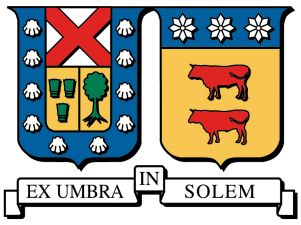
Universidad Técnica Federico Santa María
Departamento de Informática
Valparaíso, Chile



Probing Features for Automatic Algorithm Selection for Pseudo-Boolean Optimization

Amanda Paz Salinas Pinto

Tesis para optar al Grado de
Magíster en Ciencias de la Ingeniería Informática

Profesor Guía: Roberto Asín Achá, Ph.D.
Profesor Co-Guía: Ricardo Ñanculef
Profesor Correferente Interno:
Profesor Correferente Externo:
Presidente Comisión:

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☐ Memoria o trabajo de título; ☒ Tesis de Postgrado;

Título del trabajo: Probing Features for Automatic Algorithm Selection for Pseudo-Boolean Optimization

Nombre del candidato(a): Amanda Paz Salinas Pinto

Carrera / Grado: Magíster en Ciencias de la Ingeniería Informática

Campus: Casa Central ; **Departamento**: Departamento de Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Roberto Javier Asín Achá, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 16/03/2026 ; **Firma:** 

Estudiante o Candidato(a):

Fecha: 16/03/2026 ; **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Acknowledgements

The research is supported in part by grant #2112533: "NSF Artificial Intelligence Research Institute for Advances in Optimization (AI4OPT)". It is also supported by "ANID-Subdirección de Capital Humano/Magíster Nacional/ 2024-22241061".

Abstract

For NP-hard problems like Pseudo-Boolean Optimization (PBO), solver performance varies dramatically across instances, making algorithm selection a critical bottleneck. Current Automatic Algorithm Selection (AAS) systems for PBO depend primarily on static problem features, ignoring crucial dynamic information. In contrast, AAS for Boolean Satisfiability (SAT) has demonstrated that “probing" features, gathered from short solver runs, are essential for making high-quality, time-sensitive predictions.

This thesis addresses this gap by systematically integrating and evaluating probing features within AAS frameworks for PBO. The study investigates their effectiveness across multiple machine learning paradigms, including regression-based models, multiclass and multilabel classification approaches, and a novel hybrid framework that combines predictive and decision-oriented strategies. The outcome of this investigation is MetaPB, a new open-source meta-solver that leverages both static and probing features to improve solver selection performance.

Extensive empirical evaluation on benchmark instances from the 2024 PBO Competition demonstrates that MetaPB consistently outperforms the best individual solver in the portfolio. Furthermore, despite relying exclusively on open-source solvers, MetaPB narrows the performance gap to leading commercial systems and achieves results competitive with Gurobi. Beyond advancing the state of the art in AAS for PBO. This study not only establishes a new state of the art for AAS in PBO but also challenges the prevailing view of algorithm selection for PBO as a purely classification task, highlighting the effectiveness of hybrid, regression-informed approaches.

Resumen

Para problemas *NP-hard* como la Optimización Pseudo-Booleana (PBO), el rendimiento de los *solvers* varía drásticamente entre instancias, lo que convierte la selección de algoritmos en un cuello de botella crítico. Los sistemas actuales de Selección Automática de Algoritmos (AAS) para PBO dependen principalmente de características estáticas del problema, ignorando información dinámica crucial. En contraste, la AAS para Satisfacibilidad Booleana (SAT) ha demostrado que las características de dinámicas (probing), obtenidas a partir de ejecuciones cortas de los *solvers*, son esenciales para realizar predicciones de alta calidad y sensibles al tiempo.

Esta tesis aborda esta brecha mediante la integración y evaluación sistemática de características de probing dentro de marcos de AAS para PBO. El estudio investiga su efectividad a través de múltiples paradigmas de aprendizaje automático, incluyendo modelos basados en regresión, enfoques de clasificación multiclase y multietiqueta, y un nuevo marco híbrido que combina estrategias predictivas y orientadas a la decisión. El resultado de esta investigación es MetaPB, un nuevo meta-*solver* de código abierto que aprovecha tanto características estáticas como de probing para mejorar el rendimiento en la selección de *solvers*.

Una evaluación empírica exhaustiva sobre instancias de la Competencia PBO 2024 demuestra que MetaPB supera consistentemente al mejor *solver* individual del portafolio. Además, a pesar de basarse exclusivamente en *solvers* de código abierto, MetaPB reduce la brecha de rendimiento con respecto a *solvers* comerciales y alcanza resultados competitivos con Gurobi. Más allá de avanzar el estado del arte en AAS para PBO, este estudio no solo establece un nuevo referente en la selección automática de algoritmos para PBO, sino que también cuestiona la visión predominante de la selección de algoritmos para PBO como una tarea puramente de clasificación, destacando la eficacia de enfoques híbridos informados por regresión.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Automatic Algorithm Selection (AAS) is to automatically choose the most effective algorithm from a *portfolio* collection of solvers for a given problem instance. Since its formal inception by Rice in 1976 [54], this field has evolved considerably, demonstrating practical impact across numerous domains including Boolean Satisfiability (SAT) [50, 63], Pseudo-Boolean Optimization (PBO) [53], the Traveling Salesperson Problem (TSP) [33, 40], and various other NP-hard problems [34, 60]. The core principle is that no single algorithm performs optimally across all problem instances, which has motivated sophisticated selection strategies that can significantly improve overall performance compared to any single solver. These strategies enable the design of *meta-solvers*, which utilize the solvers in the portfolio by selecting, combining, or configuring them based on the specific input instance *features*.

Most AAS systems use machine learning (ML) techniques to recommend a solver for a given instance [39, 59]. These approaches receive *features* that describe the instance and learn the patterns that map these features to solver performance. The quality of the features is essential for guaranteeing good generalization performance on unseen instances. In the domain of PBO, which generalizes SAT by incorporating an objective function and pseudo-Boolean constraints, current algorithm selection approaches have predominantly relied on static features. These features are extracted directly from the problem formulation and offer structural insights but lack information on solver behavior. In contrast, the SAT community has demonstrated that probing features [8, 37, 39], obtained by briefly running solvers and recording their behavior, offer crucial additional information that significantly improves selection performance, especially in scenarios with tight time budgets. This insight has not been exploited in the PBO domain until now.

This thesis advances the state of the art in Automatic Algorithm Selection (AAS) for Pseudo-Boolean Optimization (PBO) along several dimensions. First, it introduces a comprehensive set of probing features derived from short executions of two complementary solvers: a Conflict-Driven Clause Learning (CDCL) PBO solver and a Branch-and-Bound (BnB) solver. These features are effective in capturing solver behavior characteristics that static features alone cannot, and they improve upon

previously proposed probing features taken from the SAT domain.

Second, this work provides a systematic empirical comparison of alternative machine learning formulations for algorithm selection in PBO, including regression, multiclass classification, multilabel classification, and a hybrid approach combining regression with multilabel predictions. The experimental results consistently show that regression models, which predict the performance of each solver and then select the best one, outperform direct classification approaches that attempt to identify the best solver directly. Moreover, a hybrid approach that incorporates information about whether a solver can solve an instance to optimality has achieved the best results overall. Given the prevalence of classification approaches in the literature, these findings suggest re-evaluating common assumptions about the best way to frame AAS in PBO.

Finally, this thesis presents MetaPB, an open-source meta-solver that implements the best-performing combination of features and learning formulations identified in our study. MetaPB integrates both static and probing information. Empirical evaluation on a diverse benchmark set demonstrates that MetaPB significantly outperforms previous AAS and the best individual solver in the portfolio, both in terms of the number of optimally solved instances and overall solution quality. It also surpasses the best solver from the 2024 PBO competition and achieves performance competitive with commercial tools such as Gurobi, while relying exclusively on open-source solvers.

1.1 Hypothesis

The central hypothesis of this thesis is that using dynamic features in AAS for PBO results in a lower $\hat{m}_{s^t}$, indicating a closer approximation to VBS performance than relying only on static features.

1.2 Objectives

The general objective of this work is to develop an Automatic Algorithm Selection (AAS) system for Pseudo-Boolean Optimization (PBO) that integrates dynamic features extracted from solver behavior during the early stages of execution in order to improve meta-solver performance according to the $\hat{m}_{s^t}$ metric.

The specific objectives associated with this thesis project are:

1. Update the algorithm selection benchmark for Pseudo-Boolean Optimization (PBO) by incorporating recent instances and solvers, including participants from the PBO 2024 competition.

2. Identify and analyze a set of dynamic features relevant to PBO instances.

3. Design and train a machine learning model that integrates both static and dynamic features to predict the most suitable solver for each instance.

4. Evaluate the impact of dynamic features on the performance of the algorithm selector using the $\hat{m}_{s^t}$ metric, comparing the results against baselines based solely on static features.

1.3 Structure

Chapter 2 provides the theoretical background, introducing the fundamental concepts of Pseudo-Boolean Optimization (PBO) and Automatic Algorithm Selection (AAS). Chapter 3 reviews related work, including current applications of AAS to SAT and PBO, as well as an overview of existing PBO solvers. Chapter 4 describes the proposed solver portfolio, the feature sets, and the machine learning formulations used in this study. Chapter 5 presents the experimental setup and results, including the validation and test procedures, together with a comparison of the different proposed formulations. Finally, Chapter 6 summarizes the main conclusions of this thesis and outlines directions for future research.

Chapter 2

Background

2.1 Combinatorial Optimization and NP-Hardness

Combinatorial Optimization studies problems in which the objective is to find the best solution among a finite but typically extremely large set of discrete alternatives [52]. These problems arise in classical combinatorial settings such as the Traveling Salesman Problem (TSP) [2], the Knapsack Problem [47] and vehicle routing problems [13].

Computational complexity theory provides a rigorous framework for analyzing and classifying computational problems according to their intrinsic difficulty [26]. A central class in this framework is NP (Nondeterministic Polynomial time), which contains decision problems whose proposed solutions can be verified in polynomial time. Many classical combinatorial optimization problems are NP-hard, meaning they are at least as difficult as the hardest problems in NP and that no polynomial-time algorithm is known for solving them. A fundamental reason for this difficulty is the phenomenon of combinatorial explosion: the number of feasible solutions typically grows exponentially with the size of the input, rendering exhaustive enumeration computationally infeasible even for moderately sized instances.

Given this inherent computational difficulty, NP-hard problems are typically addressed using exact algorithms (e.g., branch-and-bound or cutting planes) [12] or heuristic and metaheuristic approaches designed to produce high-quality solutions within practical time limits [29].

Because these problems frequently arise in industrial and scientific applications, developing efficient solution methods remains a central challenge in both operations research and computer science.

2.2 Boolean Satisfiability (SAT)

The **Boolean Satisfiability Problem (SAT)** [14] is the fundamental decision problem of propositional logic. It is one of the most extensively studied problems

in computer science and holds the distinction of being the first problem proven to be NP-complete. The objective of SAT is to determine whether a given Boolean formula admits a satisfying assignment. Formally, it is defined as follows.

- **Variables:** Let $X = \{x_1, x_2, \ldots, x_n\}$ be a finite set of Boolean variables.
- **Literals:** A literal is either a variable x_i or its negation $\bar{x}_i$.
- **Clauses:** A clause C_j is a disjunction of literals:

$$C_j = \bigvee_{i=1}^{k_j} \ell_{j,i}$$

  where $\ell_{j,i}$ denotes the i-th literal in clause C_j, and k_j is the number of literals in that clause.

A formula F is in **Conjunctive Normal Form (CNF)** if it is a conjunction of clauses:

$$F = \bigwedge_{j=1}^{m} C_j \tag{2.1}$$

where $m \in \mathbb{N}$ is the number of clauses.

The SAT problem asks whether there exists a truth assignment

$$\alpha : X \to \{0, 1\}$$

such that F evaluates to 1 (true) under α.

For example, consider the propositional formula ϕ defined as:

$$\phi = (x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3). \tag{2.2}$$

The formula ϕ is *satisfiable* because there exists a truth assignment, for instance

$$\alpha(x_1) = 1, \quad \alpha(x_2) = 1, \quad \alpha(x_3) = 0,$$

under which ϕ evaluates to 1.

2.3 SAT solving

Modern SAT solvers, such as *CadiCal* [5] and *Kissat* [6], operate by systematically exploring the space of partial assignments. A **partial assignment** specifies truth values for a subset of the variables, leaving the remaining ones unassigned. During the search, the solver incrementally extends these assignments through decision steps and deductive inference, while detecting and resolving conflicts along the way.

This search paradigm originates from the classical Davis–Putnam–Logemann–Loveland (DPLL) procedure [15], which introduced a depth-first backtracking framework combining branching decisions with unit propagation. In the late 1990s, Conflict-Driven Clause Learning (CDCL) [46] significantly enhanced DPLL by incorporating conflict analysis, clause learning, and non-chronological backtracking (backjumping). These mechanisms allow the solver to extract information from conflicts, prune large portions of the search space, and avoid repeating the same mistakes.

2.3.1 Davis–Putnam–Logemann–Loveland (DPLL)

The Davis–Putnam–Logemann–Loveland (DPLL) procedure [15] is a backtracking-based search algorithm for determining the satisfiability of propositional formulas in conjunctive normal form. The algorithm explores the space of truth assignments by alternating between logically implied deductions and heuristic branching.

To describe the procedure, consider the status of a clause under a partial assignment. A clause is:

- **satisfied** if at least one literal is assigned true,
- **falsified** if all literals are assigned false,
- **undetermined** if some literals are unassigned and none of the assigned literals is true.

DPLL repeatedly performs the following steps:

1. **Decision.** The solver selects an unassigned variable and assigns it a truth value according to a branching heuristic. Such an assignment is called a **decision** and introduces a new **decision level**, representing a branching point in the search tree.
2. **Unit propagation.** A **unit clause** is a clause containing a single literal. Under a partial assignment, however, any clause in which all literals except one are assigned false behaves as a unit clause. To avoid falsifying such a clause, the remaining unassigned literal must be assigned true. This forced assignment is called a **unit propagation**. The solver repeatedly performs unit propagation until no unit clauses remain. Assignments derived in this way are implied by earlier decisions and belong to the current decision level.
3. **Conflict detection.** If some clause becomes falsified under the current partial assignment, a conflict is detected.
4. **Backtracking.** Upon conflict, the solver undoes assignments up to a previous decision level and explores an alternative assignment.

2.3.2 Conflict-Driven Clause Learning (CDCL)

Conflict-Driven Clause Learning (CDCL) [46] extends the DPLL procedure by analyzing conflicts and adding new clauses that prevent the solver from repeating the same mistakes. While DPLL simply backtracks when a clause becomes falsified, CDCL extracts additional information from the conflict and permanently strengthens the formula.

The CDCL procedure, shown in Algorithm 2.1, incrementally extends a partial assignment ρ until either a satisfying assignment is found or unsatisfiability is proven.

Whenever a clause becomes falsified under the current assignment, a conflict occurs. The solver then performs conflict analysis to derive a learned clause, which is added to the formula. If the learned clause is empty, the instance is declared unsatisfiable; otherwise, the solver backtracks to an earlier decision level using backjump.

If no conflict arises and all variables are assigned, the formula is satisfiable. Otherwise, unit clauses are propagated whenever available. Modern CDCL solvers may additionally restart the search or reduce the learned clause database to maintain efficiency. When none of these cases applies, a new decision assignment is selected and the search continues.

Algorithm 2.1 CDCL

Require: CNF formula $\mathcal{C}$

$\rho \leftarrow \varnothing$
while true **do**
  if ρ falsifies some clause $C \in \mathcal{C}$ **then**
    $C_{\text{learn}} \leftarrow \text{analyzeConflict}(C, \rho)$
    if $C_{\text{learn}} = \emptyset$ **then**
      return UNSATISFIABLE
    end if
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_{\text{learn}}\}$
    $\text{backjump}(\rho, C_{\text{learn}})$
  else if all variables assigned in ρ **then**
    return SATISFIABLE, ρ
  else if unit clause C propagates ℓ **then**
    $\rho \leftarrow \rho \cup \{\ell\}$
  else
    choose decision literal ℓ
    $\rho \leftarrow \rho \cup \{\ell\}$
  end if
end while

Conflict Analysis and Clause Learning. When a clause becomes falsified under the current partial assignment, a conflict occurs. Each propagated assignment has

an associated **reason clause**, that is, the clause that forced the assignment during unit propagation.

Conflict analysis starts from the falsified clause and repeatedly applies **resolution** with reason clauses of literals assigned at the current decision level. Resolution can be read as an inference rule: given two clauses, one containing a literal x and the other containing its negation $\bar{x}$,

$$\frac{(A \vee x) \quad (B \vee \bar{x})}{A \vee B},$$

we derive the *resolvent* $A \vee B$, which is logically implied by the two original clauses. The numerator lists the clauses being resolved, the line indicates the inference step, and the denominator shows the resulting clause.

By resolving the conflicting clause with the reason clauses of recently assigned literals, the solver eliminates variables assigned at the current decision level. This process continues until a clause is obtained that contains exactly one literal assigned at the current decision level. The resulting clause is called the **learned clause**.

The learned clause is logically implied by the original formula and prevents the solver from repeating the same conflicting combination of assignments. After adding it to the set of clauses, the solver returns to the highest decision level at which the learned clause is not yet satisfied. At that level, the learned clause becomes unit and forces a new propagation.

2.4 Pseudo-Boolean Optimization Problem

PBO extends SAT [7] by allowing pseudo-Boolean constraints and adding an objective function to optimize. Let $\mathbb{B} = \{0, 1\}$, $x = (x_1, x_2, ..., x_n) \in \mathbb{B}^n$, and let a literal ℓ_i be x_i or $\bar{x}_i$. A pseudo-Boolean function is a mapping of the form

$$f(x) = \sum_{i=1}^{k} a_i \prod_{j \in S_i} \ell_j$$

where $a_i \in \mathbb{R}^+$ and S_i represent the set of literal indices defining the i-th *term*. A PBO instance can thus be defined as

$$\min_{x \in \mathbb{B}^n} f(x) \text{ s.t. } g_i(x) \geq a_i \ \forall i \in [m] = \{1, 2, ..., m\},$$

where f, g_1, $g_2 \dots g_m$ are pseudo-Boolean functions, and $a_i \in \mathbb{R}$, $\forall i \in [m]$.

Example. Consider three Boolean variables $x_1, x_2, x_3 \in \mathbb{B}$ representing binary decisions. A simple pseudo-Boolean optimization problem can be defined as

$$\begin{aligned} \min \quad & 3x_1 + 2x_2 + x_3 \\ \text{s.t.} \quad & 2x_1 + x_2 + x_3 \geq 2, \\ & x_1 + x_3 \geq 1. \end{aligned}$$

The objective function assigns a cost to each variable and the goal is to find a Boolean assignment minimizing the total cost while satisfying all constraints.

2.5 PB Solving

PB solving addresses the feasibility version of a problem defined by pseudo-Boolean constraints. The objective is to determine whether there exists a Boolean assignment that satisfies all constraints. The search terminates as soon as a satisfying assignment is found or unsatisfiability is established.

Two main approaches are commonly used for PB solving. The first approach translates pseudo-Boolean constraints into conjunctive normal form (CNF) and applies a CDCL SAT solver. This strategy is adopted by solvers such as MiniSAT+ [61], OpenWBO [48], and NaPS [56]. While this allows reuse of highly optimized SAT solving techniques, the encoding process may significantly increase the size of the instance, sometimes producing exponentially many clauses compared to the original PB formulation [22].

The second approach performs native pseudo-Boolean solving, avoiding CNF translation entirely. Solvers such as Sat4j [4] and RoundingSAT [20] operate directly on pseudo-Boolean constraints and extend the CDCL framework to linear inequalities. Decisions and propagation follow the same search structure as CDCL, but inference is no longer based on clause resolution. Instead, reasoning relies on the cutting planes proof system, where new constraints are derived through arithmetic operations over inequalities.

2.5.1 Conflict Analysis with Cutting Planes

Conflict analysis in native pseudo-Boolean solvers generalizes clause learning in SAT: linear constraints are iteratively combined using cutting planes inference rules until an asserting constraint is derived, which captures the conflict and guides future search decisions.

Starting from a conflicting constraint, the solver combines it with the constraints responsible for previous propagations. The goal is to eliminate propagated literals while preserving logical validity, ultimately deriving a constraint that prevents the same conflict from reoccurring during the search.

The fundamental operations employed in this process are presented below. These operations are based on [21], while the illustrative examples are adapted from [49].

Resolution Is the central inference rule used during PB conflict analysis. It combines two valid pseudo-Boolean constraints containing complementary literals in order to eliminate a pivot variable. This operation generalizes propositional resolution in SAT solving to linear inequalities.

Formally, let $c = \text{lcm}(a_1, b_1)$, where lcm denotes the *least common multiple*. From

$$\frac{a_1 x_1 + \sum_{i \geq 2} a_i \ell_i \geq A \qquad b_1 \bar{x}_1 + \sum_{i \geq 2} b_i \ell_i \geq B}{\sum_{i \geq 2} \left(\frac{c}{a_1} a_i + \frac{c}{b_1} b_i \right) \ell_i \;\geq\; \frac{c}{a_1} A + \frac{c}{b_1} B - c} \tag{2.3}$$

For example, lets define the constraints $C_1 : x + \bar{y} + z \geq 1$ and $C_2 : \bar{y} + \bar{z} \geq 1$, then resolution over $C1$ and C_2 results in the following constraint:

$$\frac{x + \bar{y} + z \geq 1 \qquad \bar{y} + \bar{z} \geq 1}{x + 2\bar{y} \geq 1} \tag{2.4}$$

Weakening This process removes literals from a constraint while preserving validity. Intuitively, the constraint becomes easier to satisfy, which enables coefficient normalization required for later combination steps.

Formally, given a constraint

$$\sum_i a_i \ell_i \geq A,$$

weakening with respect to literal ℓ_j produces

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_{i \neq j} a_i \ell_i \geq A - a_j} \quad [\text{weaken on } \ell_j] \tag{2.5}$$

For example,

$$\frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4}{2x_1 + 2x_3 + x_4 \geq 2} \quad [\text{weaken on } x_2] \tag{2.6}$$

Division This process rescales a constraint by a positive integer while rounding coefficients and the bound upward to preserve soundness. This operation is essential in rounding based cutting planes systems, where normalization enables subsequent resolution steps and limits coefficient growth during conflict analysis. It formally define as follows:

$$\frac{\sum_i a_i \ell_i \geq A}{\sum_i \left\lceil \frac{a_i}{c} \right\rceil \ell_i \geq \left\lceil \frac{A}{c} \right\rceil} \quad [\text{division by } c,\ c \in \mathbb{Z}_{>0}] \tag{2.7}$$

For example:

$$\frac{2x_1 + 2x_2 + 2x_3 \geq 3}{x_1 + x_2 + x_3 \geq 2} \quad [\text{divide by } 2] \tag{2.8}$$

Round-to-One Normalization The roundToOne procedure is a normalization step used during conflict analysis in the PBO solver RoundingSat [20]. Its purpose is to transform a pseudo-Boolean constraint so that a chosen pivot literal has coefficient 1 while preserving validity under the current partial assignment ρ.

During cutting-planes conflict analysis, constraints are combined with respect to a pivot variable. Large pivot coefficients may lead to rapid coefficient growth during successive resolution steps, typically caused by least common multiple computations. Normalizing the constraint before resolution mitigates this effect.

Given a constraint C:

$$\sum_i a_i \ell_i \geq A$$

and let ℓ be the pivot literal with coefficient, the coefficient of that literal on the constraint C

$$c = \text{coeff}(C, \ell).$$

The roundToOne procedure modifies C so that all coefficients become divisible by c, after which the constraint can be safely divided by c, resulting in a normalized constraint in which ℓ has coefficient 1.

As shown in Algorithm 2.2, normalization proceeds in two steps. First, literals not fixed by the current assignment are weakened whenever their coefficients are not divisible by c. This guarantees divisibility of all remaining coefficients. The constraint is then divided by c, introducing no additional rounding effects.

Algorithm 2.2 roundToOne(C, ℓ, ρ)

Require: PB constraint C, literal ℓ, assignment ρ

```
c ← coeff(C, ℓ)
for all literal ℓ_j ∈ C do
    if ℓ_j ∉ ρ and c ∤ coeff(C, ℓ_j) then
        C ← weaken(C, ℓ_j)
    end if
end for
return divide(C, c)
```

For example, consider the constraint

$$C : 4x + 3y + 2z \geq 5$$

and assume we want to normalize with respect to the pivot literal x. Here,

$$c = \text{coeff}(C, x) = 4.$$

The coefficients of y and z are not divisible by 4. Suppose neither $\bar{y}$ nor $\bar{z}$ is falsified by the current assignment, so weakening is allowed on both literals.

Weakening on y:

$$4x + 2z \geq 2.$$

Weakening on z:

$$4x \geq 0.$$

Now all remaining coefficients are divisible by 4. Dividing by 4 results in

$$x \geq 0.$$

Conflict Analysis Procedure Algorithm 2.3 describes the main conflict analysis mechanism for native pseudo Boolean solving. Starting from a constraint C_{conf} that is falsified under the current partial assignment ρ, the procedure derives a new constraint that prevents the same conflict from reoccurring.

As long as C_{conf} contains zero or more than one literal falsified at the current decision level, the algorithm considers the most recently assigned literal ℓ. If its complement occurs in C_{conf}, the constraint is resolved with the reason constraint of ℓ. Before resolution, both constraints are normalized using roundToOne (Algorithm 2.2) to ensure that the pivot has coefficient 1 and to control coefficient growth. The pivot is then eliminated by cutting-planes resolution.

The process repeats while backtracking along the trail of assignments. If no decisions remain, unsatisfiability is concluded. Otherwise, once exactly one literal of C_{conf} is falsified at the current decision level, the resulting constraint is returned (after a final normalization) as the learned constraint.

Algorithm 2.3 analyzePBConflict(C_{conf}, ρ)

Require: conflicting constraint C_{conf}
Require: trail ρ
while C_{conf} contains $\neq 1$ literals falsified at the current decision level **do**
  if no decision levels remain **then**
    return UNSATISFIABLE
  end if
  $\ell \leftarrow$ lastAssigned(ρ)
  if $\bar{\ell} \in C_{\text{conf}}$ **then**
    $C_{\text{conf}} \leftarrow$ roundToOne($C_{\text{conf}}, \bar{\ell}, \rho$)
    $C_{\text{reason}} \leftarrow$ roundToOne(reason(ℓ, ρ), ℓ, ρ)
    $C_{\text{conf}} \leftarrow$ resolve($C_{\text{conf}}, C_{\text{reason}}, \ell$)
  end if
  $\rho \leftarrow$ removeLast(ρ)
end while
$\ell \leftarrow$ literal of C_{conf} last falsified in ρ
return roundToOne($C_{\text{conf}}, \ell, \rho$)

2.6 PBO Solving

PBO extends pseudo-Boolean feasibility solving by introducing an objective function that must be minimized while satisfying a set of pseudo-Boolean constraints.

PBO generalizes several important optimization frameworks, including MaxSAT. In particular, weighted partial MaxSAT can be naturally encoded as a PBO problem [45] by associating a relaxation variable with each soft constraint and minimizing the total weight of violated constraints. Owing to this close relationship, modern PBO solvers commonly adopt algorithmic techniques originally developed for MaxSAT solving. In practice, many PBO algorithms rely on repeated calls to a PB solver while progressively improving bounds on the objective value until optimality is established. In the following, we describe the main approaches used for PBO solving.

2.6.1 Linear Search

Linear search is one of the earliest approaches for solving MaxSAT problems [10] and has been extended to PBO [16]. The main idea is to iteratively compute satisfying assignments while progressively tightening an upper bound on the objective value.

At each iteration, the solver searches for a feasible assignment satisfying the constraints, as illustrated in Algorithm 2.4. Whenever a solution is found, its objective value defines a new upper bound. A stronger constraint is then added, requiring subsequent solutions to have strictly smaller cost. The process continues until the resulting formula becomes unsatisfiable, thereby proving the optimality of the last solution found.

Algorithm 2.4 LinearSearchPBO

Require: objective $\min \sum_{i=1}^{n} a_i l_i$
Require: PB constraints $\mathcal{C}$
1: $\rho_{\text{best}} \leftarrow \varnothing$
2: **while** true **do**
3: $(status, \rho) \leftarrow solver.solve(\mathcal{C})$
4: **if** $status =$ UNSATISFIABLE **then**
5: **return** ρ_{best}
6: **end if**
7: $\rho_{\text{best}} \leftarrow \rho$
8: add constraint

$$\sum_i a_i l_i \leq \sum_i a_i \, \rho(l_i) - 1$$

9: **end while**

Linear search is simple and effective when good solutions can be found quickly,

but it may require many solver calls when the optimal value is far from the initial solutions.

2.6.2 Core-Guided Search

Core-guided search originates from MaxSAT solving and has been successfully extended to PBO [25]. The approach transforms optimization into a sequence of feasibility checks by iteratively identifying **unsatisfiable cores**, namely subsets of constraints that cannot be satisfied simultaneously.

Given a PBO instance, let the objective function be

$$f(x) = \sum_i a_i \ell_i.$$

Core-guided optimization, shown in Algorithm 2.5, proceeds by progressively improving a lower bound on the objective value. The process starts from an initial lower bound by invoking the solver under a set of assumptions, for example, setting $\ell_i = 0$ for all literals ℓ_i appearing in the objective function. The solver then checks the feasibility of the problem under these assumptions. If the formula is satisfiable, a feasible assignment exists and optimality is established for the current bound. Otherwise, the solver produces an unsatisfiable core.

$$\mathcal{K} \subseteq \{\ell_i\},$$

containing literals involved in the violation of the objective bound.

From this core, a valid pseudo-Boolean inequality is derived stating that not all literals in $\mathcal{K}$ may simultaneously take value 0:

$$\sum_{\ell_i \in \mathcal{K}} \ell_i \geq 1.$$

This strengthening constraint increases the lower bound of the objective function and removes infeasible assignments. The process iterates until a satisfiable bound is obtained.

Several improvements to PBO solving have been proposed. Core Boosting [3] first applies core-guided search to derive a strong lower bound and subsequently switches to linear search to efficiently reach optimality. Hybrid or interleaving approaches [1] alternate between linear and core-guided search throughout the solving process. While switching between strategies may incur significant overhead in CNF-based solvers, it is comparatively inexpensive in native pseudo-Boolean solvers [17].

2.6.3 Branch and Bound

Pseudo-Boolean Optimization can also be addressed using Branch-and-Bound approaches [41], which explore a search tree of restricted subproblems while maintaining

Algorithm 2.5 CoreGuidedPBO

Require: PB constraints $\mathcal{C}$
Require: objective $\min \sum_i a_i l_i$
1: $A \leftarrow \{\bar{l}_i \mid i\}$ #set of assumptions
2: $LB \leftarrow 0$
3: **while** true **do**
4: $(status, \rho, \mathcal{K}) \leftarrow solver.solve(\mathcal{C}, A)$
5: **if** $status =$ SATISFIABLE **then**
6: **return** LB, ρ
7: **end if**
8: add constraint

$$\sum_{\bar{l}_i \in \mathcal{K}} l_i \geq 1$$

9: $\delta \leftarrow \min_{\bar{l}_i \in \mathcal{K}} a_i$
10: $LB \leftarrow LB + \delta$
11: update assumptions A
12: **end while**

bounds on the objective value. Each node represents a partially fixed optimization problem obtained by assigning values to a subset of variables, thereby partitioning the search space.

To guide exploration, a *relaxation* of the problem is solved at every node. In practice, this is usually a **Linear Programming (LP) relaxation** [62], where the Boolean constraints $x_i \in \{0, 1\}$ are replaced by $x_i \in [0, 1]$, resulting in a continuous optimization problem. The solution to the LP relaxation provides a bound on the best objective value achievable within the corresponding subproblem. A node can be pruned when the relaxation is infeasible, its bound is worse than the current best solution (the *incumbent*), or when integrality is already satisfied.

Branch-and-Cut extends this framework by strengthening relaxations using cutting planes derived from violated linear inequalities. In the pseudo-Boolean setting, inference rules such as division and saturation (Section 2.5.1) can be interpreted as examples of such cuts.

Algorithm 2.6 describes a Branch-and-Cut procedure for solving Pseudo-Boolean Optimization (PBO) problems. The algorithm explores a search tree of restricted subproblems by solving LP relaxations to obtain bounds on the objective value. Subproblems whose bounds cannot improve the incumbent solution are pruned. When a fractional solution is found, cutting planes are added to strengthen the relaxation; otherwise, branching decisions partition the search space. Optimality is established once all nodes have been explored or pruned.

Branch-and-Cut methods form the basis of modern solvers for discrete optimization problems, including Pseudo-Boolean Optimization. State-of-the-art per-

Algorithm 2.6 Branch-and-Cut for Pseudo-Boolean Optimization

Require: minimize $\sum_i a_i x_i$
best $\leftarrow +\infty$, NODELIST $\leftarrow \{\text{root}\}$
while NODELIST $\neq \emptyset$ **do**
    select node N and solve its LP relaxation obtaining x^*
    if LP infeasible or value(x^*) $\geq$ best **then**
        continue
    else if x^* integral **then**
        best $\leftarrow$ value(x^*)
    else if violated cut exists **then**
        add cut to N and reinsert into NODELIST
    else
        branch on fractional variable x_j
    end if
end while
return best

formance is typically achieved by commercial systems such as Gurobi [31], while academic solvers such as SCIP [9] provide competitive open-source alternatives.

2.7 Automatic Algorithm Selection (AAS)

Automatic Algorithm Selection (AAS) [54] aims to construct meta-solvers that automatically choose the most appropriate solver from a portfolio based on characteristics of a given problem instance. Typically, this selection is performed using machine learning models trained on historical performance data.

Formally, given a set of problem instances I and a portfolio of solvers A, the goal is to learn a mapping

$$selector^t : I \to A,$$

such that $selector^t(i)$ identifies the solver expected to perform best on instance $i \in I$ under a time limit t.

To evaluate the effectiveness of an algorithm selector, the normalized performance metric $\hat{m}_{selector^t}$ introduced in [43] has become standard.

Let $o_a(i,t)$ be the best objective value obtained by a solver $a \in A$ for instance $i \in I$ given a time-limit t. We define $o_{\min}(i,t)$ as the minimum value of $o_{algorithm}(i,t)$ observed across all solvers $algorithm \in A$ at time t, and $o_{\max}(i,t)$ as the maximum value. The normalized objective value $n_a(i,t)$ of solver $a \in A$, on instance $i \in I$ at t, is defined as:

$$n_a(i,t) = \begin{cases} 0 & \text{if } o_a(i,t) \text{ is optimal,} \\ 1 & \text{if } o_a(i,t) = o_{\min}(i,t) \\ 3 & \text{if } o_a(i,t) \text{ undefined and } o_{\max}(i,t) \text{ defined,} \\ 1 + \dfrac{o_a(i,t) - o_{\min}(i,t)}{o_{\max}(i,t) - o_{\min}(i,t)} & \text{otherwise.} \end{cases} \tag{2.9}$$

Lower $n_a(i,t)$ values are better, indicating a solver's objective value is closer to the best observed across solvers.

We define the normalized cumulative performance of a solver $a \in A$ at time t, and the normalized cumulative performance of meta-solver $s^t \in \{I \to A\}$ as:

$$m_{a^t} = \sum\nolimits_{i \in I} n_a(i,t), \qquad m_{s^t} = \sum\nolimits_{i \in I} n_{s(i)}(i,t) \tag{2.10}$$

The Single Best Solver at time t (SBS^t) is the solver $a \in A$, such that $\forall a' \in A : m_{a^t} \leq m_{a'^t}$. The Virtual Best Solver, at time t (VBS^t) is a theoretically perfect meta-solver such that $m_{\text{VBS}^t} = \sum\limits_{i \in I} \min\limits_{a \in A} n_a(i,t)$.

In this work, we distinguish between "VBS-all", where A includes all solvers in our portfolio, and "VBS-Open-Source", where A is restricted to the open-source solvers in the portfolio: NaPS, RoundingSat, Mixd-Bag and SCIP, which additionally employ complementary solving techniques. Analogously, we distinguish between "SBS-all," where the best single solver is selected from the full portfolio, and "SBS-Open-Source", where the selection is limited to the open-source solvers.

The $\hat{m}_{s^t}$ metric for meta-solver s^t is defined as:

$$\hat{m}_{s^t} = \frac{m_{s^t} - m_{\text{VBS}^t}}{m_{\text{SBS}^t} - m_{\text{VBS}^t}}\,. \tag{2.11}$$

Note that an $\hat{m}_{s^t}$ value of 0 indicates that the algorithm selector is optimal, a value of 1 indicates that the selector performs as well as SBS^t, and values above 1 indicate that the meta-solver is not useful.

Example Consider a portfolio $A = \{a_1, a_2, a_3\}$ and instances $I = \{i_1, i_2, i_3\}$ with observed objective values under time limit t:

	a_1	a_2	a_3
i_1	10	8	8^*
i_2	15	14^*	16
i_3	7^*	9	8

Here, * indicates the solver certifies the optimal value for that instance.

Compute $o_{\min}(i,t)$ and $o_{\max}(i,t)$ based on all observed values:

	$o_{\min}$	$o_{\max}$
i_1	8	10
i_2	14	16
i_3	7	9

Using Equation (2.9), the normalized values $n_a(i,t)$ are:

	a_1	a_2	a_3
i_1	2	1	0
i_2	1.5	0	2
i_3	0	2	1.5

Cumulative performance (Equation (2.10)):

$$m_{a_1} = 2 + 1.5 + 0 = 3.5, \quad m_{a_2} = 1 + 0 + 2 = 3, \quad m_{a_3} = 0 + 2 + 1.5 = 3.5$$

The Single Best Solver SBS^t is a_2, this means $m_{\text{SBS}^t} = 3$. The Virtual Best Solver VBS^t picks the best solver per instance:

$$m_{\text{VBS}^t} = \min(1,0,0) + \min(0.5,0,1) + \min(0,1,0.5) = 0 + 0 + 0 = 0.$$

Now, let us consider several illustrative cases for a meta-solver selection.

Case A: Meta-solver chooses the optimal solver for each instance. Suppose the meta-solver selects a_3 for i_1, a_2 for i_2, and a_1 for i_3, where the chosen solvers are optimal whenever indicated. The cumulative performance of the meta-solver is

$$m_{\text{meta-solver}^t} = 0 + 0 + 0 = 0,$$

and the normalized performance metric becomes

$$\hat{m}_{\text{meta-solver}^t} = \frac{m_{\text{meta-solver}^t} - m_{\text{VBS}^t}}{m_{\text{SBS}^t} - m_{\text{VBS}^t}} = \frac{0-0}{3-0} = 0.$$

This value indicates a perfect meta-solver that always chooses the best solver for each instance.

Case B: Meta-solver partially correct. If the meta-solver selects a_2 for i_1, a_2 for i_2, and a_3 for i_3, it only partially identifies the optimal solver. The cumulative performance is

$$m_{\text{meta-solver}^t} = 1 + 0 + 1.5 = 2.5,$$

and the normalized performance metric is

$$\hat{m}_{\text{meta-solver}^t} = \frac{2.5-0}{3-0} \approx 0.833.$$

This result is better than the Single Best Solver but not perfect, indicating a useful yet imperfect meta-solver.

Case C: Meta-solver chooses poorly. Consider the meta-solver selects a_1 for i_1, a_3 for i_2, and a_2 for i_3, consistently avoiding the optimal solvers. Its cumulative performance is

$$m_{\text{meta-solver}^t} = 2 + 2 + 2 = 6,$$

leading to a normalized performance metric

$$\hat{m}_{\text{meta-solver}^t} = \frac{6 - 0}{3 - 0} = 2.$$

This value exceeds 1, indicating a meta-solver that performs worse than the Single Best Solver.

2.8 Machine Learning

Machine Learning (ML) is a rapidly evolving field concerned with the development of computational algorithms capable of learning patterns and making decisions from data without being explicitly programmed. In recent decades, ML has become a fundamental component in a wide range of applications, including pattern recognition, aerospace engineering, finance, entertainment, and medical analysis [19].

In machine learning, algorithms are designed to improve their performance on a specific task by learning patterns from data. One of the most widely adopted paradigms is supervised learning, in which the model is trained using a labeled dataset composed of input feature vectors and their corresponding target outputs [30]. The objective of supervised learning is to learn a mapping between inputs and outputs that enables accurate predictions for previously unseen samples.

Supervised learning problems are commonly divided into two main categories: classification and regression. In classification tasks, the goal is to assign an input instance to one of a finite set of predefined classes or categories. Typical examples include object recognition or medical diagnosis, where predictions correspond to discrete labels. In contrast, regression tasks aim to predict continuous numerical values rather than discrete categories. The model learns the functional relationship between input variables and a continuous target variable, enabling estimation of quantities such as temperature, financial indicators, or physical measurements.

Motivated by the growing integration of learning-based techniques into optimization solvers, this work employs such machine learning models to guide algorithmic decisions. The next sections provide a description of the predictive models used in this work.

2.8.1 Ensemble Learning

Ensemble learning refers to a class of techniques that combine multiple predictive models to obtain improved generalization performance compared to individual models [32]. The underlying principle is that aggregating diverse models can reduce

variance, bias, or both, thereby producing more robust predictions. Two widely used ensemble strategies are:

Bagging: Bootstrap Aggregating (Bagging) trains multiple models independently using different bootstrap samples drawn from the training dataset. The final prediction is obtained by aggregating the outputs of all models, typically through averaging or majority voting.

Boosting: Boosting methods construct models sequentially, where each new learner focuses on correcting the errors made by the previous ensemble. This iterative refinement progressively improves predictive performance.

2.8.2 Decision Trees

A Decision Tree is a hierarchical predictive model that recursively partitions the feature space using decision rules derived from input variables [11]. Decision trees can be applied to both classification and regression tasks and are valued for their interpretability and flexibility.

Tree construction proceeds by selecting, at each node, the feature and splitting criterion that optimally separates the data according to an impurity measure, such as information gain for classification or variance reduction for regression. This recursive partitioning continues until a stopping condition is satisfied.

Decision trees are capable of handling both numerical and categorical variables and provide intuitive visual representations of the decision-making process.

2.8.3 Random Forest

Random Forest is an ensemble learning method that combines multiple decision trees to improve predictive accuracy and robustness [11]. Each tree is trained on a bootstrap sample of the data, and a random subset of features is considered at each split, which decorrelates the trees and reduces overfitting.

In regression tasks, predictions are obtained by averaging the outputs of all trees, effectively reducing variance while capturing nonlinear relationships and complex feature interactions with relatively little hyperparameter tuning.

2.8.4 Gradient Boosting Regressor

Gradient Boosting [24] is an ensemble technique based on the boosting paradigm, in which models are constructed sequentially to minimize a predefined loss function. Unlike Random Forest, where trees are built independently, Gradient Boosting iteratively adds weak learners that correct the residual errors of the existing ensemble.

At each iteration, a new model is fitted to the negative gradient of the loss function with respect to the current predictions. The final model corresponds to a weighted sum of all learned trees.

By performing gradient descent in function space, Gradient Boosting Regressors can model highly complex nonlinear relationships and achieve strong predictive performance. However, their effectiveness depends on careful selection of hyperparameters such as learning rate, number of estimators, and tree depth, as inadequate regularization may lead to overfitting.

2.8.5 K-Nearest Neighbors

K-Nearest Neighbors (KNN) [23] is a non-parametric, instance-based learning algorithm used for classification and regression. Instead of learning an explicit model, KNN stores the training data and predicts outputs based on the k closest samples to a query instance according to a distance metric. Classification is typically performed by majority voting, while regression relies on averaging neighbor values. The method is sensitive to the choice of k, feature scaling, and distance measure, and may incur high prediction costs in high-dimensional datasets.

Chapter 3

Literature Review

3.1 Pseudo-Boolean Optimization Solvers

Below, we describe PBO solvers that serve as competitive alternatives to be included in a complementary portfolio of solvers.

- **Gurobi** [31] is a commercial solver using a branch-and-bound approach based on linear programming, enhanced with presolving, cutting planes, heuristics, and parallel processing.
- **NaPS** [56] is a MaxSAT solver based on MiniSat+ [18] that translates a PBO instance to a MaxSAT instance using Binary Decision Diagrams (BDD), and solves the resulting problem via binary search and alternative search strategies.
- **RoundingSat** [20] is a PBO solver that combines the CDCL procedure, based on the cutting planes proof system, with LP relaxation and cut generation. Once a feasible solution is found, it is optimized using model-improving and core-guided search.
- **SCIP** [9] is an open-source software tool designed for solving MIP, mixed integer nonlinear programming (MINLP), and constraint programming (CP) problems. To solve PBO, SCIP uses LP relaxations, the branch-and-cut algorithm, and cutting plane techniques.
- **Mixed-Bag** [38] combines three components in sequence: PB-OLL-RS (a core-guided solver based on RoundingSat), the PaPILO preprocessor [28], and SCIP as the final solver.

3.2 Algorithm Selection for SAT

In the SAT community, understanding the structural properties of SAT instances through informative features has become fundamental for performance prediction

and algorithm selection. The SATzilla feature set [51, 37] is one of the most widely used collections of SAT instance features. It has been successfully applied to Automatic Algorithm Selection (AAS) [50], Algorithm Configuration [35], and runtime prediction [37], among other tasks.

3.2.1 SATzilla Feature Set

In [51], the authors introduced 91 features designed to characterize SAT instances. Their goal was to better understand empirical hardness in SAT and to predict whether a given instance would be difficult to solve.

The methodology consists of extracting a comprehensive set of features, removing redundant ones based on correlation analysis, and training supervised learning models to predict solver runtime. The models are evaluated using the Root Mean Squared Error (RMSE) between actual and predicted runtimes. To increase expressiveness, the feature space is augmented with all pairwise products of features.

The 91 features are grouped into the following categories:

1. **Problem Size Features:** Basic statistics capturing instance size, including the number of variables, number of clauses, and clause-to-variable ratios.

2. **Graph-Based Features:** Statistics derived from three graph representations of a SAT instance:

   - *Variable–Clause Graph (VCG):* A bipartite graph with nodes for variables and clauses, and edges indicating variable occurrence in clauses.
   - *Variable Graph (VG):* A graph containing only variable nodes, where two variables are connected if they appear together in at least one clause.
   - *Clause Graph (CG):* A graph containing only clause nodes, with edges between clauses that share complementary literals.

   From these graphs, several statistics are extracted, such as node degree distributions and related summary measures.

3. **Balance Features:** Measures capturing different notions of balance in the formula, such as the distribution of positive and negative literals.

4. **Proximity to Horn Formula:** Features quantifying how close an instance is to a Horn formula.

5. **LP-Based Features:** Features obtained by solving a linear programming relaxation of an integer programming formulation of the SAT instance. These include the integer slack vector, the fraction of integer variables in the LP solution, and the LP objective value.

6. **DPLL Search Space Probing Features:** These features are obtained by running a DPLL procedure with exponentially increasing depth limits and measuring statistics such as the number of unit propagations at each depth. Additionally, random depth-first probes are performed by repeatedly instantiating random variables and applying unit propagation until a contradiction is found.

7. **Local Search Probing Features:** Statistics collected from runs of stochastic local search algorithms such as GSAT [58] and SAPS [36]. Each algorithm is executed multiple times, and characteristics of the search trajectory are averaged across runs.

Some of these features are *static*, meaning they can be computed directly from the SAT formula with relatively low computational cost. Others are *probing features*, which require running a solver for a limited amount of time and extracting statistics from its behavior. Probing features are generally more informative but computationally more expensive.

The feature set was later expanded to 138 features in [37]. The extended version includes additional probing features such as clause learning statistics (e.g., number and average length of learned clauses), survey propagation features, and timing features capturing the CPU time required for feature computation. The expansion to 138 features increases expressiveness but may exacerbate feature redundancy and risk overfitting, particularly when training data is limited.

More recently, [59] revisited the SATzilla feature set and introduced an updated and modernized implementation of the feature extraction framework.

3.2.2 SATzilla as a Portfolio-Based Algorithm Selector

Several works [50, 63, 64] combine SATzilla features with machine learning techniques to build automatic algorithm selection systems for SAT.

In [50], the authors propose SATzilla, a portfolio-based solver that predicts the runtime of multiple SAT solvers using empirical hardness models built from instance features and ridge regression.

One major innovation is the use of *hierarchical hardness models*. The system first predicts the probability that an instance is satisfiable and then combines specialized runtime models for satisfiable and unsatisfiable instances. At runtime, SATzilla executes pre-solvers for a short time to quickly solve easy instances, computes instance features, predicts the runtime of each candidate solver, and selects the solver expected to perform best.

Extensive experimental results, including those from the 2007 SAT Competition, demonstrate that SATzilla significantly outperforms the best single solver. In many cases, it solves more instances and achieves substantially lower average runtime,

showing that learned per-instance solver selection can be more effective than relying on a single dominant algorithm.

In [64], the authors introduce an improved selection procedure based on an explicit cost-sensitive loss function. Instead of predicting runtime and minimizing regression error, they directly optimize a loss that reflects the performance impact of incorrect solver choices. Misclassifications are penalized proportionally to the absolute performance difference between solvers: confusing two solvers with similar runtimes incurs a small penalty, whereas selecting a solver that times out instead of a fast one incurs a large penalty.

To implement this idea, they construct a collection of 99 cost-sensitive decision forests following the standard random forest methodology. By aligning the learning objective with portfolio performance rather than runtime prediction accuracy, the improved SATzilla version further increases the number of solved instances and overall robustness.

3.3 Automatic Algorithm Selection for PBO

For PBO, [53] present a comprehensive study of multi-classification-based Automatic Algorithm Selection (AAS) methods in an anytime setting. The goal is to select, for a given PBO instance and a specified time limit, the solver that is expected to achieve the best objective value within that time budget.

3.3.1 Feature Design

In contrast to approaches that rely on dynamic features, the authors restrict their models to static, fast-to-compute structural features extracted directly from the PBO instance. The domain-specific feature set includes:

1. **Number of constraints.** The total number of pseudo-Boolean constraints present in the instance.
2. **Number of variables.** The number of Boolean decision variables appearing in the instance.
3. **Linearity indicator.** A binary feature indicating whether the instance contains non-linear constraints.
4. **Distribution of the number of terms per constraint.** Constraints are classified according to the number of terms they contain (one, two, three, or four or more terms).
5. **Term degree distribution.** Represents the percentage of terms involving one, two, three, or four or more variables.

6. **Objective function size.** Defined as the proportion of terms appearing in the objective function relative to the total number of terms.

7. **Percentage of positive terms in constraints.** The fraction of constraint terms with positive coefficients with respect to the total number of constraint terms.

8. **Percentage of positive terms in the objective function.** The fraction of objective-function terms with positive coefficients.

Additionally, the time limit (timestep) is explicitly incorporated as a feature in the anytime models, allowing the selector to capture temporal variations in solver performance.

3.3.2 Learning Formulation.

The problem is formulated as a multi-class classification task. Given an instance–timestep pair (i, t), the model predicts the solver expected to perform best within the portfolio. Solvers serve as class labels.

The portfolio consists of seven solvers: Naps [56], OpenWBO [48], Clasp [27], LS-PBO [42], Gurobi [31], and RoundingSAT [20]. Several machine learning models are evaluated, including Random Forest [11], Gradient Boosting [24], k-Nearest Neighbors (KNN) [23], and Convolutional Neural Networks (CNN) [44].

3.3.3 Evaluation Methodology

Performance is assessed using both classification accuracy and the anytime metric $\hat{m}$. In this work, the $\hat{m}$ metric is computed as the cumulative normalized score across all instance–timestep pairs. Experiments use a one-hour time limit, discretized into 500 logarithmically spaced timesteps, recording the incumbent objective value at each timestep.

3.3.4 Results and Discussion.

The best-performing configuration uses the full domain-specific feature set and achieves an accuracy of approximately 0.71. Feature importance analysis highlights the timestep as the most influential feature, followed by objective function related features (e.g., percentage of positive terms in the objective), as well as the number of constraints and variables. These results indicate that PBO-specific structural characteristics substantially improve predictive performance compared to the basic feature set. Models incorporating time as an explicit feature consistently outperform time-agnostic variants, reflecting the temporal variability in solver behavior.

Chapter 4

Methodology

This chapter presents the solver portfolio, instance dataset, feature sets, and machine learning formulations employed in this thesis. The methodology builds upon our previously published work [57].

4.1 Solver Portfolio and PBO Instances

The solver portfolio was constructed using participants from the 2024 PBO Competition optimization track [55]. Selection focused on top-performing solvers as well as those exhibiting complementary solving strategies. In addition, Gurobi was included due to its strong performance reported in related work [53].

The final portfolio consists of the following solvers: Gurobi, NaPS, RoundingSat, Mixed-Bag, and SCIP. For RoundingSat, two configurations were included: its default hybrid optimization mode (denoted as RoundingSat) and its core-boosted mode (denoted as RoundingSat-CB). These configurations exhibit complementary behavior and competitive standalone performance.

Benchmark instances were obtained from the 2024 PBO Competition website [55], which includes instances from competitions held between 2005 and 2016. Only optimization and linearized instances were considered, resulting in an initial dataset of 14,586 instances.

Solver performance was evaluated using a one-hour time limit on an Intel Xeon E5-2670 v3 machine with 64GB RAM. During execution, objective values were recorded at five time checkpoints: 60, 100, 300, 600, and 3600 seconds. These checkpoints enable the analysis of performance dynamics under varying time constraints.

Instances solved optimally by all solvers within 60 seconds were classified as trivial and removed. Likewise, instances unsolved by any solver were excluded. After filtering, 6,394 instances remained for analysis.

Additionally, a separate set of 478 instances from the 2024 PBO Competition optimization track was reserved for evaluation. These instances were selected by competition organizers to represent particularly challenging cases and were used to

assess generalization performance on difficult benchmarks.

4.2 Probing Feature Sets

This thesis introduces two complementary probing feature sets designed to characterize solver behavior during the early stages of execution: a CDCL-based feature set and a Branch-and-Bound (BnB) feature set. Rather than relying solely on static instance properties, probing features capture *dynamic search behavior*, providing information about how a solver interacts with a specific instance.

The CDCL feature set extends probing strategies previously developed for SAT solving [37] and adapts them to the Pseudo-Boolean Optimization (PBO) setting. Additional optimization-oriented features were introduced to better capture characteristics specific to pseudo-Boolean reasoning.

Features were extracted during the execution of `RoundingSat` using a probing time limit of five seconds. Solver statistics were sampled at conflict events and averaged over the probing period to obtain stable measurements. The resulting feature set, denoted as **CDCL**, combines adapted SAT probing features with newly proposed metrics.

4.2.1 CDCL Probing Features

Solver Activity Metrics. These features quantify the overall progress and intensity of the search process:

- **Decisions:** number of variable assignments chosen by the solver.
- **Propagations:** number of assignments implied by constraints through unit propagation.
- **Conflicts:** number of detected conflicts during search.
- **Learned Constraints:** number of constraints derived through conflict analysis and other heuristics.
- **Restarts:** number of times the solver resets the search.

Together, these metrics describe how actively the solver explores and prunes the search space.

Clause Quality Metric. These features assess the structural quality of learned constraints:

- **Learned Clauses Length:** the average number of literals contained in learned constraints. Shorter learned constraints generally propagate more effectively and restrict larger portions of the search space.

- **Literal Block Distance (LBD)** The **LBD value** measures the number of distinct decision levels appearing in a learned constraint. Low LBD values indicate constraints linking few decision levels and are typically associated with highly effective learned constraints that improve future propagation. No features of this type have been proposed before.

Efficiency Metrics. To characterize search efficiency, several ratios between solver operations are introduced:

- **Propagation-to-Conflict Ratio:** average number of propagated assignments observed per conflict. It reflects how much inference is required before the solver derives a contradiction.
- **Propagation-to-Decision Ratio:** average number of implied assignments generated per branching decision. It indicates the strength of propagation triggered by each decision.
- **Decision-to-Conflict Ratio:** average number of branching decisions made per conflict. It captures how quickly decisions lead to inconsistencies in the current search.

These ratios normalize raw activity counts and provide insight into the effectiveness of solver decisions independently of runtime.

Conflict Resolution Metrics. These features describe the complexity of conflict analysis:

- **Conflict Resolution Steps:** number of inference steps required to derive a learned constraint during conflict analysis.
- **Trail Pops:** number of assignments removed from the decision trail when resolving conflicts.

4.2.2 Branch-and-Bound Probing Features

To capture complementary information related to optimization oriented solving, a novel probing feature set based on Branch-and-Bound search was introduced (denoted as **BnB**). These features were extracted using `SCIP` executed under a five-second probing timeout.

While CDCL features characterize logical inference behavior, BnB features describe progress toward optimality and search-tree exploration.

Solution Quality Indicators. These features measure optimization progress:

- **Primal Bound:** objective value of the best feasible solution currently known.

- **Dual Bound:** bound obtained from relaxations representing the best achievable objective value.
- **Gap:** relative difference between primal and dual bounds, indicating distance from optimality.
- **Current Objective Value:** objective value associated with the incumbent solution.

Variable Metrics. These features describe structural properties of incumbent solutions:

- **Active Variables:** number of binary variables assigned value 1.
- **Inactive Variables:** number of binary variables assigned value 0.

Search Tree Metrics. These metrics quantify exploration of the Branch-and-Bound tree:

- **Feasible Leaves:** number of explored leaf nodes corresponding to feasible solutions.
- **Iterations:** number of processed nodes or LP solves.
- **Solutions Found:** total number of feasible solutions discovered during probing.

Solver Activity Metric. This feature reflects how strongly variables influence the search process during probing.

- **Average Variable Activity** measures how frequently variables participate in branching decisions or constraint reasoning.

Together, the CDCL and BnB probing feature sets provide complementary perspectives on solver behavior.

4.2.3 Machine Learning Tasks and Models

Four alternative learning formulations are investigated for solver selection.

Multiclass Classification: It trains the selector to predict the solver from the portfolio, achieving the lowest normalized objective value within the time limit. In case of ties, the model is trained to choose the fastest one.

Regression: It trains the selector to predict the normalized objective value $n_a(i,t)$ of each solver-instance pair at time t, as defined in Equation 2.9. The selector then chooses the solver with the lowest value.

Multilabel Classification: It trains the selector to predict, for each solver in the portfolio, the probability that it fails to solve an instance optimally within the time limit. For each solver, a binary label that indicates success (0) or failure (1) on the instance is predicted. The selector then chooses the solver with the lowest predicted failure probability.

Hybrid: This approach combines the predictions of both regression and multilabel models to compute a score for each solver-instance pair. The final score is defined as:

$$\text{score}_a(i,t) = \alpha \cdot \widehat{n}_a(i,t) + (1-\alpha) \cdot \widehat{p}_a(i,t), \tag{4.1}$$

where $\widehat{n}_a(i,t)$ is the predicted normalized objective from the regression model, $\widehat{p}_a(i,t)$ is the predicted failure probability from the multilabel classifier, and $\alpha \in [0,1]$ is a weighting parameter. The selector chooses the solver minimizing this score, thereby balancing expected solution quality and risk of failure at time t.

The following machine learning models were evaluated across all formulations: Random Forest [11], Gradient Boosting [24], and K-Nearest Neighbors [23]. These models were chosen due to their computational efficiency and strong empirical performance in related algorithm selection studies [53].

4.2.4 MetaPB

MetaPB[1] is the meta-solver introduced in this work, designed to predict the most suitable solver for a given PBO instance by combining static and probing features. Figure 4.1 illustrates the overall pipeline.

Given an input PBO instance, MetaPB first extracts static features that describe the structural properties of the instance. Then, for instances with manageable size, with fewer than $100,000$ constraints, MetaPB adds probing features obtained from short runs of two complementary solvers: SCIP, which provides BnB probing features, and RoundingSAT, which provides CDCL probing features.

For larger instances (with more than $100,000$ constraints), probing becomes computationally expensive, so MetaPB relies only on static features. This ensures it can generate predictions for these larger instances, ensuring coverage of the entire benchmark collection.

Additionally, the best solution found during probing is preserved. Once feature extraction is complete, a pre-trained machine learning model predicts the best solver from the portfolio. Finally, the selected solver processes the PBO instance receiving the best solution found during the computation of the probing features, as a warm start.

[1] `https://github.com/amanda-salinas-pinto/MetaPB`

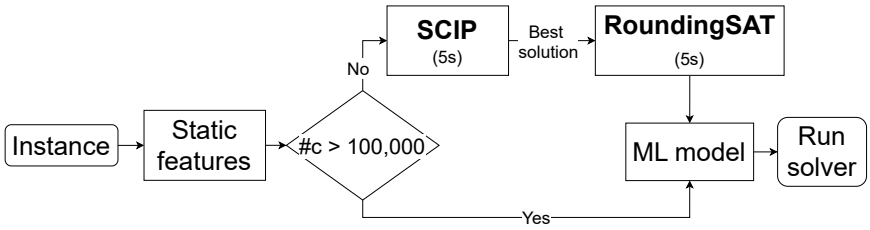


Figure 4.1: Overview of the MetaPB meta-solver pipeline. Static features are extracted for all instances, while Branch and Bound and CDCL probing features are dynamically collected for instances with fewer than 100,000 constraints (denoted as `#c` in the illustration). These features are then used to predict the most suitable solver for each instance.

Chapter 5

Experiments

This chapter presents the experimental evaluation of the proposed Automatic Algorithm Selection (AAS) framework for Pseudo-Boolean Optimization (PBO). The objective of this evaluation is twofold: (i) to identify the most effective meta-solver configurations through systematic validation, and (ii) to assess their generalization performance on an unseen competition benchmark.

The experimental setup is structured into two main phases. First, a validation phase is conducted using cross-validation on the training set to select the best-performing configurations. Second, the selected meta-solvers are evaluated on a held-out test set composed of instances from the 2024 PBO Competition, ensuring a fair and unbiased assessment.

5.1 Validation

The initial benchmark consists of 6,394 instances from the 2016 PBO Competition. From this set, 1,500 instances were randomly sampled to form the training dataset used for model development and hyperparameter tuning. To analyze solver behavior prior to learning, the performance of individual solvers on the training instances is first examined. Figure 5.1(a) reports normalized objective values (Equation 2.9) over a time horizon of 3600 seconds. Across the five evaluated time limits, Gurobi emerges as the Single Best Solver (SBS). Among open-source solvers, the default configuration of RoundingSAT achieves the strongest overall performance, constituting the SBS Open Source. Furthermore, the Virtual Best Solver composed exclusively of open-source solvers surpasses Gurobi after approximately 500 seconds, highlighting the complementarity within the open-source portfolio.

Figure 5.1(b) shows the number of instances for which each solver reaches the optimal solution over time. Gurobi achieves the highest overall optimality count. However, the VBS-Open-Source frequently exceeds Gurobi, confirming the potential of algorithm selection within the open-source portfolio. Additionally, although Mixed-Bag and SCIP obtain weaker normalized objective values on average, they

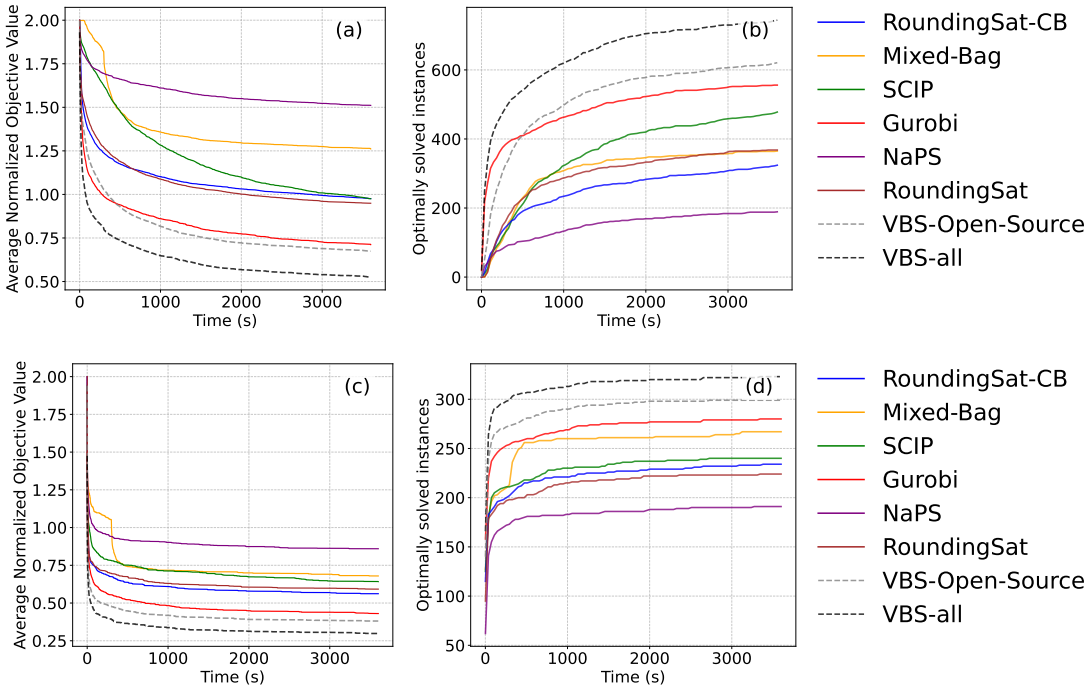


Figure 5.1: Preliminary analysis of solver performance on the training set (top figures) and test set (bottom figures). On the left, normalized objective values (Equation 2.9) over time. On the right, number of times a solver finds the optimal solution over time.

achieve optimal solutions more frequently, suggesting distinct performance profiles across solvers. All models are trained independently for each time limit in order to capture time-dependent solver behavior. Model selection is conducted using 10-fold cross-validation on the training set. Hyperparameter configurations are selected exclusively based on validation performance, ensuring that no information from the test set influences model development.

5.2 Testing

In this section, the evaluation focuses on (i) the best-performing meta-solvers identified during the validation phase and (ii) MetaPB trained exclusively on open-source solvers using the selected optimal configurations. Both approaches are assessed on 478 instances from the 2024 PBO Competition, which constitute a final evaluation benchmark curated by the competition organizers to reduce imbalance and ensure the inclusion of challenging and diverse instances. Figure 5.1(c) shows that Gurobi achieves the best normalized objective values, while the VBS-Open-Source often surpasses it, highlighting the strength of the open-source portfolio. Figure 5.1(d)

favors Gurobi in terms of the number of optimal solutions found across all time limits. Mixed-Bag approaches Gurobi despite weaker normalized scores, and the VBS-Open-Source again frequently outperforms it. Test instances also present better normalized objective values than training instances, suggesting that solvers could obtain good initial solutions more easily on this set.

5.3 Solver Portfolio

Table 5.1 presents the solvers included in the portfolio, together with the corresponding versions or commits employed to ensure full reproducibility of the experiments. The portfolio comprises both open-source and commercial solvers in order to capture complementary solving strategies and performance profiles.

Solver	**Version**	**Commit**
RoundingSat	pb24-log	01be2c2d
NaPS	v1.03a2	125435e0
Mixed-Bag	-	d144418
SCIP	SCIP Optimization Suite 9.2.1	-
Gurobi	10.0.0	-

Table 5.1: Solver versions.

To ensure reproducibility, table 5.2 reports the principal Python libraries and their versions used for model training and feature extraction from PBO instances.

Library	**Version**
scikit-learn	1.3.0
joblib	1.3.2
PySCIPOpt	4.3.0

Table 5.2: Python libraries used for training models and feature extraction.

5.4 Feature Sets

The static feature set proposed for PBO in [53] is adopted as the baseline representation. This set comprises ten features for linear PBO instances and captures structural characteristics such as instance size, objective function properties, and constraint-related metrics.

To assess the impact of dynamic information, the static features are combined with the probing features introduced in Section 4.2. The following feature configurations are considered:

- **st**: static features only.
- **st+CDCL**: static features combined with CDCL-based probing features.
- **st+BnB**: static features combined with Branch-and-Bound probing features.
- **all**: static features combined with all probing features.

Each feature configuration is evaluated across all machine learning models and learning formulations described in Chapter 4, enabling a systematic comparison of the contribution of dynamic probing information to algorithm selection performance.

5.5 Experimental results

5.5.1 Validation Results

This section presents the results obtained during the validation phase using cross-validation on the training set. Tables 5.3 and 5.4 presents cross-validated results on $\hat{m}_{s^t}$ and average number of optimally solved instances for the best-performing ML model (Random Forest) evaluated on the same feature sets and time limits.

The results reveal several key trends. First, feature sets combining probing and static features consistently achieve better results than just using static features. Most significantly, using all features provides the best results in most cases, suggesting complementarity between CDCL and Branch and Bound probing features.

Second, regression-based approaches systematically outperform multiclass classification on the $\hat{m}_{s^t}$ metric across all experimental conditions. This advantage likely arises from the regression model's ability to capture subtle differences in solver performance, avoiding the oversimplification inherent in discrete classification. The hybrid approach achieves the best overall performance, indicating that incorporating information about whether solvers can solve an instance optimally improves the regression model.

Even though regression achieved lower $\hat{m}_{s^t}$ scores, predicting the normalized objective value does not always lead to the highest number of optimally solved instances. The hybrid approach, which also predicts solver feasibility, overcomes this limitation.

Finally, the $\hat{m}_{s^t}$ metric tends to worsen for higher time budgets. This behavior aligns with expectations, as tighter time limits impose stronger penalties for unsolved instances, while performance differences between solvers gradually diminish during later stages of execution.

5.5.2 Feature Analysis

Figure 5.2 through Figure 5.6 present leave-one-out feature importance, $\Delta\hat{m}_{s^t}$ represents the difference between the baseline of using all features on the $\hat{m}_s^t$ across all

Table 5.3: Cross-validated $\hat{m}_{s^t}$ (mean ± SD, lower is better) of best ML models with different feature sets (st = Static, st+CDCL = Static + CDCL, st+BnB = Static + BnB, all = All features).

Task	**t**	**st**	**st+CDCL**	**st+BnB**	**all**
Multiclass	60	0.73 ± 0.03	0.72 ± 0.04	0.73 ± 0.03	**0.72 ± 0.04**
	100	0.78 ± 0.05	0.75 ± 0.06	0.78 ± 0.06	**0.74 ± 0.06**
	300	0.83 ± 0.05	0.80 ± 0.05	0.83 ± 0.05	**0.79 ± 0.04**
	600	0.84 ± 0.06	0.81 ± 0.05	0.82 ± 0.06	**0.82 ± 0.06**
	3600	0.90 ± 0.06	0.90 ± 0.06	**0.88 ± 0.06**	0.89 ± 0.06
Hybrid (best α)	60	0.28 ± 0.04	0.28 ± 0.04	0.28 ± 0.04	**0.28 ± 0.04**
	100	0.35 ± 0.07	0.31 ± 0.08	0.34 ± 0.06	**0.31 ± 0.06**
	300	0.44 ± 0.08	0.39 ± 0.10	0.43 ± 0.09	**0.38 ± 0.08**
	600	0.46 ± 0.05	0.44 ± 0.06	0.44 ± 0.05	**0.43 ± 0.05**
	3600	0.81 ± 0.09	0.79 ± 0.08	**0.79 ± 0.08**	0.79 ± 0.09
Multilabel ($\alpha = 0$)	60	0.83 ± 0.05	0.80 ± 0.07	0.83 ± 0.07	**0.79 ± 0.09**
	100	0.78 ± 0.11	**0.77 ± 0.14**	0.78 ± 0.11	0.77 ± 0.12
	300	0.77 ± 0.13	**0.72 ± 0.11**	0.77 ± 0.11	0.74 ± 0.10
	600	0.90 ± 0.21	**0.84 ± 0.15**	0.89 ± 0.20	0.87 ± 0.19
	3600	0.83 ± 0.08	0.83 ± 0.10	**0.82 ± 0.09**	0.85 ± 0.09
Regression ($\alpha = 1$)	60	0.33 ± 0.04	0.33 ± 0.04	0.32 ± 0.05	**0.31 ± 0.05**
	100	0.38 ± 0.06	0.37 ± 0.07	0.38 ± 0.07	**0.35 ± 0.07**
	300	0.50 ± 0.11	0.41 ± 0.12	0.49 ± 0.09	**0.41 ± 0.11**
	600	0.51 ± 0.05	0.48 ± 0.09	0.51 ± 0.05	**0.46 ± 0.06**
	3600	0.83 ± 0.07	0.82 ± 0.08	0.83 ± 0.07	**0.80 ± 0.07**

timesteps. Features are grouped into three categories: **Static**, **CDCL-based**, and **BnB-based**. Positive $\Delta\hat{m}_{s^t}$ values indicate a decrease in predictive performance when the feature is removed, representing higher importance.

At the early stage (time limit of 60 seconds), predictive performance is primarily influenced by CDCL-based features. Metrics related to learned clauses, decisions per conflict, average LBD, and conflict resolution steps exhibit the highest $\Delta\hat{m}_{s^t}$ values, indicating that early predictions are strongly dependent on clause-learning dynamics and conflict structures. BnB features such as the mean objective value and the number of solutions found have a moderate influence, while static structural features contribute only minimally. This pattern suggests that early performance estimation relies heavily on conflict-driven solver behavior rather than global structural or objective-related properties.

By timestep 100, CDCL features remain dominant but the difference relative to other feature groups narrows. Decisions per conflict and trail pops per conflict con-

Table 5.4: Cross-validated number of optimally solved instances (mean ± SD, higher is better) of best ML models with different feature sets (st = Static, st+CDCL = Static + CDCL, st+BnB = Static + BnB, all = All features).

Task	**t**	**st**	**st+CDCL**	**st+BnB**	**all**
Multiclass	60	27.70 ± 5.53	**27.90 ± 5.03**	27.90 ± 5.37	27.80 ± 5.08
	100	33.00 ± 6.10	**33.70 ± 6.36**	33.20 ± 6.08	33.60 ± 5.99
	300	43.00 ± 5.78	42.70 ± 5.51	**43.10 ± 5.84**	42.80 ± 5.62
	600	46.20 ± 5.15	46.60 ± 5.24	**46.80 ± 4.69**	46.40 ± 5.71
	3600	59.60 ± 5.78	59.10 ± 5.79	**59.90 ± 5.66**	59.00 ± 5.73
Hybrid (best α)	60	26.90 ± 5.70	**28.20 ± 5.38**	27.00 ± 5.57	27.00 ± 5.35
	100	33.70 ± 5.57	33.70 ± 6.10	33.30 ± 6.10	**34.00 ± 5.48**
	300	42.10 ± 5.32	42.70 ± 5.55	41.70 ± 5.35	**43.30 ± 5.88**
	600	45.70 ± 4.65	45.60 ± 5.10	46.10 ± 4.32	**46.40 ± 5.35**
	3600	67.60 ± 7.27	67.30 ± 7.40	66.60 ± 6.87	**68.40 ± 6.96**
Multilabel ($\alpha = 0$)	60	28.20 ± 4.66	**28.70 ± 4.94**	28.40 ± 4.88	28.60 ± 5.02
	100	34.20 ± 5.86	35.30 ± 5.93	34.50 ± 5.84	**35.40 ± 5.68**
	300	43.90 ± 5.63	44.30 ± 5.62	43.70 ± 5.69	**44.30 ± 5.57**
	600	48.20 ± 5.56	48.40 ± 5.75	48.40 ± 5.75	**48.50 ± 6.00**
	3600	68.00 ± 6.83	67.80 ± 6.97	68.20 ± 6.94	**68.40 ± 7.09**
Regression ($\alpha = 1$)	60	24.20 ± 4.56	**24.40 ± 4.84**	23.70 ± 5.42	23.70 ± 4.63
	100	28.80 ± 4.71	**30.10 ± 4.76**	29.40 ± 4.36	30.00 ± 4.69
	300	38.30 ± 5.93	39.50 ± 4.94	38.00 ± 4.98	**39.80 ± 5.23**
	600	42.70 ± 3.95	43.50 ± 5.14	42.90 ± 3.53	**43.60 ± 4.82**
	3600	65.00 ± 6.00	63.80 ± 6.97	**65.30 ± 5.88**	64.7 ± 7.64

tinue to rank highly, while structural constraint features, such as the percentage of quaternary constraints, gain importance. BnB-related metrics, including the optimality gap and ratios of positive objective terms, also become more relevant. Static features, such as the number of variables and constraints, increase their contribution compared to the earliest phase. Overall, the model begins integrating both search behavior and structural composition, indicating that conflict activity remains central but objective and constraint structures start to play a more significant role in predictive performance.

At the mid stage (timestep 300), feature importance becomes more evenly distributed across all categories. Static features, including the percentage of terms in constraints and the binary constraint ratio, rise in prominence. BnB-related metrics, such as feasible leaves explored and objective-related features, also gain importance. CDCL-based features remain relevant but no longer dominate the predictions. Some features display slightly negative $\Delta\hat{m}_{s^t}$ values. This shift indicates that solver

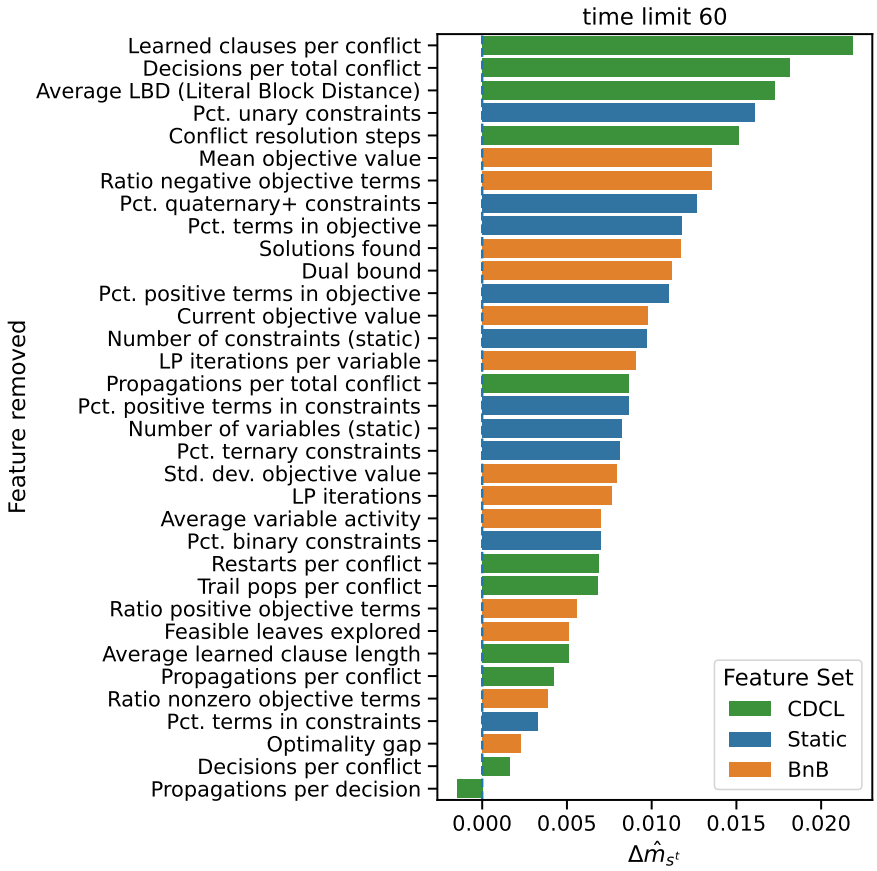


Figure 5.2: Feature importance at timestep 60. CDCL-based features dominate early predictions.

behavior depends less exclusively on clause-learning dynamics and more on global structural and objective characteristics.

During the late-mid stage (timestep 600), BnB features dominate the importance rankings. Metrics related to LP iterations per variable, the percentage of terms in constraints, and standard deviation of objective values rise to the top, while CDCL features remain present but are no longer leading. Static structural features maintain moderate influence, highlighting the solver's shift toward branch-and-bound behavior. Predictive performance is now primarily driven by optimization-specific dynamics rather than conflict-learning.

By the final stage (timestep 3600), static features become more prominent. The most important predictors include the percentage of positive terms in constraints and in the objective, the ratio of binary constraints, average variable activity, propagations per conflict, and the total number of variables, while objective-related and other Branch-and-Bound features remain relevant. This indicates that the solver has reached a more stable phase in which performance is shaped primarily by structural

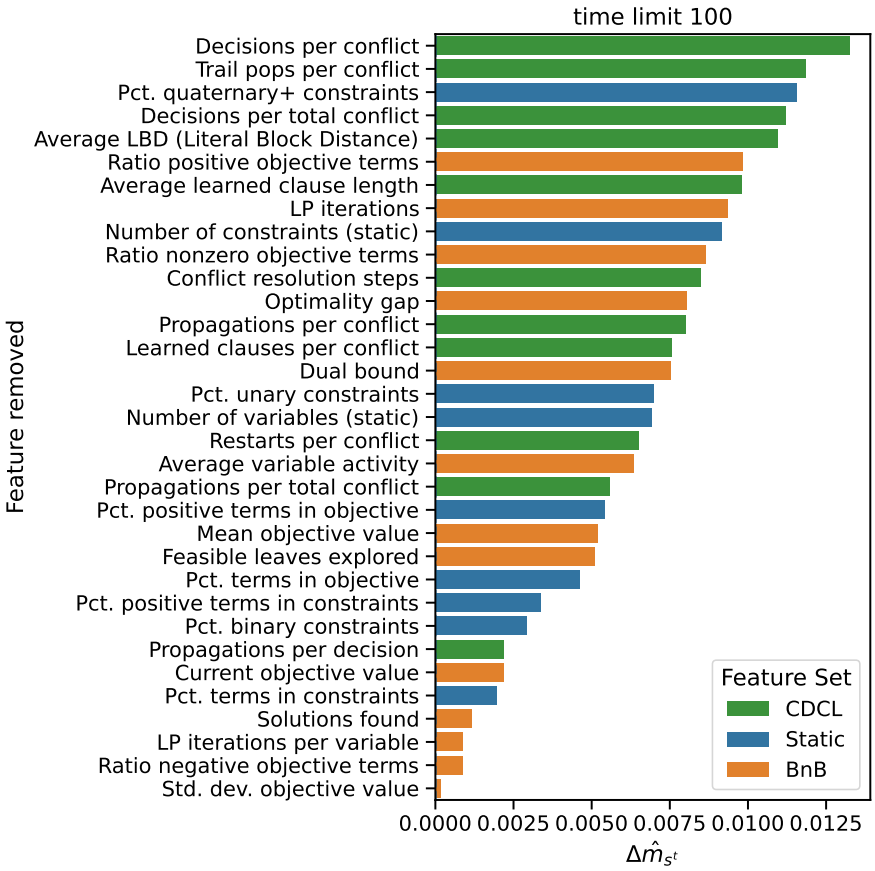


Figure 5.3: Feature importance at timestep 100. CDCL features remain dominant, with increasing structural and BnB influence.

properties, whereas probing-based features that reflect early search behavior become less informative over time.

5.5.3 Impact of the Weighting Parameter α

Figure 5.7 illustrates how the performance changes in terms of the $\hat{m}_{s^t}$ metric and the number of optimally solved instances as the hybrid model's parameter α is varied for each feature set. When $alpha = 1$, the model relies exclusively on the regression model, which results in fewer optimally solved instances. Incorporating some information about the number of optimally solved instances improves both metrics compared to using only regression. However, when relying entirely on the multilabel approach ($\alpha = 0$), the $\hat{m}_{s^t}$ metric worsens despite solving more instances. This indicates that there is no clear correlation between the number of optimally solved instances and the $\hat{m}_{s^t}$ metric. It also suggests that intermediate values of α are more effective for achieving both lower $\hat{m}_{s^t}$ values and a higher number of optimally solved

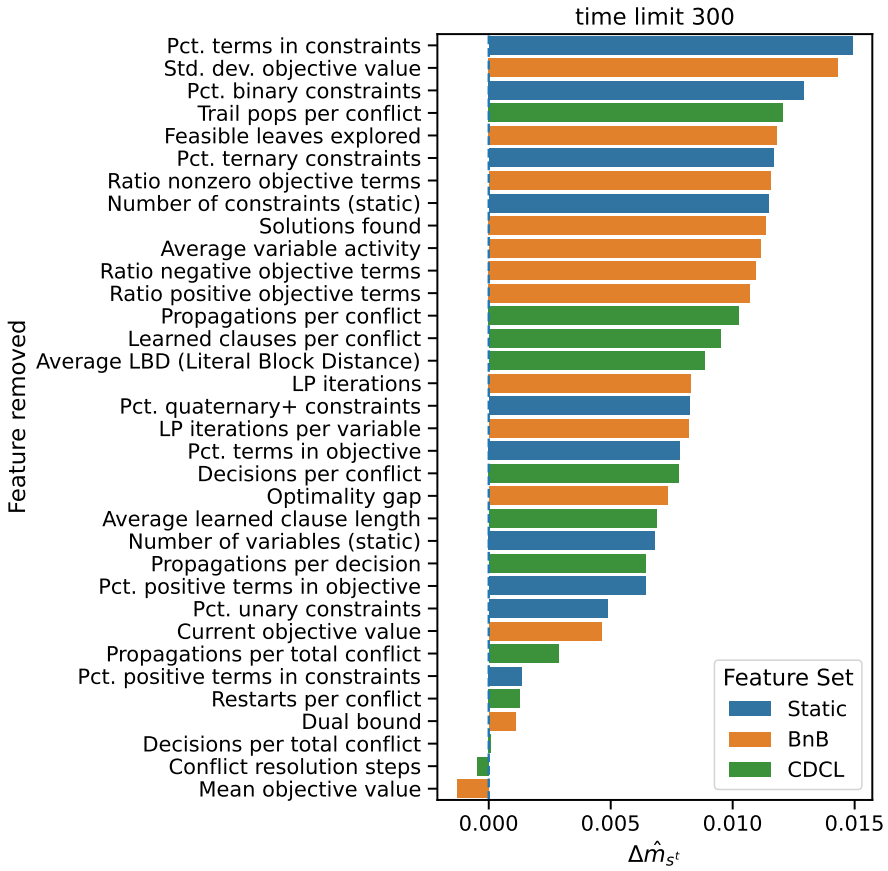


Figure 5.4: Feature importance at timestep 300. Importance is more evenly distributed across feature categories.

instances across all feature sets. We can see this behavior in all the feature sets. In this work, we prioritize minimizing the $\hat{m}_{s^t}$ metric and set $\alpha = 0.8$, but if the main objective is to maximize the number of optimally solved instances, a lower α can be chosen.

5.5.4 Results on the Test Set (2024 PBO competition)

The selected configurations are evaluated on the independent test set of the 2024 PBO Competition. Two evaluation scenarios are considered: first, performance using the complete solver portfolio without accounting for probing overhead; second, the full evaluation of MetaPB, which incorporates feature extraction cost and relies exclusively on open-source solvers.

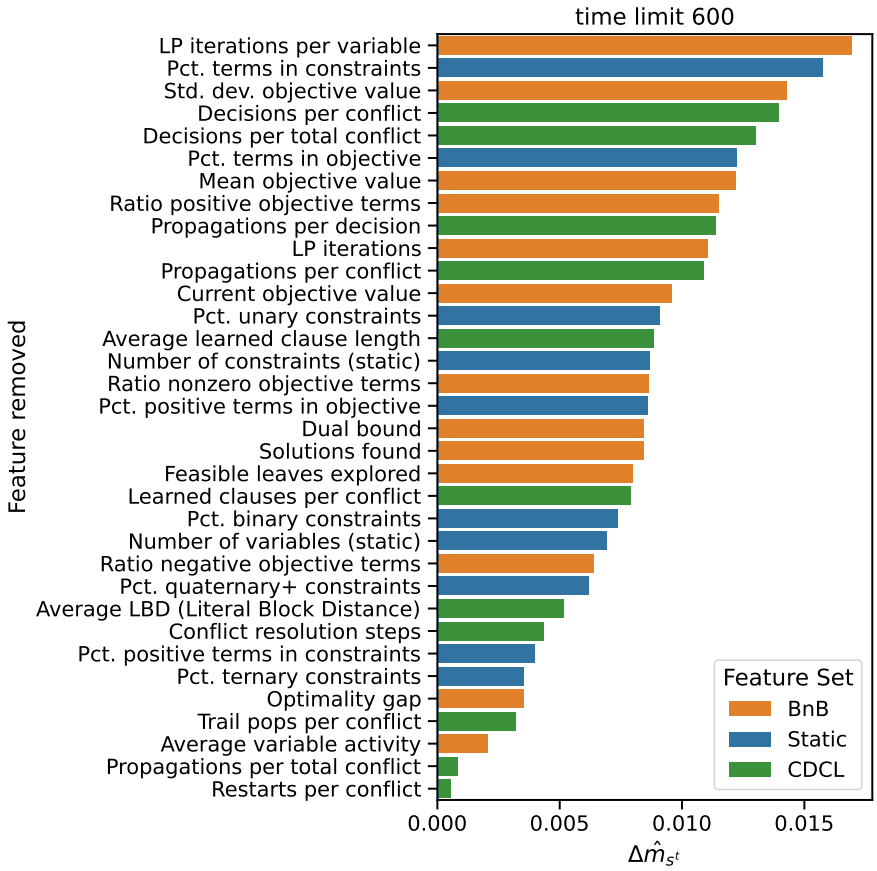


Figure 5.5: Feature importance at timestep 600. BnB features dominate, reflecting optimization-driven solver behavior.

Performance using the complete portfolio

The configurations selected during the validation phase are evaluated on the test set of the 2024 PBO Competition. Results are reported under two metrics: a) the $\hat{m}_{s^t}$ optimization metric (Table 5.5) and b) the number of optimally solved instances within the time limit (Table 5.6).

Table 5.5 shows that combining static features with probing features consistently improves the $\hat{m}s^t$ metric compared to using static features alone. Both the **st+CDCL** and **st+BnB** feature sets outperform **st**, and in most cases, using all features provides the best performance, except at the longest timeout, where **st+BnB** performs slightly better. All feature sets also improve upon the SBS (Gurobi), achieving $\hat{m}s^t$ values below 1.

As shown in Table 5.6, the meta-solver consistently solves more instances optimally than the SBS (Gurobi) across all time marks. Adding probing features generally increases the number of optimally solved instances compared to using only

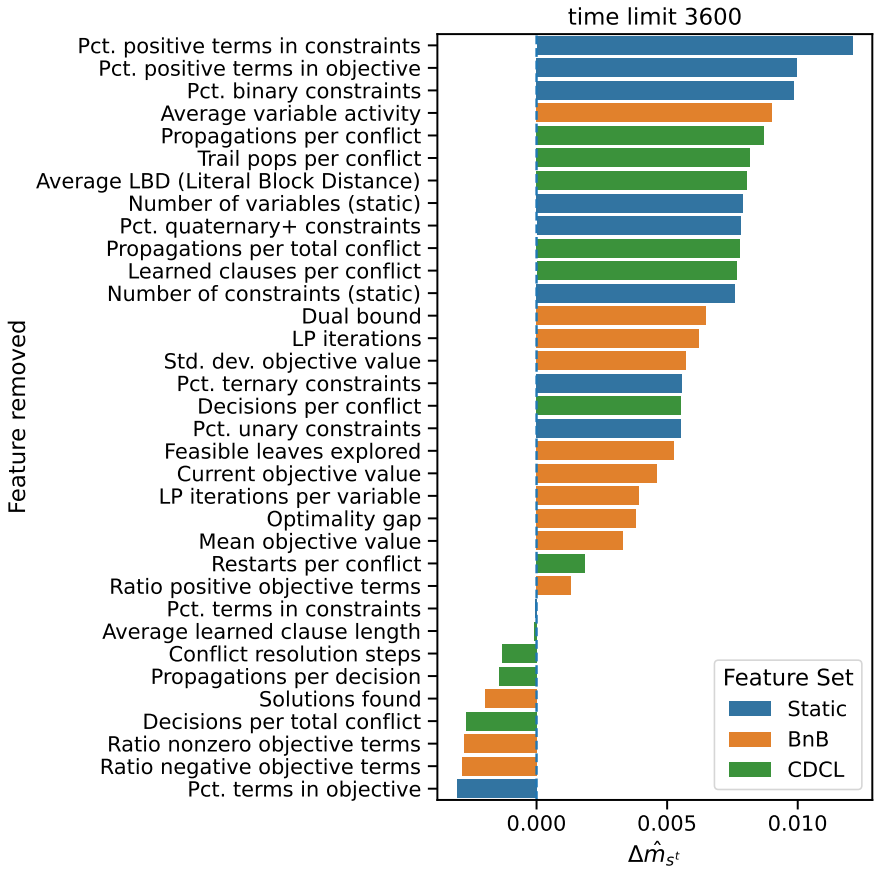


Figure 5.6: Feature importance at timestep 3600. Static features dominate long-term predictions; probing features are less influential.

static features. The benefit of CDCL features is more evident at shorter time marks, whereas BnB features contribute more as the time budget increases.

As a limitation of this study, it is observed that the $\hat{m}_{s^t}$ values generally get worse for increasing time marks for all feature sets, suggesting that AAS for PBO is more challenging as the time limit increases.

MetaPB Evaluation

MetaPB, the proposed meta-solver that combines static and probing features for solver selection using only open-source solvers, is evaluated on the test set of the 2024 PBO Competition. Performance is compared against three baselines: Gurobi (commercial solver), Mixed-Bag (winner of the 2024 PBO Competition), and RoundingSat (Single Best Open-Source Solver). Tables 5.7 and 5.8 report performance under two metrics: the accumulated normalized objective value m_{s^t} (Equation 2.10) and the total number of optimally solved instances.

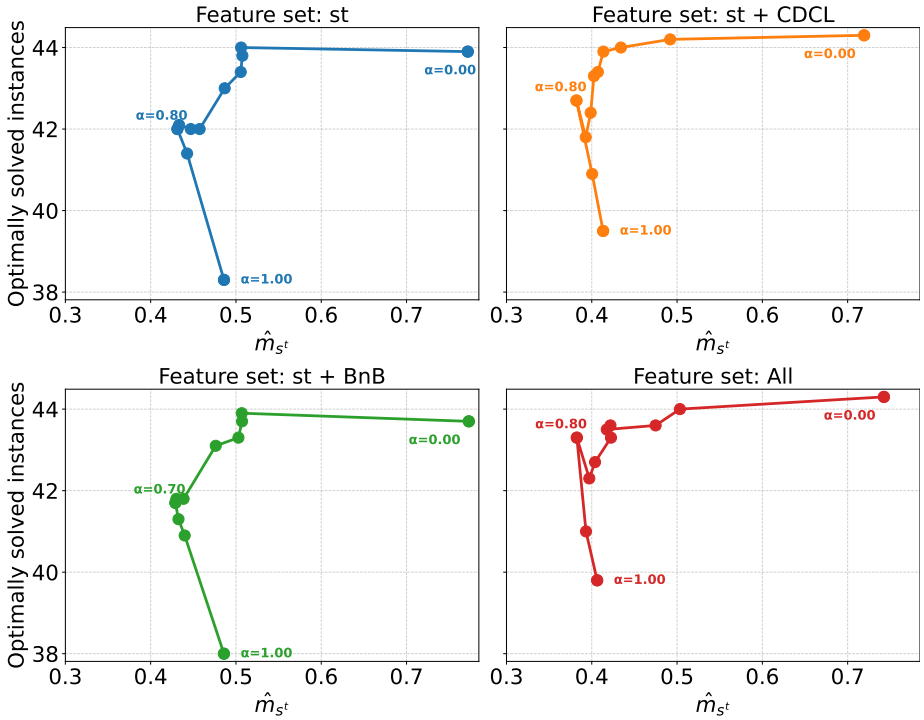


Figure 5.7: Effect of varying α in the hybrid approach across the four feature sets under a 300 second time limit. For each, we report the number of optimally solved instances and the $\hat{m}_{s^t}$ metric. The parameter α controls the trade-off between the regression prediction $\widehat{n}_a(i,t)$ and the multilabel prediction $\widehat{p}_a(i,t)$. Intermediate values of α lead to better performance.

The statistical analysis (Table 5.7) reveals a clear performance hierarchy. The two strongest MetaPB configurations are **all** and **st**+**BnB**. These variants are statistically equivalent across most time limits ($t = 60, 100, 600$), belonging to the same statistical tier. The **st**+**BnB** configuration is a more efficient feature set for enhancing the static approach because the cost-to-benefit ratio of **BnB** features is inherently superior to CDCL-based features. This is evidenced by two points: 1) **all** is only statistically superior to **st**+**BnB** at $t = 300$; and 2) the high probing overhead of **CDCL** features causes the **st**+**CDCL** variant to be statistically worse than the base **st** variant at short limits ($t = 60, 100$). The performance of **st**+**CDCL** only recovers to join the same tier as **st** at longer limits ($t \geq 300$) once the feature cost is amortized.

In comparison with baselines, MetaPB remains highly competitive. The best MetaPB variants (**all** and **st**+**BnB**) are statistically equivalent to **Gurobi** in most time limits and significantly outperform **st** features alone. While **st**+**CDCL** is statistically equivalent to the RoundingSat baseline at $t = 60, 100$, all MetaPB variants are ranked superior to RoundingSat from $t = 300$ onward. Importantly, MetaPB consistently achieves results statistically superior to Mixed-Bag, the PBO 2024 win-

Table 5.5: $\hat{m}_{s^t}$ scores (lower is better) for different feature sets, including statistical ranking on the test set. Superscript numbers denote statistical tiers (pairwise Wilcoxon $p < 0.05$). Tier 1 is the best-performing group. Solvers sharing the same tier number are statistically equivalent. All feature sets are statistically better than Gurobi at all timesteps.

t	**st**	**st+CDCL**	**st+BnB**	**all**
60	0.448^{4}	0.344^{2}	0.413^{3}	$\mathbf{0.316^{1}}$
100	0.632^{4}	0.465^{2}	0.556^{3}	$\mathbf{0.420^{1}}$
300	0.688^{4}	0.606^{2}	0.633^{3}	$\mathbf{0.595^{1}}$
600	0.726^{3}	0.648^{2}	0.686^{2}	$\mathbf{0.627^{1}}$
3600	0.852^{2}	0.829^{2}	$\mathbf{0.789^{1}}$	0.799^{1}

Table 5.6: Number of optimally solved instances per solver and meta-solver (algorithm selector), including *all* solvers in the portfolio.

t	**st**	**st+CDCL**	**st+BnB**	**all**	**Gurobi^t**
60	254	254	254	**261**	231
100	262	270	270	**274**	241
300	271	276	**277**	**277**	254
600	277	278	**280**	**280**	261
3600	298	295	**302**	300	280

ner, which is grouped into the lowest statistical tiers.

Regarding the number of optimally solved instances (Table 5.8), all MetaPB variants generally outperform both the SBS-Open-Source and Mixed-Bag baselines. The **all** feature set is the best performer, surpassing Gurobi at every timestep except $t = 600$. These results confirm MetaPB's effectiveness for solver selection, demonstrating that it achieves high solution quality and strong performance in solving instances to optimality. This results indicate that MetaPB would have won the Optimization track of the 2024 PBO Competition.

Table 5.7: Accumulated normalized objective value m_{s^t} (lower is better) and statistical ranking on the test set for MetaPB variants and baselines. Superscript numbers denote statistical tiers (pairwise Wilcoxon $p < 0.05$). Tier 1 is the best-performing group. Solvers sharing the same tier number are statistically equivalent.

t	st	st+CDCL	st+BnB	all	MB	Gu	RS
60	337^{2}	359^{3}	323^{1}	323^{1}	738^{4}	336^{1}	355^{3}
100	333^{1}	340^{2}	326^{1}	327^{1}	732^{3}	316^{1}	352^{2}
300	291^{3}	297^{3}	281^{2}	272^{1}	656^{5}	285^{1}	330^{4}
600	274^{3}	269^{3}	253^{2}	259^{1}	453^{5}	257^{1}	306^{4}
3600	236^{3}	234^{3}	230^{2}	234^{3}	424^{5}	211^{1}	288^{4}

Table 5.8: Number of optimally solved instances on the test set for MetaPB variants and baselines: Mixed-Bag (MB), Gurobi (Gu), and RoundingSat (RS).

t	st	st+CDCL	st+BnB	all	MB	Gu	RS
60	225	220	234	**239**	194	231	180
100	232	227	239	**243**	202	241	187
300	244	241	250	**254**	216	**254**	202
600	253	253	260	257	256	**261**	217
3600	280	279	281	**284**	267	280	224

Chapter 6

Conclusions and future work

This thesis investigated the role of dynamic features in Automatic Algorithm Selection (AAS) for Pseudo-Boolean Optimization (PBO), guided by the hypothesis that incorporating solver behavior from early execution stages would reduce the $\hat{m}_{s^t}$ metric, thereby closing the performance gap with the Virtual Best Solver (VBS). The experimental results provide compelling support for this hypothesis. Across all validation and test scenarios, feature sets that integrated static and probing information consistently achieved lower $\hat{m}_{s^t}$ values than those relying on static features alone. This confirms that dynamic information offers a measurable and significant benefit for meta-solver performance.

Feature analysis showed that relevance is strongly time-dependent. CDCL-based features dominate at short time limits, reflecting the importance of early conflict behavior, while Branch-and-Bound (BnB) and structural features become increasingly important at medium and long time limits. Overall, the best performance was achieved by combining all static and dynamic feature families, confirming that these features capture complementary aspects of solver behavior.

From a machine learning perspective, the results challenge the predominance of classification-based approaches in the AAS for PBO literature. Regression-based formulations consistently outperformed direct multiclass classification. By predicting solver performance, this approach produces more robust decisions than directly predicting a single "best solver" label.The hybrid formulation, which integrates regression predictions with information about solver optimality, achieved the best overall balance between normalized objective quality and the number of optimally solved instances. Intermediate values of the weighting parameter α consistently provided the most favorable compromise, indicating that performance prediction and optimality likelihood capture complementary aspects of solver behavior.

Building on these insights, MetaPB was developed as an open-source meta-solver integrating the best-performing feature sets and learning formulation identified in this study. When evaluated on the 2024 PBO Competition test set, MetaPB outperformed every individual solver in its portfolio, surpassed the competition winner, and achieved performance competitive with the commercial solver Gurobi.

Despite these advances, several limitations remain. The $\hat{m}_{s^t}$ metric tends to deteriorate as the time limit increases, indicating that algorithm selection becomes more challenging under longer time budgets where solver performance differences narrow. Furthermore, probing features introduce overhead that must be amortized over the solving time, creating a trade-off between information richness and computational cost.

Future research should explore dynamic feature representations that evolve along with solver behavior to improve long-term prediction stability. Increasing the diversity of solvers in the portfolio may also enhance generalization, giving AAS greater adaptability. Moreover, analyzing the time required to extract probing features would help clarify the trade-off between feature extraction cost and predictive performance. The observed shifts in solver performance over time suggest strong potential for adaptive scheduling strategies, where solvers are dynamically selected based on runtime progress to maximize efficiency.

Bibliography

[1] M. Alviano, C. Dodaro, J. Marques-Silva, and F. Ricca. Optimum stable model search: algorithms and implementation. *Journal of Logic and Computation*, 30(4):863–897, 8 2015.

[2] D. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. 1 2007.

[3] J. Berg, E. Demirović, and P. J. Stuckey. *Core-Boosted Linear search for incomplete MaxSAT*. 1 2019.

[4] D. L. Berre and A. Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability Boolean Modeling and Computation*, 7(2-3):59–64, 7 2010.

[5] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt. CaDiCaL 2.0. In A. Gurfinkel and V. Ganesh, editors, *Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I*, volume 14681 of *Lecture Notes in Computer Science*, pages 133–152. Springer, 2024.

[6] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt. CaDiCaL, Gimsatul, IsaSAT and Kissat entering the SAT Competition 2024. In M. Heule, M. Iser, M. Järvisalo, and M. Suda, editors, *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, volume B-2024-1 of *Department of Computer Science Report Series B*, pages 8–10. University of Helsinki, 2024.

[7] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021.

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

[9] S. Bolusani, M. Besançon, K. Bestuzheva, A. Chmiela, J. Dionísio, T. Donkiewicz, J. van Doornmalen, L. Eifler, M. Ghannam, A. Gleixner, C. Graczyk, K. Halbig, I. Hedtke, A. Hoen, C. Hojny, R. van der Hulst, D. Kamp, T. Koch, K. Kofler, J. Lentz, J. Manns, G. Mexi, E. Mühmer, M. E. Pfetsch, F. Schlösser, F. Serrano, Y. Shinano, M. Turner, S. Vigerske, D. Weninger, and L. Xu. The SCIP Optimization Suite 9.0. Technical report, Optimization Online, February 2024.

[10] B. Borchers and J. Furman. A Two-Phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2(4):299–306, 12 1998.

[11] L. Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[12] S. Ceria, C. Cordier, H. Marchand, and L. A. Wolsey. Cutting planes for integer programs with general integer variables. *Mathematical Programming*, 81(2):201–214, 4 1998.

[13] G. Clarke and J. W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12(4):568–581, 8 1964.

[14] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery.

[15] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962.

[16] J. Devriendt, A. Gleixner, and J. Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 26(1-4):26–55, 1 2021.

[17] J. Devriendt, S. Gocht, E. Demirović, J. Nordström, and P. J. Stuckey. Cutting to the core of pseudo-boolean optimization: Combining core-guided search with cutting planes reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3750–3758, 2021.

[18] N. Eén and N. Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.

[19] I. El Naqa and M. J. Murphy. *What Is Machine Learning?*, pages 3–11. Springer International Publishing, Cham, 2015.

[20] J. Elffers and J. Nordström. Divide and conquer: Towards faster pseudo-boolean solving. In *IJCAI*, volume 18, pages 1291–1299, 2018.

[21] J. Elffers and J. Nordström. Divide and conquer: Towards faster pseudo-boolean solving. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1291–1299. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[22] N. Eén and N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability Boolean Modeling and Computation*, 2(1-4):1–26, 3 2006.

[23] E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination, consistency properties. 1951.

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

[25] Z. Fu and S. Malik. On solving the partial max-sat problem. In A. Biere and C. P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006*, pages 252–265, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[26] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1 1979.

[27] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In C. Baral, G. Brewka, and J. Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning*, pages 260–265, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[28] A. Gleixner, L. Gottwald, and A. Hoen. Papilo: A parallel presolving library for integer and linear optimization with multiprecision support. *INFORMS Journal on Computing*, 35(6):1329–1341, 2023.

[29] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers Operations Research*, 13(5):533–549, 1986. Applications of Integer Programming.

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

[31] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024.

[32] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning*. 1 2009.

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

[34] I. I. Huerta, D. A. Neira, D. A. Ortega, V. Varas, J. Godoy, and R. J. As'in Ach'a. Anytime automatic algorithm selection for knapsack. *Expert Syst. Appl.*, 158:113613, 2020.

[35] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and intelligent optimization: 5th international conference, LION 5, rome, Italy, January 17-21, 2011. selected papers 5*, pages 507–523. Springer, 2011.

[36] F. Hutter, D. A. D. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002*, pages 233–248, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

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

[38] C. Jabs, J. Berg, and M. Järvisalo. Pb-oll-rs and mixed-bag in pseudo-boolean competition 2024. 2024.

[39] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, 27(1):3–45, 2019.

[40] L. Kotthoff, P. Kerschke, H. Hoos, and H. Trautmann. Improving the state of the art in inexact tsp solving using per-instance algorithm selection. In *Learning and Intelligent Optimization: 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers 9*, pages 202–217. Springer, 2015.

[41] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497, 7 1960.

[42] Z. Lei, S. Cai, C. Luo, and H. Hoos. Efficient local search for pseudo boolean optimization. In C.-M. Li and F. Manyà, editors, *Theory and Applications of Satisfiability Testing – SAT 2021*, pages 332–348, Cham, 2021. Springer International Publishing.

[43] M. Lindauer, J. N. van Rijn, and L. Kotthoff. The algorithm selection competitions 2015 and 2017. *Artificial Intelligence*, 272:86–100, 2019.

[44] A. Loreggia, Y. Malitsky, H. Samulowitz, and V. Saraswat. Deep learning for algorithm portfolios. In *Proceedings of the aaai conference on artificial intelligence*, volume 30, 2016.

[45] V. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for weighted boolean optimization. 04 2009.

[46] J. Marques-Silva and K. Sakallah. GRASP: a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 5 1999.

[47] S. Martello and P. Toth. Knapsack problems: Algorithms and computer implementations. *Journal of the Operational Research Society*, 42(6):513, 6 1991.

[48] R. Martins, V. Manquinho, and I. Lynce. *Open-WBO: A modular MaxSAT Solver,*. 1 2014.

[49] J. Nordström. Pseudo-boolean solving and optimization. Satisfiability: Theory, Practice, and Beyond Boot Camp, Feb. 2021. Workshop lecture, February 4, 2021.

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

[51] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham. Understanding random sat: Beyond the clauses-to-variables ratio. In *Principles and Practice of Constraint Programming–CP 2004: 10th International Conference, CP 2004, Toronto, Canada, September 27-October 1, 2004. Proceedings 10*, pages 438–452. Springer, 2004.

[52] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, volume 32. 01 1982.

[53] C. Pezo, D. S. Hochbaum, J. Godoy, and R. J. As'in Ach'a. Automatic algorithm selection for pseudo-boolean optimization with given computational time limits. *Comput. Oper. Res.*, 173:106836, 2025.

[54] J. R. Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier, 1976.

[55] O. Roussel. Pseudo Boolean Competition 2024, 2024.

[56] 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*, 98(6):1121–1127, 2015.

[57] A. Salinas-Pinto, C. Pezo-Vergara, D. Hochbaum, B. Dilkina, R. Nanculef, and R. Asín-Achá. Probing features for automatic algorithm selection for pseudo-boolean optimization. In *at International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*, 2026.

[58] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, page 440–446. AAAI Press, 1992.

[59] H. Shavit and H. H. Hoos. Revisiting satzilla features in 2024. In *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*, pages 27–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[60] K. Smith-Miles, D. Baatar, B. Wreford, and R. Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12–24, 2014.

[61] N. Sörensson and N. Eén. Minisat 2.1 and minisat++ 1.0-sat race 2008 editions. *SAT*, page 31, 2009.

[62] H. P. Williams, G. L. Nemhauser, and L. A. Wolsey. Integer and combinatorial optimization. *Journal of the Operational Research Society*, 41(2):177, 2 1990.

[63] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

[64] L. Xu, F. Hutter, J. Shen, H. H. Hoos, and K. Leyton-Brown. Satzilla2012: Improved algorithm selection based on cost-sensitive classification models. *Proceedings of SAT Challenge*, pages 57–58, 2012.