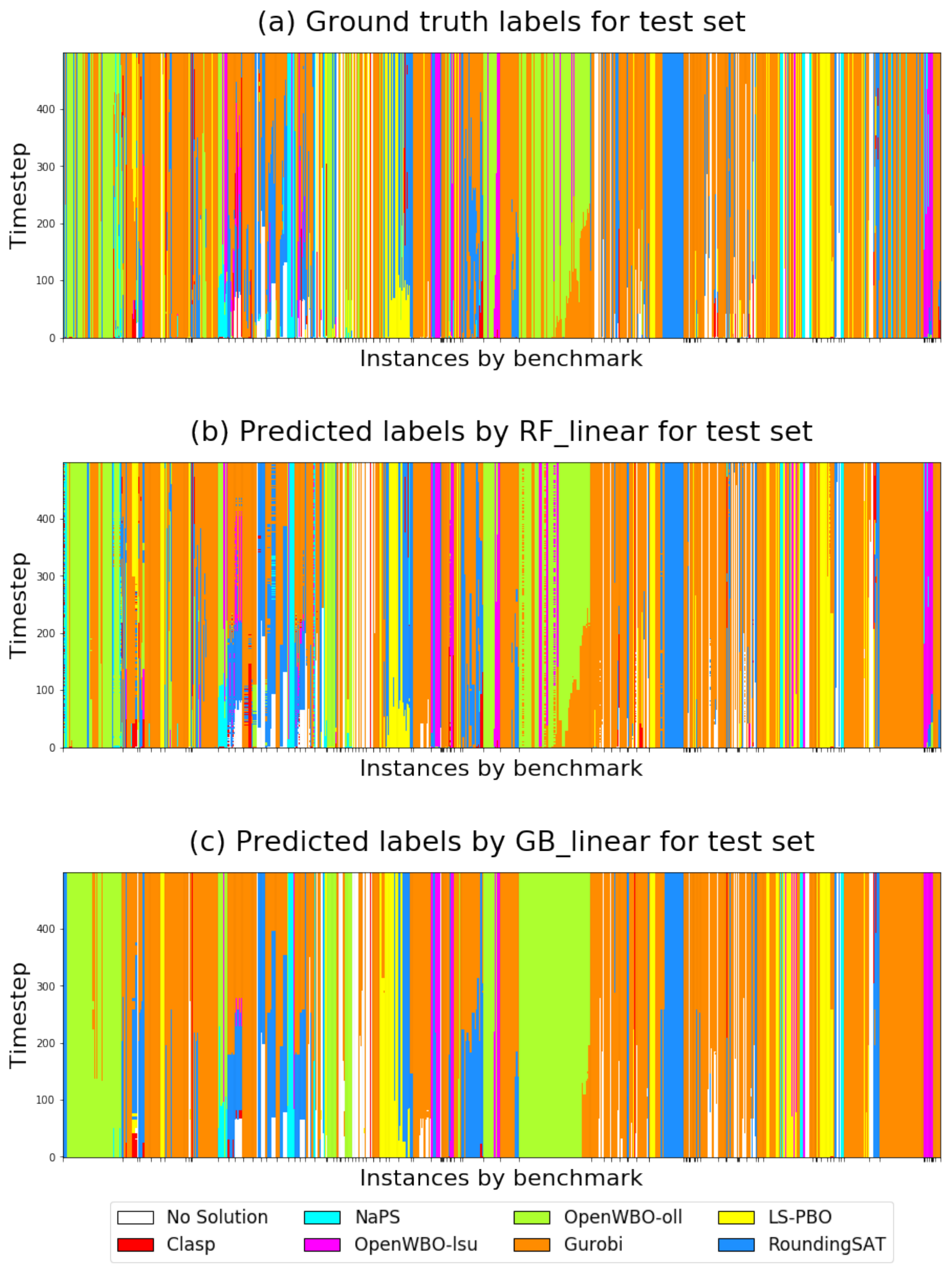


Figure 4.11: Comparison of ground truth labels with predicted labels of the RF_linear and GB_linear for the test set.

same solver for a given instance, regardless of the timestep. In contrast, the RF model demonstrates a more varied selection of solvers based on the timestep. This observation aligns with the analysis of feature importance in the GB and RF models, providing support for this observation.

5 Results

5.1 Meta-solver's performance

In this Section, we present the results of the performance of our meta-solver based on the four best models explained in the previous Section: RF_nonlinear, RF_linear, GB_nonlinear and GB_linear ML oracles. As explained in Subsection 2.3, the best way to measure the performance of a meta-solver is through the $\hat{m}_{ms}$ metric. For our particular case, $\hat{m}_{ms}$ was calculated considering Gurobi as the SBS, for all timesteps. For computing the cumulative score m_s, for each solver s, as defined in Equation 2.1, the normalization function $n(o_s(i,t),i,t)$ of $o_s(i,t)$ (the objective value of s on instance i at timestep t), is defined so that its co-domain is in the range $[0,1] \cup \{2\}$. For this, we compute $o_{min}(i)$, the minimum feasible value (in many cases the optimal value) of the objective function for the instance i and $o_{max}(i)$, the maximum feasible value of the objective function for the instance i, both considering all the feasible solutions found by all the solvers. The by-default normalization of a given value $o_s(i,t)$ is computed as follows:

$$n'(o_s(i,t),i,t) = \frac{o_s(i,t) - o_{min}(i)}{o_{max}(i) - o_{min}(i)} \tag{5.1}$$

This by-default normalization is not always well defined and some special cases have to be considered. Considering such cases, we formally define $n(o_s(i,t),i,t)$ as:

$$\begin{cases} 0 & \text{if } o_s(i,t) = o_{min}(i) = o_{max}(i) \\ 2 & \text{if } o_s(i,t) \text{ is undefined} \\ & \text{but } o_{max}(i) \text{ is defined} \\ n'(o_s(i,t),i,t) & \text{otherwise} \end{cases} \tag{5.2}$$

As $\hat{m}_{ms}$ compares the meta-solver with SBS, and such solver is not able to use the "no solution" label in its favor, for our meta-solver's evaluation, we decide to only consider instance-time pairs for which $o_{max}(i,t)$ is defined (i.e. we don't consider the instance-timesteps pairs that correspond to white points on Fig 4.11(a)).

One issue to consider concerning the computational time limit is whether to include the feature computation and prediction times needed by the ML oracle in addition to running the solver. The prediction requires input preparation for the instance (computing the features for the model, constructing the image for CNN, and linearizing the instances for the models with only linear features) and running the ML model. If we consider the prediction time, there is less time to run the solver and, consequently, the value of our performance metric $\hat{m}_{ms}$ goes up. We report on the performance of RF_nonlinear, RF_linear, GB_nonlinear, and GB_linear for both cases, when prediction time "overhead" is included or not, for each timestep, in Table 5.1.

Model	$\hat{m}_{ms}$ (no)	$\hat{m}_{ms}$ (o)
RF_nonlinear	**0.5250**	**0.5318**
RF_linear	0.5729	0.6042
GB_nonlinear	0.6198	0.6270
GB_linear	0.6501	0.6712

Table 5.1: $\hat{m}_{ms}$ calculated with no overhead time (no) and $\hat{m}_{ms}$ calculated considering the overhead time (o) for the PBO meta-solvers based on RF_nonlinear, RF_linear, GB_nonlinear and GB_linear. Lower $\hat{m}_{ms}$ values are better.

It is evident that while the RF_linear model exhibits the better accuracy value, it is the RF_nonlinear one that achieves the best $\hat{m}$ value. The difference in the $\hat{m}$ scores between these two models grows even bigger when considering the overhead. This is primarily due to the time impact of linearizing the instances before computing the features in the RF_linear case. This also happens with the GB models, although the $\hat{m}$ values are less competitive than the ones achieved by the RF models.

Figure 5.1 provides insight into how the $\hat{m}$ value changes for each timestep for the best-performing RF and GB models, taking into account the overhead. As anticipated, we observe that the overhead has a negative impact on the $\hat{m}$ value during the initial timesteps. RF consistently demonstrates better $\hat{m}$ values than GB, suggesting that RF learns more effectively from the anytime data.

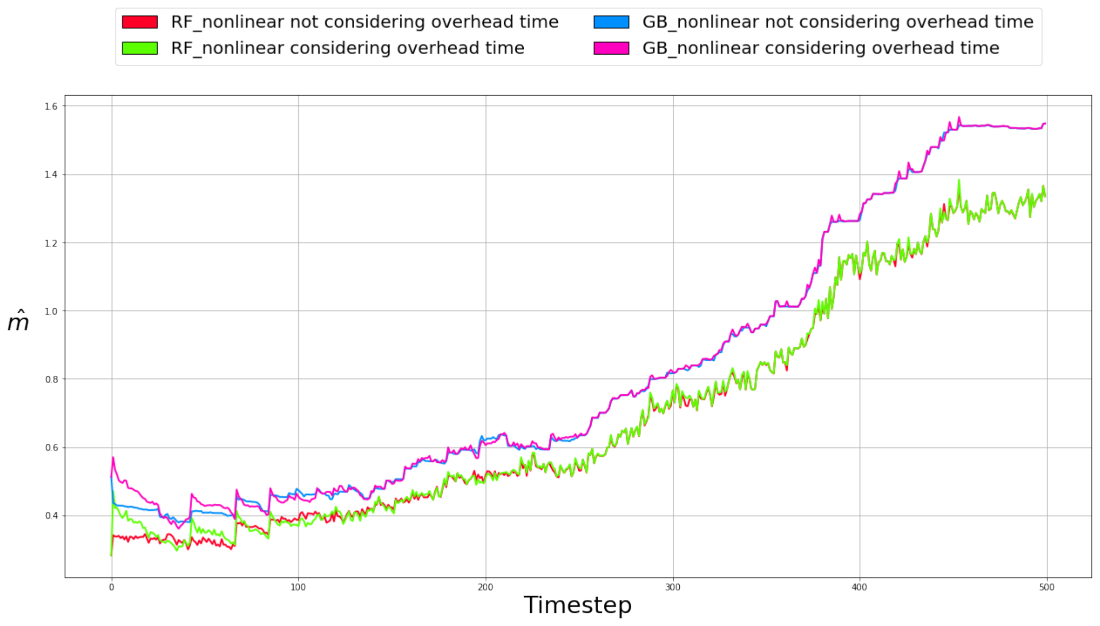


Figure 5.1: Values of $\hat{m}$ across timesteps for GB_nonlinear and RF_nonlinear. Lower $\hat{m}$ values are better.

5.2 Comparing the meta-solver with the Single Best Solver

Recall that Gurobi is the single best solver (SBS). Here we elaborate on where the gain from the meta-solver (MS) comes from. To do this, we consider all *test* instance-timestep pairs for which a feasible solution is known, $502,258$ in total. In Figure 5.2, we compare, over this set of instances, the number of instances for which each of the solvers (SBS and the MS based on the RF_nonlinear model) report the best-found incumbent solution (red), a feasible worse-than-the-best-found incumbent solution (orange) or for which no incumbent solution has been computed yet (light blue). It is apparent from this figure that the meta-solver MS provides a significant improvement over the use of the SBS by, for many instance-timestep pairs, selecting alternative solvers that are either able to find better solutions than the SBS or that are able to compute an incumbent solution when the SBS is not. This justifies the use of our meta-solver for the PBO problem in practice.

Overall, from $502,258$ test instance-timestep pairs, Gurobi is able to find $296,103$ best-found solutions, $142,824$ non-best-found incumbent solutions, and is unable to find feasible solutions for $63,331$ instance-timestep pairs. The meta-solver finds

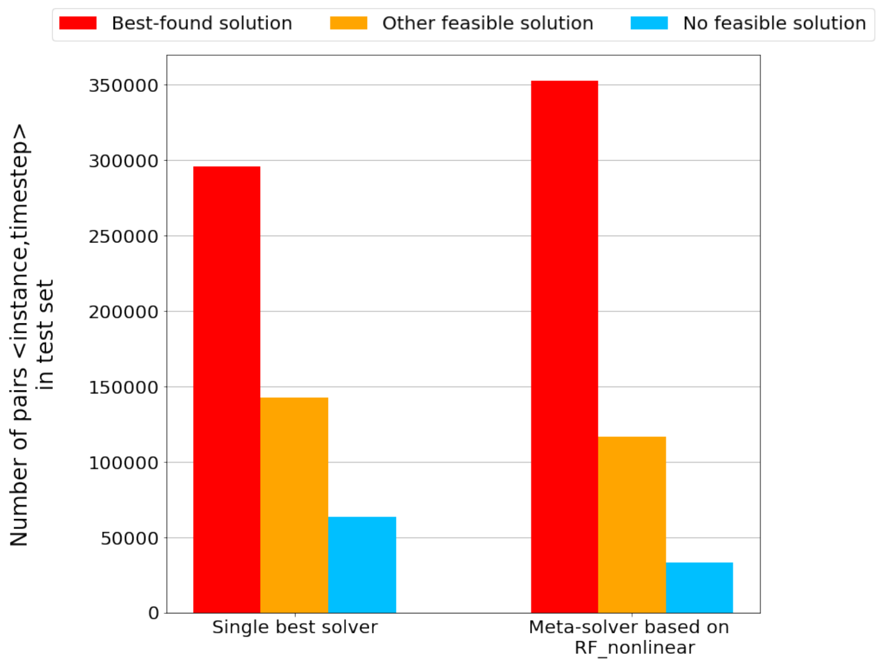


Figure 5.2: Number of feasible instance-timestep pairs of the test set where the SBS (Gurobi), and the meta-solvers based on RF_nonlinear and GB_nonlinear find the optimal solution, feasible non-optimal solution or no feasible solution.

352, 462 best-found solutions, 116, 717 non-best-found solutions and is unable to find feasible solutions for 33, 079 instance-timestep pairs. As we can see, the meta-solver improves Gurobi's performance by achieving the best-found solution in around 19% more instance-timestep pairs and diminishing in up to 47.7% the number of instance-timestep pairs for which a feasible solution is not yet found.

6 Conclusions and Future Work

We propose here an Anytime meta-solver for the Pseudo-Boolean Optimization problem. Our meta-solver is able to predict and execute a solver that, among 7 different solvers, performs best for a given problem instance and a specified time limit. Our results show that our meta-solver (based on any of the two best models) significantly outperforms all individual solvers, while it also identifies when feasibility cannot be achieved for a given instance.

Based on the above, the objectives of our work were accomplished. Likewise, the implementation of the ML based meta-solver and the results showing that the meta-solvers based on RF_nonlinear, RF_linear, GB_nonlinear and GB_linear outperform the SBS in regards to the $\hat{m}$ metric, prove that the hypothesis stated at the beginning of our work is supported.

Among all available features, our best ML models show that one of our proposed features, the *percentage of terms that are present in the objective function*, is a very important one to characterize the instances. Furthermore, the RF models determined that the computational time limit is another important feature, supporting the interest in anytime scenarios.

A logical next step is to propose ways of adapting Anytime Algorithm Selection Problems to the scenarios of the Algorithm Selection Library [39], which, currently, are not *anytime*. We will use this as an efficient way of sharing our data.

For future work, we plan to explore the application of Graph Neural Networks as a potential ML oracle. This type of Neural Network has recently been shown to perform well on data that can be represented as a graph, which is the case for PBO. We also plan to explore a two-layer meta-solver approach, where the first layer selects a solver from a portfolio while the second layer chooses the most suitable set of parameters for the chosen solver.

References

[1] J. R. Rice, "The algorithm selection problem," in *Advances in computers*, vol. 15, pp. 65–118, Elsevier, 1976.

[2] I. I. Huerta, D. A. Neira, D. A. Ortega, V. Varas, J. Godoy, and R. Asín-Achá, "Anytime automatic algorithm selection for knapsack," *Expert Systems with Applications*, vol. 158, p. 113613, 2020.

[3] I. I. Huerta, D. A. Neira, D. A. Ortega, V. Varas, J. Godoy, and R. Asín-Achá, "Improving the state-of-the-art in the traveling salesman problem: An anytime automatic algorithm selection," *Expert Systems with Applications*, vol. 187, p. 115948, 2022.

[4] E. Boros and P. L. Hammer, "Pseudo-boolean optimization," *Discrete applied mathematics*, vol. 123, no. 1-3, pp. 155–225, 2002.

[5] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*, vol. 185. IOS press, 2009.

[6] V. M. Manquinho and J. Marques-Silva, "Effective lower bounding techniques for pseudo-boolean optimization [eda applications]," in *Design, Automation and Test in Europe*, pp. 660–665, IEEE, 2005.

[7] R. Wille, H. Zhang, and R. Drechsler, "Atpg for reversible circuits using simulation, boolean satisfiability, and pseudo boolean optimization," in *2011 IEEE Computer Society Annual Symposium on VLSI*, pp. 120–125, IEEE, 2011.

[8] P. Trezentos, I. Lynce, and A. L. Oliveira, "Apt-pbo: solving the software dependency problem using pseudo-boolean optimization," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pp. 427–436, 2010.

[9] R. A. Achá, R. López, S. Hagedorn, and J. A. Baier, "Multi-agent path finding: A new boolean encoding," *Journal of Artificial Intelligence Research*, vol. 75, pp. 323–350, 2022.

[10] R. Asín Achá and R. Nieuwenhuis, "Curriculum-based course timetabling with sat and maxsat," *Annals of Operations Research*, vol. 218, no. 1, pp. 71–91, 2014.

[11] V. Manquinho, O. Roussel, and M. Deters, "Pseudo-boolean competition 2010," *See http://www. cril. univ-artois. fr/PB10*, 2011.

[12] M. Sakai and H. Nabeshima, "Construction of an robdd for a pb-constraint in band form and related techniques for pb-solvers," *IEICE TRANSACTIONS on Information and Systems*, vol. 98, no. 6, pp. 1121–1127, 2015.

[13] R. Martins, V. Manquinho, and I. Lynce, "Open-wbo: A modular

maxsat solver," in *International Conference on Theory and Applications of Satisfiability Testing*, pp. 438–445, Springer, 2014.

[14] N. Sörensson, "Minisat 2.2 and minisat++ 1.1," *A short description in SAT Race*, vol. 2010, 2010.

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[16] E. Alpaydin, *Machine learning.* MIT Press, 2021.

[17] N. Burkart and M. F. Huber, "A survey on the explainability of supervised machine learning," *Journal of Artificial Intelligence Research*, vol. 70, pp. 245–317, 2021.

[18] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain, and A. J. Aljaaf, "A systematic review on supervised and unsupervised machine learning algorithms for data science," *Supervised and unsupervised learning for data science*, pp. 3–21, 2020.

[19] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[20] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, "Isac–instance-specific algorithm configuration," in *ECAI 2010*, pp. 751–756, IOS Press, 2010.

[21] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[22] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[23] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[24] E. Fix and J. L. Hodges, "Discriminatory analysis: nonparametric discrimination: consistency properties," *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, pp. 32–39, 1991.

[25] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[26] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[27] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary computation*, vol. 27, no. 1, pp. 3–45, 2019.

[28] M. Lindauer, J. N. van Rijn, and L. Kotthoff, "The algorithm selection

competitions 2015 and 2017," *Artificial Intelligence*, vol. 272, pp. 86–100, 2019.

[29] R. Amadini and P. J. Stuckey, "Sequential time splitting and bounds communication for a portfolio of optimization solvers," in *Principles and Practice of Constraint Programming: 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings 20*, pp. 108–124, Springer, 2014.

[30] A. Biere, M. Heule, H. van Maaren, and T. Walsh, "Conflict-driven clause learning sat solvers," *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications*, pp. 131–153, 2009.

[31] R. Martins, S. Joshi, V. Manquinho, and I. Lynce, "Incremental cardinality constraints for maxsat," in *International Conference on Principles and Practice of Constraint Programming*, pp. 531–548, Springer, 2014.

[32] L. A. Wolsey and G. L. Nemhauser, *Integer and combinatorial optimization*, vol. 55. John Wiley & Sons, 1999.

[33] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023.

[34] Z. Lei, S. Cai, C. Luo, and H. Hoos, "Efficient local search for pseudo boolean optimization," in *Theory and Applications of Satisfiability Testing–SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings 24*, pp. 332–348, Springer, 2021.

[35] M. Koshimura, T. Zhang, H. Fujita, and R. Hasegawa, "Qmaxsat: A partial max-sat solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 8, no. 1-2, pp. 95–100, 2012.

[36] C. Ansótegui, M. L. Bonet, J. Gabas, and J. Levy, "Improving sat-based weighted maxsat solvers," in *International conference on principles and practice of constraint programming*, pp. 86–101, Springer, 2012.

[37] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "clasp: A conflict-driven answer set solver," in *International Conference on Logic Programming and Nonmonotonic Reasoning*, pp. 260–265, Springer, 2007.

[38] J. Elffers and J. Nordström, "Divide and conquer: Towards faster pseudo-boolean solving.," in *IJCAI*, vol. 18, pp. 1291–1299, 2018.

[39] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren, "ASlib: A Benchmark Library for Algorithm Selection," *Artificial Intelligence Journal (AIJ)*, no. 237, pp. 41–58, 2016.

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

[41] Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, “Parallel sat solver selection and scheduling,” in *Principles and Practice of Constraint Programming: 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, pp. 512–526, Springer, 2012.

[42] C. Ansótegui, J. Gabas, Y. Malitsky, and M. Sellmann, “Maxsat by improved instance-specific algorithm configuration,” *Artificial Intelligence*, vol. 235, pp. 26–39, 2016.

[43] H. Hoos, R. Kaminski, M. Lindauer, and T. Schaub, “aspeed: Solver scheduling via answer set programming1,” *Theory and Practice of Logic Programming*, vol. 15, no. 1, pp. 117–142, 2015.

[44] L. Pulina and A. Tacchella, “A multi-engine solver for quantified boolean formulas,” in *Principles and Practice of Constraint Programming–CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings 13*, pp. 574–589, Springer, 2007.

[45] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. T. Schneider, and S. Ziller, “A portfolio solver for answer set programming: Preliminary report,” in *Logic Programming and Nonmonotonic Reasoning: 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings 11*, pp. 352–357, Springer, 2011.

[46] M. Maratea, L. Pulina, and F. Ricca, “A multi-engine approach to answer-set programming,” *Theory and Practice of Logic Programming*, vol. 14, no. 6, pp. 841–868, 2014.

[47] E. Nudelman, K. Leyton-Brown, A. Devkar, Y. Shoham, and H. Hoos, “Satzilla: An algorithm portfolio for sat,” *Solver description, SAT competition*, vol. 2004, 2004.

[48] F. Hutter, D. A. Tompkins, and H. H. Hoos, “Scaling and probabilistic smoothing: Efficient dynamic local search for sat,” in *Principles and Practice of Constraint Programming-CP 2002: 8th International Conference, CP 2002 Ithaca, NY, USA, September 9–13, 2002 Proceedings 8*, pp. 233–248, Springer, 2002.

[49] D. Mitchell, B. Selman, and H. Leveque, “A new method for solving hard satisfiability problems,” in *Proceedings of the tenth national conference on artificial intelligence (AAAI-92)*, pp. 440–446, 1992.

[50] A. Loreggia, Y. Malitsky, H. Samulowitz, and V. Saraswat, “Deep learning for algorithm portfolios,” in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[51] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[52] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with

deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[53] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[54] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.