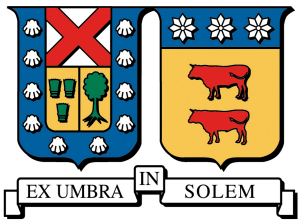
Universidad Técnica Federico Santa María
Departamento de Informática
Valparaíso, Chile



Backbone-based Predict and Search for Pseudo-Boolean Optimization

Bryan Jesús Alvarado Ulloa

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

Profesor Guía: Dr. Roberto Asín
Profesor Co-Guía: Dr. Ricardo Ñanculef
Profesor Correferente Interno: Dra. Elizabeth Montero
Profesor Correferente Externo: Dr. Albert Oliveras
Presidente Comisión: Dra. Elizabeth Montero

12 de marzo de 2026

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: Backbone-based Predict and Search for Pseudo-Boolean Optimization

Nombre del candidato(a): Bryan Jesús Alvarado Ulloa

Carrera / Grado: Magíster en Ciencias de la Ingeniería Informática

Campus: Casa Central Valparaiso **;** **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: 12/03/2026 ; **Firma:** 

Estudiante o Candidato(a):

Fecha: 12/03/2026 ; **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

First, I would like to express my deepest gratitude to my advisor, Roberto Asín, for his invaluable guidance and mentorship. I am profoundly grateful for his role in introducing me to the world of scientific research and for providing constant, insightful feedback that shaped this work. I would also like to thank him for the extraordinary opportunity to participate in research internships at UC Berkeley, an experience that significantly broadened my academic horizons.

I want to thank my parents Rosa Ulloa and Patricio Alvarado, who have been my constant pillar of support. Despite the complexities of my field of study, their unwavering interest in my progress and their constant encouragement have been the fuel that kept me moving forward.

I gratefully acknowledge the financial support provided by ANID-Subdirección de Capital Humano/Magíster Nacional/2025 - 22252175. This work was made possible thanks to the scholarship granted by the agency.

Finally, I would like to thank the Universidad Técnica Federico Santa María for the scholarship awarded to pursue my Master's degree, providing me with the resources and environment necessary to complete this research.

Abstract

This thesis investigates the integration of Machine Learning (ML) techniques to accelerate the solution of combinatorial optimization problems, focusing on instances formulated under the Pseudo-Boolean Optimization (PBO) paradigm. In PBO, constraints are expressed as linear inequalities over Boolean variables, and the objective function is defined as a linear combination of these variables.

Although ML-based approaches have been increasingly applied to combinatorial optimization, their direct integration into solvers may introduce prediction errors that lead to infeasible solutions or degradation in solution quality. The Predict-and-Search (PaS) framework addresses this limitation by using ML predictions to generate additional constraints that reduce the search space before invoking the solver. However, existing PaS implementations have been primarily developed for Mixed Integer Linear Programming (MILP) and typically rely on heuristically generated labels derived from near-optimal solutions during training, thereby sacrificing optimality guarantees.

This thesis introduces *Backbone-based Predict and Search* (BackPaS), a specialized adaptation of the PaS framework for PBO. The main contribution consists of redefining the prediction task as the identification of the backbone of an instance, defined as the set of variables whose assignments remain fixed across all optimal solutions. Correctly predicting these variables enables a reduction of the solver's search space without compromising optimality. To achieve this, a Graph Neural Network (GNN) architecture is designed based on a literal-level bipartite graph representation, formulating backbone membership and polarity prediction as a multi-class classification problem. The predictions are incorporated into the solver through a parameterized adaptive trust region that dynamically adjusts the search space.

Resumen

Esta tesis aborda la integración de técnicas de aprendizaje automático (ML, *Machine Learning*) para acelerar la resolución de problemas combinatorios de optimización, específicamente aquellos formulados bajo el paradigma de Optimización Pseudo-Booleana (PBO, *Pseudo-Boolean Optimization*). En PBO, las restricciones se expresan como desigualdades lineales sobre variables booleanas, junto con una función objetivo también definida como combinación lineal de dichas variables.

Si bien diversos enfoques de ML han sido aplicados a problemas combinatorios, su incorporación directa en solucionadores puede generar errores de predicción que conduzcan a soluciones infactibles o a una pérdida de calidad respecto del óptimo. El marco de trabajo *Predict-and-Search* (PaS) surge como una estrategia para mitigar estos problemas, utilizando predicciones de un modelo de ML para añadir restricciones adicionales que reducen el espacio de búsqueda antes de invocar el solucionador. No obstante, las implementaciones existentes de PaS han sido desarrolladas principalmente para Programación Entera Mixta (MILP, *Mixed Integer Linear Programming*) y suelen entrenar los modelos con etiquetas heurísticas derivadas de soluciones cercanas al óptimo, sacrificando garantías de optimalidad.

En esta tesis se propone *Backbone-based Predict and Search* (BackPaS), una adaptación especializada del marco PaS al contexto de PBO. La contribución principal consiste en redefinir la tarea de predicción como la identificación del *backbone* de una instancia, entendido como el conjunto de variables cuya asignación permanece fija en todas las soluciones óptimas. Predecir correctamente estas variables permite reducir el espacio de búsqueda sin comprometer la optimalidad. Para ello, se diseña una arquitectura de Redes Neuronales de Grafos (GNN) basada en una representación bipartita a nivel de literales, formulando la predicción como un problema de clasificación multiclase que identifica pertenencia y polaridad del backbone. Finalmente, las predicciones se incorporan al solucionador mediante una región de confianza adaptativa y parametrizada que ajusta dinámicamente el espacio de búsqueda.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Combinatorial optimization problems arise in numerous areas of science and engineering, such as planning, logistics, network design, bioinformatics, and hardware verification. These problems are characterized by the search for optimal solutions within large discrete spaces. In many cases, they belong to the class of NP-hard problems, meaning that no algorithms are known that can solve all instances in polynomial time. As a consequence, the size of instances that can be solved exactly in practice is limited by computational constraints.

Within the field of Operations Research, exact and heuristic techniques have been developed over the past decades to address increasingly large and complex instances. However, the growth in scale and difficulty of real-world problems continues to pose new challenges. In this context, the recent rise of machine learning techniques has motivated their integration into combinatorial optimization algorithms [9], with the goal of learning structural patterns from previously solved instances and improving the performance of traditional methods. Such improvements may be reflected in reduced solving times, the ability to handle larger instances, or, in the case of incomplete methods, better solution quality.

Among this broad spectrum of problems, Pseudo-Boolean Optimization (PBO) [15] provides a general framework for modeling problems defined over binary variables. In these formulations, constraints are expressed as linear inequalities over weighted combinations of Boolean variables, together with an objective function to be optimized. This framework allows modeling a wide variety of fundamental problems, such as *Vertex Cover*, *Independent Set*, and *Set Cover* [5, 20]. Despite significant advances in algorithms and solver technologies [25, 23, 2, 30, 48], many practical PBO instances remain challenging to solve efficiently.

In parallel, several learning-guided approaches have begun to be systematically integrated into algorithms for discrete optimization problems [8, 39, 61, 43]. These methods aim to exploit information learned from previously solved instances to guide the search process in new ones. Strategies explored in the literature include predicting relevant structural features [54], estimating variable values [37, 50], and defining restricted search regions based on learned information [31].

The application of machine learning models is particularly attractive in scenarios where instances share a common underlying structure. In many industrial contexts, the same type of problem must be solved repeatedly, with only certain parameters or dimensions varying, while the overall structure of the model remains similar. This regularity creates an opportunity to learn meaningful patterns that can be reused to substantially improve efficiency when solving new instances [17, 44].

In the analysis of discrete optimization problems, certain properties capture global characteristics of the set of optimal solutions. One such property is the *backbone*, defined as the set of variables whose assignment remains fixed across all optimal solutions of a given instance. The backbone provides information about the invariant components of the optimal solution space, identifying decisions that cannot vary without sacrificing optimality. From an algorithmic perspective, this information can be used to restrict the search space and guide decision-making strategies within a solver.

In Boolean satisfiability (SAT), the backbone has been used as a training signal in learning-guided approaches [62, 41], demonstrating that exploiting global structural information can improve the search process. Furthermore, in the context of Mixed-Integer Linear Programming (MILP), frameworks combining prediction and restricted search have been proposed [31, 26, 37], in which a model learns to estimate variable values and the solver explores bounded regions around those predictions. Together, these advances show that learned information can effectively complement traditional optimization methods, provided that it captures relevant structural aspects of the problem.

However, important limitations and open challenges remain. In particular, although the backbone has been exploited in SAT [62], its systematic use in the context of Pseudo-Boolean Optimization has not been developed to the same extent. Moreover, learning-guided methods face challenges related to generalization under distribution shifts and to the preservation of formal optimality guarantees [35, 7]. Predictive models are subject to various sources of error—stemming from training data, model limitations, or inherent uncertainty—making it impossible to ensure absolute correctness of their predictions. Consequently, integrating learned information into exact optimization frameworks requires mechanisms that balance empirical performance gains with theoretical robustness.

In this context, the research problem addressed in this thesis is to study the use of the *backbone* as a training signal in Pseudo-Boolean Optimization problems and to integrate it into a learning-guided search framework. Specifically, we investigate how to generate backbone information to construct suitable training datasets, how to design a model capable of capturing the structure of PBO formulations, and how to incorporate its predictions in a controlled manner within the solving process. The ultimate objective is to determine whether the backbone can serve as an effective structural signal for guiding search and under what conditions its integration can improve performance without compromising the fundamental properties of the

optimization process.

Hypothesis

Based on the identified research gap and the proposed methodological framework, we formulate the following research hypothesis:

Hypothesis:

Predicting backbones in Pseudo-Boolean Optimization (PBO) problems using a Graph Neural Network (GNN), when integrated within a Predict-and-Search (PaS) framework, will improve the primal integral metric compared to previous PaS-based approaches.

This hypothesis is empirically testable through the implementation of the proposed approach and its comparative evaluation against existing methods.

Objectives

The general objective of this work is to develop a novel approach for Pseudo-Boolean Optimization (PBO) that leverages Graph Neural Networks (GNNs) to predict PBO backbones within a Predict-and-Search (PaS) framework, with the aim of improving the primal integral metric.

The specific objectives of this thesis project are:

1. To design and implement a backbone extraction tool for PBO instances, enabling the generation of high-quality labels for training and evaluation.

2. To develop and train a Graph Neural Network (GNN) capable of accurately predicting backbone variables in PBO instances.

3. To integrate the GNN-generated backbone predictions into a Predict-and-Search (PaS) framework, allowing the solver to effectively guide its search using these neural recommendations.

4. To rigorously evaluate the performance of the proposed BackPaS framework by measuring improvements in primal integral metric, and to compare it against state-of-the-art commercial solvers and learning-based approaches.

Document Organization

This document is structured as follows. Chapter 1 introduces the research problem, presents the hypothesis, states the objectives, and outlines the structure of the

thesis. Chapter 2 provides the necessary theoretical background, including Pseudo-Boolean Optimization, the Predict-and-Search framework, and the concept of backbone. Chapter 3 reviews related work in the literature. Chapter 4 describes the proposed model and its main components. Chapter 5 details the experimental design and presents the obtained results. Finally, Chapter 6 concludes the thesis and discusses directions for future work.

Chapter 2

Background

2.1 Constraint Programming

Constraint Programming (CP) is a powerful paradigm for solving hard combinatorial problems by focusing on the relations between variables rather than the specific steps required to compute a result. Unlike imperative programming, CP is *declarative*, this means that the user describes the properties of a valid solution through a set of constraints, and a specialized program, known as a *solver*, searches for an assignment that satisfies them [51].

CP is particularly effective for **combinatorial problems**, which involve finding a specific configuration, grouping, or arrangement of discrete objects from a finite but extremely large set of possibilities. These problems emerge in many modern industrial and academic fields, some examples are: Assigning gate numbers to aircraft at an airport [16], allocating courses and teachers to classrooms and time slots [18], optimizing production flows in industrial job-shop scheduling[63] and determining the most efficient paths for a fleet of vehicles to deliver goods[58]

Most of these problems belong to the *NP* complexity class [6], meaning no efficient (polynomial-time) solution is known often requiring exponential time to solve exactly.

In the CP paradigm, the problems are classified based on whether the objective is to find any valid solution or to identify the best possible one according to a quality metric.

- **Constraint Satisfaction Problem (CSP)** [51]: Corresponds to a *decision problem*, where the goal is to determine if a feasible assignment exists. Formally, a CSP is defined as a triple (X, D, C):
  - $X = \{x_1, x_2, \ldots, x_n\}$ is a set of **decision variables**.
  - $D = \{D_1, D_2, \ldots, D_n\}$ is a set of **domains**, where each D_i contains the permitted values for the variable x_i.
  - $C = \{c_1, c_2, \ldots, c_m\}$ is a set of **constraints**. Each c_j is a relation over a subset of X that restricts the values the variables can take simultaneously.

A solution to a CSP is an assignment of a value $v_i \in D_i$ to each variable x_i such that all constraints in C are satisfied.

- **Constraint Optimization Problem (COP)** [51]: Extends the CSP framework to handle *optimization problems*. A COP is formally defined as a quadruple (X, D, C, f), where (X, D, C) is a standard CSP and:
  - $f : D_1 \times \cdots \times D_n \to \mathbb{R}$ is an **objective function**.

  The goal of a COP is to find a feasible assignment x^* that minimizes (or maximizes) the value of $f(x^*)$. This framework allows for the comparison of different feasible solutions, guiding the solver toward the *optimal* solution according to a specific criterion.

2.1.1 Mathematical Modeling Frameworks

While CSP and COP provide the formal definitions needed to model many different types of problems. Sometimes it is preffered to use more restricted modeling frameworks.

The Boolean Satisfiability Problem (SAT)

The *Boolean Satisfiability Problem* (SAT) [19] is a canonical *decision problem* in computational logic and complexity theory. Its historical and technical importance lies in being the first problem classified as *NP-complete* [19]. This classification implies that any problem belonging to the *NP* class can be reduced to an *instance* of SAT in polynomial time. Therefore, efficiency in solving SAT is fundamental to addressing a vast number of problems in engineering and science.

Thanks to the generality of SAT, it is used as a mathematical modeling framework for other subproblems. In this context, SAT is a specific type of CSP with the following definitions:

- **Decision variables and domains**: The model uses a set of n **binary** variables x_i for $i \in \{1, \ldots, n\}$ that can take one of the truth values: `false` or `true`. These variables represent the logical atoms of a formula. In practice, they are manipulated through *literals* (l), where each literal can be the variable in its direct form (x_i) or negated ($\bar{x}_i$).
- **Constraints**: Constraints are expressed as *clauses*. A clause is a disjunction ($\vee$) of literals. The complete formula can be given in *Conjunctive Normal Form* (CNF), which is a conjunction ($\wedge$) of m clauses:

$$\Phi = \bigwedge_{j=1}^{m} \left(\bigvee_{k=1}^{r_j} l_{jk} \right) \tag{2.1}$$

A clause is considered satisfied if at least one of its literals is `true`, and the complete formula is satisfied only if all clauses are simultaneously satisfied.

Pseudo-Boolean modeling

While the SAT paradigm is limited to logical clauses, *Pseudo-Boolean* (PB) [15] modeling allows arithmetic operations on binary variables, facilitating the expression of more complex constraints and objective functions. A distinction is made between *Pseudo-Boolean Satisfaction* (PBS), which seeks any feasible assignment, and *Pseudo-Boolean Optimization* (PBO), which aims at an assignment that minimizes or maximizes a specific cost function. Comprehensive reviews on this topic can be found in [20, 14]. The modeling framework is therefore defined by:

- **Decision variables and domains**: They remain binary variables $x_i \in \{0, 1\}$ but are now treated as integers rather than purely logical values. Literals l are still being used, but now the logical negation can also be expressed as a mathematical expression $\bar{x}_i = (1 - x_i)$.
- **Objective Function**: PBO uses a pseudo-boolean function [15] to quantify the quality of solutions.
- **Constraints**: They are arithmetic inequalities or equalities between pseudo-boolean functions.

Depending on the type of pseudo-boolean functions admitted, and how they are represented, the models can be formulated in one of the following frameworks:

- **General Literal-Based Formulation**: This is the most expressive form, allowing nonlinear constraints and monomials (products of literals). The objective function is defined as $\min \sum_{S \subseteq \{1,\dots,n\}} c_S \prod_{l \in S} l$, where S is a subset of literals and $c_S \in \mathbb{R}$. The constraints follow a similar structure:

$$\sum_{S \subseteq \{1,\dots,n\}} a_{jS} \prod_{l \in S} l \ \bowtie_j \ b_j, \quad j = 1, \dots, m \tag{2.2}$$

  where $\bowtie_j \in \{\leq, \geq, =\}$. Although these high-degree monomials are highly expressive, they can be linearized into quadratic or linear forms in polynomial time [15].

- **Linear Literal-Based Formulation**: By restricting monomials to individual literals, linear PB constraints are obtained.

$$\min \sum_{i=1}^{n} c_i l_i \quad \text{s.t.} \quad \sum_{i=1}^{n} a_{ij} l_{ij} \ \bowtie_j \ b_j \tag{2.3}$$

  where a_{ij} and b_j represent the real coefficients and thresholds of the *instance*, respectively.

- **Linear Variable-Based Formulation**: By substituting each literal l_i with its equivalent variable form (x_i or $1 - x_i$) and simplifying the terms, the model can be written as:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} x_i \ \bowtie_j \ b_j, \quad j = 1, \ldots, m \\ & x_i \in \{0, 1\}, \quad i = 1, \ldots, n \end{aligned} \tag{2.4}$$

This thesis focuses specifically on the **linear pseudo-Boolean optimization** modeling framework. Since any nonlinear PBO formulation can be transformed into a linear PBO in polynomial time, from this point onward the term PBO will refer exclusively to linear PBO unless explicitly stated otherwise.

Mixed-Integer Linear Programming (MILP)

Mixed-Integer Linear Programming (MILP) [49] is an *optimization framework* that generalizes a linear PBO by allowing the simultaneous inclusion of continuous and discrete decision variables. This flexibility enables the modeling of complex industrial problems in which logical decisions, represented by integer or binary variables, must interact with physical quantities such as time, mass, or distance, modeled through real-valued variables.

Following the established modeling structure, an *instance* of MILP is defined through the following components:

- **Decision variables and domains**: The model consists of a set of n **continuous** variables and p **integer** variables:

$$x_i \in \mathbb{R}, \quad i = 1, \ldots, n, \qquad y_k \in \mathbb{Z}, \quad k = 1, \ldots, p \tag{2.5}$$

  If all variables in the model are integers ($n = 0$), the model is referred to as *Integer Linear Programming* (ILP). If, in addition, these variables are restricted to the binary domain $\{0, 1\}$, the model corresponds to a *Binary Integer Linear Programming* (BILP), which is formally equivalent to a *linear PBO*.

- **Objective Function**: The model seeks to optimize a linear relationship among the decision variables. The general form is expressed as:

$$\min \quad \sum_{i=1}^{n} c_i x_i + \sum_{k=1}^{p} d_k y_k \tag{2.6}$$

  where c_i and d_k are the costs or contributions associated with the continuous and integer variables, respectively.

- **Constraints**: They are defined as a set of m linear inequalities or equalities that relate both types of variables:

$$\sum_{i=1}^{n} a_{ji} x_i + \sum_{k=1}^{p} b_{jk} y_k \ \bowtie_j \ e_j, \quad j = 1, \ldots, m \tag{2.7}$$

  where $e_j \in \mathbb{R}$ represents the bound value and the operator belongs to the set $\bowtie_j \in \{\leq, \geq, =\}$.

.

2.1.2 Modern Solving Paradigm

Since many of the problems modeled as SAT, PB or MILP are NP-hard, they are solved using specialized *solvers*, high-performance software systems designed to efficiently explore enormous search spaces. In practice, these *solvers* are used as "black boxes" from the user's perspective, decoupling the high-level mathematical modeling of an instance from the low-level algorithmic execution.

A *solver* takes as input an instance of a problem formally modeled under a given framework, such as SAT, PBO, or MILP, and begins its resolution process. Due to the high level of specialization of these programs, a solver typically accepts as input only one or a limited set of modeling frameworks.

Exact vs. Incomplete Search Solvers

Due to the high complexity of some optimization problems, obtaining optimal solutions may be very costly or practically infeasible. In such cases, solvers capable of providing very good, although not necessarily optimal, solutions are useful when the problem does not require a guarantee of optimality.

Solvers are classified according to their ability to provide optimality guarantees into the following classes: **Exact algorithms** are complete, which means that they guarantee finding the global optimum eventually and provide a mathematical proof of such optimality, or proof that no feasible solution exists. Conversely, **incomplete algorithms** prioritize finding sufficiently good solutions quickly in massive search spaces where a proof of optimality would be computationally prohibitive.

During the search for an optimal solution, or a sufficiently good one in the case of incomplete algorithms, solvers generally provide feasible solutions as they are found. The best feasible solution identified at a specific point during execution is referred to as the **incumbent**.

It is desirable for a solver to find high-quality incumbents rapidly. The way in which the quality of the incumbent evolves during the execution of the solver is referred to as the anytime behavior. This behavior is measured using the Primal Integral metric [3].

The Primal Integral is a standard metric for measuring the anytime behavior of a solver. It quantifies how quickly the incumbent $\mathbf{v_t}$ at time t approaches the best-known solution $\mathbf{v_{opt}}$ of the instance. It is defined by integrating the Primal Gap function [3] $p(t)$ (Equation 2.8), which measures the relative distance between the incumbent at time t and the best-known solution.

$$p(t) = \begin{cases} 1 & \text{if no incumbent exists at } t \\ 1 & \text{if } f(\mathbf{v_t}) \cdot f(\mathbf{v}_{opt}) < 0 \\ 0 & \text{if } f(\mathbf{v_t}) = f(\mathbf{v}_{opt}) = 0 \\ \frac{|f(\mathbf{v_t}) - f(\mathbf{v}_{opt})|}{\max\{|f(\mathbf{v_t})|, |f(\mathbf{v}_{opt})|\}} & \text{otherwise} \end{cases} \tag{2.8}$$

where $f(\mathbf{v})$ denotes the objective function value. The Primal Integral (PI) over a total time limit T is calculated as $PI(T) = \int_0^T p(t)\, dt$. Geometrically, this represents the area under the curve of the primal gap function over time. A lower Primal Integral indicates that a solver finds high-quality solutions more rapidly, making it a robust metric for comparing the efficiency of different search strategies or heuristic improvements.

Algorithmic Engines: CDCL and Branch-and-Cut

Although this thesis employs *solvers* as high-level tools, it is important to distinguish two key algorithms for solving these problems, since they have specialized in handling models with different levels of abstraction.

Conflict-Driven Clause Learning (CDCL) [45] is the dominant paradigm for SAT and many PBO *solvers*. It operates by performing a search with non-chronological *backtracking*. When the *solver* encounters a conflict, that is, an inconsistent assignment of variables, it analyzes the failure to learn a new constraint, or clause, that prevents the same mistake from recurring. This mechanism enables the *solver* to skip multiple levels in the search tree to escape dead ends. Relevant *solvers* that implement this paradigm include Kissat [12] and CadiCal [11] for SAT, and RoundingSAT [25] for PBO.

Branch-and-Cut [46] is the primary algorithm implemented by MILP *solvers*. It extends the basic *Branch-and-Bound* method, which partitions the problem into smaller subproblems through branching on variables, by adding cutting planes. These are additional linear inequalities generated during the search that remove fractional parts of the feasible region, tightening the search space without eliminating any valid integer solution. Relevant *solvers* that implement this paradigm include SCIP [2], an *open source* MILP solver, and Gurobi [30], a *closed source* MILP solver.

Note that both algorithms can be applied to PBO instances.

2.1.3 Backbones

The concept of the "backbone" was introduced to study the intrinsic hardness of combinatorial optimization and satisfiability problems [47]. It identifies the structural "core" of a problem by isolating variables that are "fixed" across all optimal solutions. Intuitively, backbone variables represent the most constrained part of the search space, as they allow for no flexibility if optimality (or satisfiability) is to be maintained.

Formally, given a PBO instance, let $\mathcal{F} \subseteq \{0,1\}^n$ be the feasible set and f^* be the optimal objective value of the instance. We define the set of optimal solutions as $\mathcal{O} = \{x \in \mathcal{F} \mid f(x) = f^*\}$.

- **Definition of the Backbone**: A variable x_i belongs to the backbone if it takes the same value $v \in \{0,1\}$ in every optimal solution:

$$\mathcal{B} = \{(i, v) \mid x_i = v \quad \forall x \in \mathcal{O}\}. \tag{2.9}$$

- **Computational Complexity**: Determining the backbone set $\mathcal{B}$ is co-NP-complete [38]. This complexity arises because verifying that a variable belongs to the backbone requires proving that no optimal solution exists where that variable takes a different value, essentially requiring exhaustive reasoning over the entire set of optimal assignments.

- **Impact on Search**: The relationship between backbone size and problem hardness is complex and domain-dependent [57]. In problems like 3-SAT and Graph Coloring, computational hardness often increases with the size of the backbone. In these cases, the backbone acts as a "hidden" structure that solvers must uncover. Conversely, in problems like the Traveling Salesman Problem (TSP) or Blocks World planning, hardness is weakly or even negatively correlated with backbone size.

2.1.4 Graph Representations

Combinatorial problems can be represented using **bipartite graphs**. This representation is constructed by defining two disjoint sets of nodes: one corresponding to variables (or literals) and another corresponding to constraints or clauses. An edge between the two sets indicates that a variable participates in a specific constraint.

In optimization models, the objective function is often incorporated as an additional specialized constraint node.

Variable-Based Bipartite Graph

A *variable-based bipartite graph* is used to represent the relationship between decision variables and linear constraints.

- **Nodes**: The graph consists of two disjoint sets of nodes. The first set contains **Variable Nodes**, representing decision variables (binary variables in PB, or a mixture of continuous and integer variables in MILP). The second set contains **Constraint Nodes**, representing each linear inequality or equality in the model.
- **Edges**: An edge connects a variable node to a constraint node if the variable appears in that constraint. Each edge is weighted by the corresponding coefficient of the variable in the constraint.

For example, consider the PBS instance defined in Equation 2.10:

$$\begin{aligned} a_{1,1}x_1 + a_{2,1}x_2 &\leq b_1 \\ a_{2,2}x_2 + a_{3,2}x_3 &\leq b_2 \\ a_{3,3}x_3 &\leq b_3 \\ a_{1,4}x_1 &\leq b_4 \end{aligned} \tag{2.10}$$

This instance is represented by the graph shown in Figure 2.1, where circular nodes correspond to the variables x_1, x_2, x_3, and rectangular nodes correspond to the constraints $C_1, \ldots, C_4$. The right-hand side values $(b_1, \ldots, b_4)$ and the type of relation $(\leq, \geq, =)$ are typically stored as attributes of the constraint nodes. Alternatively, constraints can be normalized so that they all share the same inequality type and right-hand side convention.

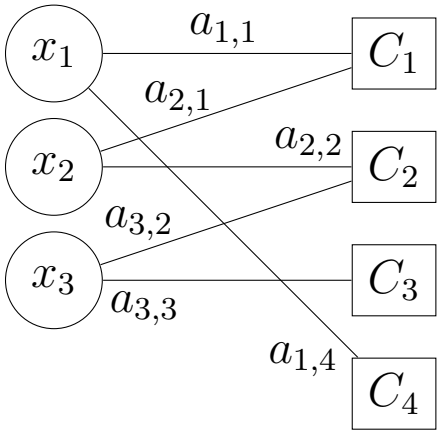


Figure 2.1: Variable-based Bipartite graph representation of a PBS instance.

Literal-Based Bipartite Graph

A **literal-based bipartite graph** distinguish explicitly between a variable and its negation:

- **Nodes**: The variable set is expanded into **Literal Nodes**. Each variable x_i is represented by two distinct nodes: the positive literal l_i (or x_i) and the negative literal $\bar{l}_i$ (or $\bar{x}_i$). The second partition consists of **Constraint (or Clause) Nodes**.
- **Edges**: An edge connects a literal node to a constraint node if and only if that specific literal appears in the constraint. As in the variable-based representation, edges may be weighted in PB settings to reflect the corresponding coefficients.

For example, consider the PBS instance defined in Equation 2.11:

$$\begin{aligned} a_{1,1}l_1 + a_{2,1}l_2 &\leq b_1 \\ a_{2,2}\bar{l}_2 + a_{3,2}l_3 &\leq b_2 \\ a_{3,3}\bar{l}_3 &\leq b_3 \\ a_{1,4}\bar{l}_1 &\leq b_4 \end{aligned} \tag{2.11}$$

This instance is represented by the graph shown in Figure 2.2. Here, circular nodes correspond to literals and rectangular nodes correspond to constraints $C_1, \ldots, C_4$. The right-hand side values $(b_1, \ldots, b_4)$ and the type of relation $(\leq, \geq, =)$ are handled in the same way as in the variable-based graph, typically as attributes of the constraint nodes.

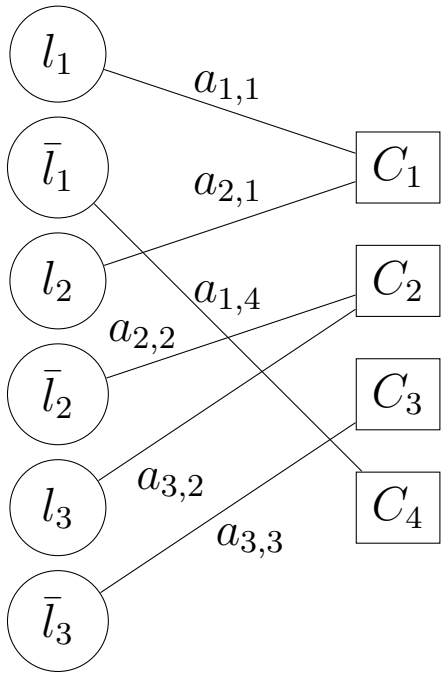


Figure 2.2: Literal-based Bipartite graph representation of a PBS instance. The circular nodes represent literals, and the rectangular nodes represent constraints.

2.1.5 Benchmark Problems

Classic Graph Problems: MIS and MVC

The *Minimum Vertex Cover* (MVC) and *Maximum Independent Set* (MIS) problems are fundamental combinatorial optimization problems defined on graphs. Due to their simple mathematical structure and well-understood properties, they are widely used as *benchmarks* to evaluate the efficiency of optimization *solvers*.

- **Definitions and Duality**:

  Given a graph $G = (V, E)$, a *vertex cover* is a subset $C \subseteq V$ such that every edge $\{u, v\} \in E$ has at least one endpoint in C. In contrast, an *independent set* is a subset $I \subseteq V$ in which no two vertices are adjacent. These two problems are structurally dual: a set C is a vertex cover if and only if its complement $V \setminus C$ is an independent set. Consequently, $|C| + |I| = |V|$, implying that minimizing the size of a vertex cover is mathematically equivalent to maximizing the size of an independent set.

- **Mathematical Formulations**:

  Using binary decision variables $x_v \in \{0, 1\}$, where $x_v = 1$ indicates that vertex v is selected, both problems can be formulated within the PBO or MILP frameworks.

  For the *weighted MVC* with vertex weights $w(v)$, the objective and constraints are formulated in Equation 2.12.

$$\begin{aligned} \min \quad & \sum_{v \in V} w(v) x_v \\ \text{s.t.} \quad & x_u + x_v \geq 1 \quad \forall \{u, v\} \in E. \end{aligned} \tag{2.12}$$

  For the unweighted MIS, the objective and constraints are formulated in Equation 2.13.

$$\begin{aligned} \max \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad & x_u + x_v \leq 1 \quad \forall \{u, v\} \in E. \end{aligned} \tag{2.13}$$

- **Instance Generation**:

  To evaluate solver performance on realistic topologies, this thesis employs **Barabási–Albert (BA)** graphs [4]. These graphs are generated through a random and iterative process based on a *preferential attachment* mechanism, where new nodes connect to existing ones with probability proportional to their degree. This process produces *scale-free networks* characterized by a power-law degree distribution, in which a small number of highly connected hubs coexist with many low-degree nodes. Such structures resemble real-world social and biological networks, making BA graphs a common choice for synthetic benchmarking.

Economic Mechanisms: Combinatorial Auctions

Combinatorial Auctions (CA) model complex market environments in which bidders assign values to *bundles* of items rather than to individual goods. This framework captures important economic phenomena such as *complementarity* (items that are more valuable together, e.g., a pair of earrings) and *substitutes* (items that are less valuable when acquired jointly, e.g., two brands of the same product) [21, 42].

- **The Winner Determination Problem (WDP)**:

  The central computational challenge in combinatorial auctions is the *Winner Determination Problem* (WDP), which seeks to select a subset of non-conflicting bids that maximizes the auctioneer's total revenue.

- **Mathematical Formulation**:

  Consider m items and n bids, where each bid i offers a price p_i for a bundle B_i. Let $x_i \in \{0, 1\}$ be a binary decision variable indicating whether bid i is accepted. The WDP can be formulated as a PBO problem:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} p_i x_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} x_i \leq 1, \quad \forall j \in \{1, \ldots, m\}, \end{aligned} \tag{2.14}$$

  where $a_{ij} = 1$ if item j is included in bundle B_i, and $a_{ij} = 0$ otherwise. This formulation ensures that each item is allocated at most once.

- **Instance Generation**:

  This thesis employs the *Arbitrary* distribution [42] to generate benchmark instances. Under this distribution, items are added to bundles according to similarity measures and the intrinsic preferences of bidders. As a result, the generated bundles exhibit realistic synergies and dependencies, making them more representative of real-world bidding behavior than purely random constructions.

2.2 Deep Learning and Graph Neural Networks

2.2.1 Statistical Learning

Machine Learning

Machine Learning (ML) is a subfield of artificial intelligence focused on the development of algorithms that improve their performance on a task through experience,

where experience is provided in the form of data [53]. Formally, a learning algorithm receives samples drawn from an underlying data distribution and aims to learn a function that generalizes well to previously unseen instances by identifying statistical patterns in the observed data.

Depending on the structure of the available data and the learning objective, machine learning methods are typically categorized into different paradigms:

- **Supervised Learning**: The learning experience consists of structured input–output pairs $\{(x_i, y_i)\}_{i=1}^{N}$. The goal is to learn a mapping f such that $f(x)$ approximates the target output y. In this thesis, this paradigm is used to predict the backbone of a PBO instance.
- **Unsupervised Learning**: The learning experience consists only of input data $\{x_i\}_{i=1}^{N}$ without specified targets. The objective is to uncover latent structural patterns in the data, such as clusters or low-dimensional representations, without explicit supervision.

Within supervised learning, the nature of the target space $\mathcal{Y}$ defines the type of learning task:

- **Classification**: The goal is to learn a mapping $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{Y}$ is a finite set of discrete labels.
- **Regression**: The goal is to learn a mapping $f : \mathcal{X} \to \mathbb{R}^d$, where the output space is continuous.

Deep Learning

This thesis focuses on a branch of ML denominated *Deep Learning* (DL). While classical *Machine Learning* methods often rely on *feature engineering*, where domain experts manually design relevant descriptors of the data, DL employs artificial neural networks (NNs) to learn these descriptors. Formally, DL employs NNs to model a parametric function

$$f_\theta : \mathcal{X} \to \mathcal{Y} \tag{2.15}$$

where $\theta \in \mathbb{R}^P$ denotes the set of trainable *weights and biases*.

The defining characteristic of deep learning is the shift from manual feature design to automated representation learning. Rather than relying on human-engineered heuristics, deep neural networks map raw input (like text, images, or audio) into *embeddings*: dense, continuous-valued vectors in a lower-dimensional latent space that capture the underlying semantics of the data.

By composing affine transformations with nonlinear activation functions across multiple hidden layers, the model progressively refines these embeddings into increasingly abstract representations. Through this hierarchical succession of transformations, the network learns to cluster similar inputs together in the embedding space, extracting task-relevant features directly from the data.

This architecture enables the automatic extraction of complex patterns from raw inputs, leading to state-of-the-art performance in areas such as computer vision [40, 32], natural language processing [59, 22], and multimodal learning [1].

2.2.2 Neural Networks

The Multi-Layer Perceptron (MLP)

The *Multi-Layer Perceptron* (MLP) is the simplest and most foundational Neural Network (NN) in deep learning. It consists of an input layer, one or more hidden layers, and an output layer. The architecture of a neural network specifies how these layers are organized and how information propagates through the model. Formally, an architecture determines the class of functions that can be represented through the composition of parameterized layers.

Let $L \in \mathbb{N}$ denote the number of layers. The network function can be expressed as a composition:

$$f_\theta(x) = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}(x) \tag{2.16}$$

where each $f^{(\ell)}$ represents a parameterized transformation. The expressive power of this hierarchical structure is formalized by the **Universal Approximation Theorem** [33], which states that a feedforward network with at least one hidden layer can approximate any continuous function on compact subsets of $\mathbb{R}^n$, provided it has sufficiently many neurons and nonlinear activation functions.

The Feedforward Layer The fundamental building block of the MLP is the fully connected (affine) layer, also known as a *feedforward* layer. Let $x \in \mathbb{R}^{d_{\text{in}}}$ be the input vector, where $d_{\text{in}} \in \mathbb{N}$ denotes the input dimension, and let $d_{\text{out}} \in \mathbb{N}$ denote the output dimension. The affine transformation is defined as:

$$z = Wx + b \tag{2.17}$$

where $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and $b \in \mathbb{R}^{d_{\text{out}}}$ are trainable parameters representing the *weights* and *biases*, respectively.

In practice, a nonlinear activation function σ is applied to enable the model to learn complex, nonlinear mappings:

$$h = \sigma(z) = \sigma(Wx + b). \tag{2.18}$$

Common choices for σ include the **sigmoid** function, which maps inputs to the range $(0, 1)$, and the **Rectified Linear Unit (ReLU)**, defined as $\text{ReLU}(z) = \max(0, z)$. ReLU is typically preferred in deep architectures due to its favorable optimization properties and improved gradient propagation.

Network Training Training a neural network consists of determining the parameters θ that minimize a predefined loss function $\mathcal{L}$, which can be interpreted as defining an **error surface** over the parameter space. Given a supervised dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where $N \in \mathbb{N}$ denotes the number of training samples, the empirical risk minimization objective is:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\big(f_\theta(x_i), y_i\big). \tag{2.19}$$

Optimization is typically performed using first-order gradient-based methods via the **backpropagation** algorithm [52]. Methods such as Stochastic Gradient Descent (SGD) or adaptive variants like Adam iteratively update the parameters according to:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}, \tag{2.20}$$

where $\eta > 0$ denotes the learning rate. This iterative procedure drives the parameters toward a local minimum of the loss function by following the negative gradient direction.

Loss Functions and Learning Strategies

In classification tasks with a label set $\mathcal{Y} = \{1, \ldots, C\}$, the goal is to map input data to a probability distribution over C discrete classes. This is achieved by coupling the network's architectural output with an appropriate probability transformation and a logarithmic loss function.

Categorical Cross-Entropy (CCE) The final layer of a neural network typically produces a vector of raw scores called *logits*, $z \in \mathbb{R}^C$. To interpret these as probabilities, the **softmax** function is applied:

$$\text{softmax}(z)_c = \hat{p}_c = \frac{e^{z_c}}{\sum_{k=1}^{C} e^{z_k}}, \tag{2.21}$$

which ensures that each component $\hat{p}_c \in [0, 1]$ and the entire vector sums to one.

In the context of multi-class classification the **Categorical Cross-Entropy (CCE) loss is utilized.** For a single example with a ground-truth label y, the cross-entropy loss measures the performance of the model by taking the negative log-likelihood of the correct class:

$$\mathcal{L}_{\text{CE}} = -\log \hat{p}_y. \tag{2.22}$$

Over the full dataset of N samples, the empirical loss is minimized to drive the predicted probability of the correct labels toward unity:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \log \hat{p}_{i,y_i}. \tag{2.23}$$

Similarity-Based Objectives: Contrastive Learning While cross-entropy relies on explicit labels, *contrastive learning* is a framework designed to learn meaningful representations by comparing samples directly within a latent space.

The objective is to learn an embedding function $g_\theta(x) \in \mathbb{R}^d$ such that "similar" samples (positive pairs) are pulled together, while "dissimilar" samples (negative pairs) are pushed apart in the manifold.

A standard formulation for this task is the **InfoNCE** (Information Noise-Contrastive Estimation) loss. For a positive pair (i, j) and a batch of size B, the loss is defined as:

$$\mathcal{L}_{\text{NCE}} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{B} \exp(\text{sim}(z_i, z_k)/\tau)}, \tag{2.24}$$

where $z = g_\theta(x)$, $\text{sim}(\cdot, \cdot)$ is a similarity metric (often cosine similarity), and τ is a temperature hyperparameter. By maximizing the agreement between different views of the same data point relative to other points in the batch, contrastive learning allows the model to capture the intrinsic structure of the data.

2.2.3 Graph Neural networks

Message Passing Neural Networks (MPNN)

While classical neural architectures such as MLPs are effective for data defined on regular structures (e.g., tabular data), they do not naturally generalize to more complex structures such as graphs. Graphs exhibit arbitrary topology: the number of neighbors varies across nodes, and there is no inherent ordering of the input. Consequently, a *Graph Neural Network* (GNN) must be permutation invariant, meaning that its output should remain unchanged regardless of the order in which a node's neighbors are processed. This requirement makes standard fixed-length weight matrices insufficient without specialized aggregation mechanisms.

The MPNN Framework The *Message Passing Neural Network* (MPNN) framework [29] provides a unifying abstraction for most modern GNN architectures. A message-passing layer constitutes the fundamental building block of GNNs, which operate on graph-structured data by propagating information along edges.

Let $G = (V, E)$ be a graph with $|V| = n$ nodes and edge set $E \subseteq V \times V$. Each node $v \in V$ is associated with a feature vector $h_v^{(\ell)} \in \mathbb{R}^d$ at layer ℓ, where $d \in \mathbb{N}$ denotes the feature dimension.

A message-passing layer updates node representations through three fundamental operations: *Message Generation*, *Aggregation*, and *Update*:

1. **Message and Aggregation:**

$$m_v^{(\ell)} = \text{AGG}\left(\{\, \phi(h_v^{(\ell)}, h_u^{(\ell)}) \mid u \in \mathcal{N}(v) \,\}\right), \tag{2.25}$$

where $\mathcal{N}(v)$ denotes the set of neighbors of node v, ϕ is a trainable message function (for example, an MLP), and AGG is a permutation-invariant aggregation operator such as sum, mean, or max.

2. **Node Update:**
$$h_v^{(\ell+1)} = \psi\big(h_v^{(\ell)}, m_v^{(\ell)}\big), \tag{2.26}$$
where ψ is a trainable update function, often implemented as an MLP.

By stacking multiple message-passing layers, each node representation progressively incorporates information from larger neighborhoods in the graph, enabling learning over relational and structured data.

Note on Terminology The terms "Graph Convolution" and "Graph Convolutional Networks" (GCN) frequently appears in the GNN literature. In the context of this thesis, and following modern usage, it refers to the spatial process of local neighborhood filtering and aggregation described above. This differs from earlier spectral methods, which defined convolutions via the Graph Laplacian and operations in the frequency domain. Although mathematically distinct, modern spatial graph convolutions are generally preferred due to their computational efficiency and their ability to generalize to previously unseen graph structures.

Graph Tasks

Depending on the specific application, Graph Neural Networks (GNNs) can be designed to produce outputs at different levels of granularity within the graph structure. These tasks are generally categorized into three hierarchical levels:

- **Graph-Level Tasks**: The objective is to predict a single property for the entire graph G. This is typically achieved by applying a *global pooling* or *readout* operation that aggregates all node embeddings into a single vector representation, which is then used for prediction.
- **Edge-Level Tasks**: These tasks focus on predicting properties of relationships between nodes or the existence of connections. *Link prediction* is a prominent example, where the network estimates the probability of an edge between two nodes. This approach is commonly used in applications such as recommendation systems and knowledge graph completion.
- **Node-Level Tasks**: The goal is to predict properties or labels for individual nodes $v \in V$ based on their learned embeddings $h_v^{(L)}$. This thesis adopts this paradigm to identify backbone variables, where variables are modeled as nodes in a graph representing a PBO instance.

2.2.4 Attention in GNNs

In the classical MPNN framework, the aggregation operation treats all neighboring nodes equally, meaning that each neighbor contributes uniformly to the update of a node representation. More advanced architectures incorporate attention mechanisms to perform weighted aggregation. This modification allows the model to dynamically learn the relative importance of different neighbors.

Graph Attention Networks (GAT)

The *Graph Attention Network* (GAT) [60] introduced learned attention weights for graph-structured data. Instead of fixed summation or averaging, GAT employs **additive attention** to compute an attention coefficient α_{ij} that quantifies the importance of node j's features for node i:

$$e_{ij} = \text{LeakyReLU}\left(\mathbf{a}^T[Wh_i \parallel Wh_j]\right), \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}(i)}\exp(e_{ik})}, \tag{2.27}$$

where $\mathbf{a}$ is a trainable weight vector and $\parallel$ denotes concatenation. LeakyReLu is a modification of the ReLu activation function that outputs negative values instead of zero when the input is negative.

By using these learned coefficients, the model can effectively filter noise and prioritize relevant information. This provides significant advantages over standard aggregation models such as GCNs, particularly in heterogeneous neighborhoods where nodes may play different structural roles or exert varying levels of influence on the central node.

Modern Graph Transformers

The state of the art in graph representation learning has evolved toward adopting the **scaled dot-product attention** mechanism, transferring the effectiveness of Transformers from natural language processing (NLP) to structured domains. Unlike classical attention-based convolutional networks such as GAT, Graph Transformers are capable of modeling global dependencies and capturing complex relationships between nodes.

Unified Message Passing Attention Following the paradigm proposed by [59] and extended to graphs by [56], the model projects node and edge features into query (Q), key (K), and value (V) spaces. A key innovation of [56] is the direct incorporation of edge attributes into the attention score computation, allowing the graph topology to dynamically influence information propagation.

For a source node j and a target node i connected by an edge with attributes e_{ij}, the attention coefficient is computed as:

$$\alpha_{ij} = \text{softmax}_j \left(\frac{(W_Q x_i) \cdot (W_K x_j + W_E e_{ij})}{\sqrt{d_k}} \right), \tag{2.28}$$

where W_Q, W_K, and W_E are learnable weight matrices. This formulation enables edge features to act as an additive bias in the key space, increasing the expressiveness of the model compared to traditional additive attention mechanisms.

Although not used in this thesis, [56] also proposed the inclusion of the edge features in the aggregation step, where the message is enriched by relational information before being weighted by the attention coefficients:

$$h_i^{(l+1)} = \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \left(W_V h_j^{(l)} + W_E e_{ij} \right) \right), \tag{2.29}$$

where $\mathcal{N}(i)$ denotes the set of neighbors of node i, and W_V, W_E are the learned projection matrices for the node values and edge features, respectively. This formulation ensures that the edge attributes e_{ij} do not only modulate the attention scores but also contribute directly to the updated node representation $h_i^{(l+1)}$.

Multi-Head Attention and Layer Dynamics To capture diverse aspects of the graph structure simultaneously, the model employs a **Multi-head Attention** mechanism. This process is executed independently across H attention heads, whose outputs are concatenated to form the final representation:

$$\text{MultiHead}(Q, K, V) = (\text{head}_1 \parallel \text{head}_2 \parallel \cdots \parallel \text{head}_H)\, W^O. \tag{2.30}$$

Finally, the architecture incorporates residual connections and layer normalization (*LayerNorm*) to stabilize the training of deep networks:

$$x_i^{(l+1)} = \text{LayerNorm} \left(x_i^{(l)} + \text{Attention}(x_i^{(l)}, x_j^{(l)}, e_{ij}) \right). \tag{2.31}$$

This design mitigates common issues in deep GNNs, such as *over-smoothing*, and facilitates the learning of expressive representations in large-scale graph structures.

Chapter 3

Literature Review

3.1 Machine Learning for Combinatorial Optimization

Machine Learning (ML) models have been integrated into combinatorial optimization through various methodologies, which can be broadly categorized according to the role they play in the solving process.

3.1.1 End-to-End Approaches

End-to-end models aim to map an optimization instance directly to a solution, without explicitly relying on a traditional solver during inference. A notable example is NeuroSAT [55], whose objective was to solve SAT instances.

The model is a GNN that takes as input a bipartite graph representation of the SAT instance (see Section 2.1.4). The GNN was trained to perform a graph classification task, i.e., to classify the entire bipartite graph as SAT or UNSAT. The authors demonstrated that, beyond binary classification, satisfying variable assignments could often be successfully recovered from intermediate node embeddings.

Despite these advances, end-to-end solvers present important limitations. In particular, they do not provide formal guarantees of optimality or feasibility, and the generated solutions may be suboptimal or even infeasible. This is especially problematic in applications where strict guarantees are required.

3.1.2 Online Integration

To mitigate the limitations of end-to-end ML solvers, researchers have explored integrating ML models within, or in close interaction with, classical solvers. Within this line of work, a common paradigm is *online integration*.

In this approach, the ML model is repeatedly invoked during the execution of the solver to guide critical internal decisions. A representative work following this

paradigm is [28], where a GNN is integrated into the execution of a Branch-and-Bound algorithm. Specifically, the GNN is trained to imitate a computationally expensive branching heuristic known as *Strong Branching*.

Strong Branching can significantly reduce the size of the search tree, leading to improved decision quality. However, its direct use is computationally prohibitive. In this context, employing a GNN as an approximation is attractive, since it can evaluate branching candidates substantially faster.

The GNN receives as input a variable-based bipartite graph (See Section 2.1.4) representation of the MILP instance. The nodes of the graph and the edges are represented using vectors of features comming from the solution state of the algorithm.

Then it performs a variation of message passing for bipartite graphs in two steps named half-convolutions, which update constraint and variable embeddings by aggregating information from their respective neighbors:

The graph convolution is performed as two successive interleaved half-convolutions. In this architecture, the update functions f_C and f_V are realized by concatenating the aggregated neighborhood messages with the nodes' previous features, followed by a multi-layer perceptron (MLP):

1. **Variable-to-Constraint Pass:** Updates constraint features $\mathbf{c}_i$ by aggregating messages from variables $j \in \mathcal{N}(i)$:

$$\mathbf{c}_i \leftarrow f_C \left(\left[\sum_{j \in \mathcal{N}(i)} g_C(\mathbf{c}_i, \mathbf{v}_j, \mathbf{e}_{i,j}) \parallel \mathbf{c}_i \right] \right) \tag{3.1}$$

2. **Constraint-to-Variable Pass:** Updates variable features $\mathbf{v}_j$ by aggregating messages from constraints $i \in \mathcal{N}(j)$:

$$\mathbf{v}_j \leftarrow f_V \left(\left[\sum_{i \in \mathcal{N}(j)} g_V(\mathbf{c}_i, \mathbf{v}_j, \mathbf{e}_{i,j}) \parallel \mathbf{v}_j \right] \right) \tag{3.2}$$

Where g_C, g_V are MLPs used to compute the message and f_C, f_V are MLPs update functions.

Nevertheless, although GNN inference is faster than executing Strong Branching itself, its computational cost remains high when compared to the highly optimized efficiency of modern solvers. As a result, online learning approaches may still incur significant computational overhead due to frequent model inference during runtime.

3.1.3 Offline Integration

Another approach to leveraging ML models alongside traditional solvers is *offline* integration. In this setting, the ML model performs inference only once, prior to the execution of the solver, thereby avoiding runtime computational overhead.

An important strategy that follows this paradigm is the **Predict and Search (PaS)** framework [31]. In this framework, ML predictions are used to generate auxiliary constraints that prune the search space. By restricting the feasible region, the framework enables traditional solvers to skip unproductive branches.

Since this framework is central to the present thesis, its methodology is described in detail in the following section.

3.2 The Predict-and-Search Framework

The Predict and Search (PaS) framework [31] integrates statistical predictions into the symbolic search of a solver by decomposing the process into two stages: prediction and restricted search. It was proposed to improve the any-time behavior of MILP solvers.

3.2.1 The Prediction Step

In the prediction step the goal is to retrieve relevant information from the instance that could be used to suggest values to a subset of the decision variables. Specifically the implementation given by [31], uses a GNN to predict the values of each binary variable of a given MILP instance.

The GNN receives as input a variable-based bipartite graph (See Section 2.1.4) representation of the MILP instance. The nodes of the graph are represented using vectors of features computed directly from the mathematical formulation of the instance, some variable nodes features are: avg coefficient of the variable, coeffiecient of the varible in the objective function, how many constraints have this variable. examples of the constraint features are: how many variables have, what is the RHS, what is the direction of the constraint $(\leq, \geq, =)$.

The GNN architecture uses the half-convolution layers described in Section 3.1.2. And was trained using sets of high-quality feasible solutions for each training instance. The solution quality was used to compute a weight $w_{i,j}$ for each solution, which was incorporated into a weighted Cross-Entropy loss. The final loss is defined as:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N}\sum_{j=1}^{N_i} w_{i,j} \sum_{k=1}^{V_i} [x_{i,j,k} \log(\hat{x}_{i,k}) + (1 - x_{i,j,k}) \log(1 - \hat{x}_{i,k})] \tag{3.3}$$

Where:

- N is the total number of instances.
- N_i is the number of training solutions for instance i.
- V_i is the number of binary variables for instance i.
- $x_{i,j,k} \in \{0, 1\}$ is the ground-truth value of variable k in solution j of instance i.

- $\hat{x}_{i,k} \in [0, 1]$ is the GNN's predicted probability for variable k of instance i.
- $w_{i,j}$ is the normalized weight associated with solution j of instance i, calculated as:

$$w_{i,j} = \frac{\exp(-f_i(x_{i,j})/\tau)}{\sum_{m=1}^{N_i} \exp(-f_i(x_{i,m})/\tau)} \tag{3.4}$$

Where f_i is the objective function of instance i and τ is a temperature hyperparameter that was manually chosen for each benchmark.

Once the GNN has been trained, it can be used to compute scores $p_i \in [0, 1]$, representing the predicted likelihood that $x_i = 1$ in an high-quality solution.

3.2.2 The Search Step and Trust Regions

The search step is responsible for integrating the predictions into the original instance before the execution of the solver. This is achieved by constructing additional constraints that jointly define a *trust region*. This trust region represents the set of solutions that lie within a given distance from the ML model's predictions.

The implementation proposed in [31] first selects, for a given instance, the variables to be used in the trust region construction. This selection is controlled by two hyperparameters, k_0 and k_1. The k_1 variables with the highest prediction scores produced by the GNN are assigned $v_i = 1$, while the k_0 variables with the lowest scores are assigned $v_i = 0$. The selected variable set S, together with their assigned values, defines a partial assignment for the instance.

A **tolerance** parameter $\Delta \in \mathbb{Z}^+$ then defines the maximum allowable distance between this partial assignment and the solution found by the solver. For each $i \in S$, an auxiliary binary indicator variable δ_i is introduced, where $\delta_i = 1$ indicates a deviation from v_i. The following constraints are appended to the optimization instance:

$$x_i \leq \delta_i \qquad \forall i \in \{j \in S \mid v_j = 0\} \tag{3.5}$$

$$1 - x_i \leq \delta_i \qquad \forall i \in \{j \in S \mid v_j = 1\} \tag{3.6}$$

$$\sum_{i \in S} \delta_i \leq \Delta \tag{3.7}$$

Constraint (3.7) ensures that the solver only explores solutions within the specified distance Δ from the predicted assignment over S.

Finally, note that the trust region is governed by three hyperparameters: k_0, k_1, and Δ. These parameters are typically tuned using a validation set to optimize empirical performance. This thesis proposes a refined parameterization that reduces the number of hyperparameters to two, linking the trust region more directly to the model's confidence scores.

3.2.3 Contrastive Predict and Search

While the foundational PaS implementation [31] focuses on learning from high-quality solutions, the ConPaS framework [34] extends this approach by addressing the absence of negative training signals in the original model. By incorporating **contrastive learning**, ConPaS trains the GNN using pairs of high-quality (positive) and low-quality or infeasible (negative) solutions. This strategy enables the model to learn a more robust decision boundary for the feasible region, leading to improved empirical performance compared to the original PaS implementation.

The authors propose two variants: ConPaS-Inf, which uses infeasible solutions as negative samples, and ConPaS-LQ, which uses low-quality feasible solutions as negative samples. In the context of this thesis, ConPaS-LQ (hereafter referred to simply as ConPaS) serves as the primary baseline for comparison.

Importantly, aside from the training data construction and the loss function employed, the GNN architecture, the trust region construction, and the graph representation of the instances remain identical to those used in the original PaS framework.

3.3 Backbones in Search and Optimization

3.3.1 Usage to Improve Solvers

The *backbone* of a combinatorial optimization problem is defined as the set of variables that take the same value in all optimal (or all satisfying) solutions (for more details see Section 2.1.3). Identifying such variables can be used to reduce the effective problem complexity by shrinking the feasible search space.

This idea was explored by Dubois et al. [24], who proposed a search heuristic for hard 3-SAT instances based on backbone estimation. Their approach heuristically approximates (without using ML) the backbone of subsets of clauses and exploits this information to guide branching decisions in the DPLL algorithm (a precursor to the CDCL algorithm). By prioritizing assignments consistent with the estimated backbone, the method reduces the size of the search tree and yields significant performance improvements on hard instances.

More recently, ML-driven approaches have been proposed to predict backbone variables. Notably, NeuroBack [62] employs GNNs to predict backbone variables for SAT instances. These predictions are injected into the solver's phase-saving mechanism prior to the start of the search, resulting in substantial empirical speedups.

3.3.2 Extraction Tools

To study backbones (and, in the context of this thesis, to learn from them) it is necessary to rely on backbone extractors, i.e., specialized tools designed to compute backbone variables efficiently. Although determining whether a variable belongs to

the backbone is co-NP-complete [6] in general [38], several practical algorithms have been developed to extract backbone variables efficiently in real-world instances.

In the SAT domain, *CadiBack* [13] performs backbone computation through a sequence of iterative solver calls, combined with modern preprocessing and incremental solving techniques to identify literals that are fixed across all satisfying assignments.

More recently, in the context of Pseudo-Boolean Optimization (PBO), Carramiñana et al. [27] introduced three backbone extraction algorithms tailored to linear PBO instances. These methods take inspiration from *CadiBack* as a conceptual baseline but are implemented using different underlying solvers. Among them, **GuroBack** leverages the commercial solver Gurobi to certify variable fixations across all optimal solutions, exploiting its optimization and proof capabilities to determine backbone membership.

The present thesis builds directly upon the extraction methodologies introduced in [27], in particular using **GuroBack** as the backbone extraction tool for collecting training labels.

Chapter 4

Backbone-based Predict and Search for PBO

This chapter introduces the proposed framework BackPaS (Backbone-based Predict and Search) for Pseudo-Boolean Optimization (PBO). At a high level, BackPaS is a modified version of the original Predict and Search (PaS) framework, specifically tailored to improve performance on PBO instances. To this end, the following modifications are proposed:

- Use of backbone variables as training signals, instead of high-quality or low-quality solutions.
- Adoption of a literal-based bipartite graph representation rather than a variable-based representation.
- Use of deeper GNN architectures based on Graph Transformers.
- A new trust region parameterization that relies more directly on the confidence of the ML model.

These modifications are detailed in the following subsections. Section 4.1 presents the prediction step within BackPaS, including how the training data is collected (via backbone extraction), the GNN architecture employed, and the training procedure. Section 4.2 describes the search step, detailing how the trust region is constructed from the GNN predictions and which hyperparameters are used.

A fundamental distinction between BackPaS and previous PaS implementations is that each training sample conveys information about the global structure of the instance—namely, the set of all optimal solutions—whereas prior PaS approaches provided information only about the quality of individual assignments. The proposed approach is based on the hypothesis that learning backbone variables enables BackPaS to capture global structural properties of the optimization landscape, thereby providing richer and more reliable guidance for the search process.

4.1 Prediction Step

4.1.1 Backbone Extraction

To extract the ground-truth backbone labels required for training, the GuroBack [27] algorithm is employed. GuroBack is a general-purpose backbone extractor for optimization problems with binary variables, adapted from the CadiBack SAT backbone extractor [13]. This tool was developed by the author of this thesis primarily to be used in this work, but it was also included as part of a bigger collaborative paper regarding analysis and extraction of backbones in PBO discussed in Section 3.3.2. The algorithm utilizes an iterative refinement strategy where, in each step, a subset of binary variables is classified as either belonging to the backbone or not. This classification is performed by invoking Gurobi to determine if the assignments of these variables can be altered while maintaining the optimal objective value.

Given an optimization instance, the backbone extraction process is defined by the following stages:

Initialization: GuroBack first utilizes Gurobi to solve the original instance and obtain an optimal solution $\mathbf{v}^*$. A candidate set $\Lambda = \{i \mid x_i \text{ is binary}\}$, encompassing all binary variable indices, is created. The verified backbone set B is initialized as $\emptyset$, and the initial batch size is set to $m = |\Lambda|$.

Iterative Step: The following sub-steps are performed iteratively until the candidate set Λ is empty:

1. The next m indices of Λ are selected to form the current batch C.
2. A modified instance is solved that constrains the objective function to remain at $f(\mathbf{v}^*)$ while enforcing the condition that at least one $x_i \in C$ must differ from its optimal assignment v_i^*.
3. If the modified instance is *infeasible*, all variables in C are confirmed as backbone variables: $B \leftarrow B \cup \{(x_i, v_i^*) \mid i \in C\}$, and $\Lambda \leftarrow \Lambda \setminus C$. The batch size is then increased: $m \leftarrow 2m$.
4. If the modified instance is *feasible*, any index $i \in C$ such that $v_i \neq v_i^*$ is removed from Λ, and the batch size is reset to $m = 1$.

Termination: The process concludes when the candidate set Λ is exhausted. For a PBO instance, the resulting set B corresponds to the full backbone.

4.1.2 Prediction of the Backbone

To predict the backbones of PBO instances, a new GNN architecture is proposed which accepts a literal-based bipartite graph representation of the instances as input

(See Section 2.1.4). In this representation, each variable node is replaced by two nodes representing its positive and negative literals. To ensure consistent encoding, instances are normalized to use strictly non-negative coefficients. Specifically, any weighted sum of variables $\sum_{i=1}^{n} a_i x_i$ is transformed as follows:

$$\sum_{i \in I^+} a_i x_i + \sum_{i \in I^-} (-a_i)\bar{x}_i + \sum_{i \in I^-} a_i \,, \tag{4.1}$$

where a_i denotes the coefficient associated with variable x_i, $I^+ = \{i \in \{1, \ldots, n\} \mid a_i \geq 0\}$, and $I^- = \{i \in \{1, \ldots, n\} \mid a_i < 0\}$. The resulting constant term $\sum_{i \in I^-} a_i$ is used to adjust the constraint's right-hand side or the offset of the objective function.

Edges are established between literal nodes and constraint nodes based on this normalized formulation. Node embeddings are initialized using literal-based metrics, representing a modified version of the embeddings proposed by [31, 34] where specific literal-based metrics replace variable-based ones. Table 4.1 details the metrics used for the initial representation of each node.

The GNN architecture replaces the standard half-convolution layers used in [31, 34] with Bipartite Transformer layers adapted for interleaved message passing. This architecture processes the bipartite graph structure through a full graph convolution cycle composed of two sequential transformer-based passes:

1. **Variable-to-Constraint Pass:** The constraint nodes $\mathcal{C}$ are updated by treating them as target nodes $\mathbf{x}_{tgt}$ and variable nodes $\mathcal{V}$ as source nodes $\mathbf{x}_{src}$. A dedicated transformer layer computes attention over the variables participating in each constraint:

$$\mathbf{c}_i^{(l+1)} = \text{HalfBipartiteTransformer}_{\mathcal{V} \to \mathcal{C}} \left(\mathbf{v}^{(l)}, \mathcal{E}, \mathbf{a}, \mathbf{c}^{(l)} \right) \tag{4.2}$$

2. **Constraint-to-Variable Pass:** The variable nodes $\mathcal{V}$ are subsequently updated using the newly computed constraint embeddings $\mathbf{c}^{(l+1)}$ as the source. This second pass allows variables to aggregate global contextual information from the constraints:

$$\mathbf{v}_j^{(l+1)} = \text{HalfBipartiteTransformer}_{\mathcal{C} \to \mathcal{V}} \left(\mathbf{c}^{(l+1)}, \mathcal{E}^\top, \mathbf{a}, \mathbf{v}^{(l)} \right) \tag{4.3}$$

Each Half Bipartite Transformer layer computes a representation by performing multi-head attention over neighbors $j \in \mathcal{N}(i)$. For each head h, the attention coefficient $\alpha_{ij}^{(h)}$ is derived from the query of the target node and the edge-augmented key of the source node:

$$\alpha_{ij}^{(h)} = \text{softmax}_j \left(\frac{(\mathbf{Q}^{(h)} \mathbf{x}_i) \cdot (\mathbf{K}^{(h)} \mathbf{x}_j + \mathbf{E}^{(h)} \mathbf{a}_{ij})}{\sqrt{d_h}} \right) \tag{4.4}$$

Name	Meaning
Literal Embeddings	
`obj`	Normalized coefficient of the literal in the objective function.
`l_coeff`	Mean coefficient of the literal over all constraints in which it occurs.
`Nl_coeff`	Number of constraints in which the literal appears.
`max_coeff`	Maximum coefficient value among all constraints for the literal.
`min_coeff`	Minimum coefficient value among all constraints for the literal.
`Nv_coeff`	Number of constraints in which the corresponding variable occurs.
Constraint Embeddings	
`c_coeff`	Average of all coefficients in the constraint.
`Nc_coeff`	Degree of the constraint node in the bipartite representation.
`rhs`	Right-hand-side value of the constraint.
`sense`	The sense of the constraint (0 if $=$, 1 if $\geq$, -1 if $\leq$).
Edge Embeddings	
`coeff`	Coefficient of the literal in the constraint.

Table 4.1: Literal, Constraint, and Edge Embeddings for the literal-based bipartite graph representation.

The final update for each pass incorporates a residual connection and layer normalization:

$$\mathbf{x}_i^{(l+1)} = \text{LayerNorm}\left(\mathbf{x}_i^{(l)} + \text{MLP}_{out}\left(\|_{h=1}^{H} \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(h)} (\mathbf{V}^{(h)} \mathbf{x}_j)\right)\right) \tag{4.5}$$

where $\|$ denotes the concatenation of the H attention heads. In addition to the transformer-based message passing, the proposed GNN significantly increases the model depth, utilizing a total of 8 full Bipartite Transformer Layers, which correspond to 16 interleaved Half Bipartite Transformer layers.

The output layer is designed for a 3-class classification task, predicting a probability distribution $\mathbf{p}_i$ for each variable x_i over the following classes: **B0** (backbone-

negative, variable must be 0), **B1** (backbone-positive, variable must be 1), and **NB** (non-backbone).

Standard Cross-Entropy (CE) loss can be biased in the context of combinatorial optimization, as different instances often contain varying numbers of variables. To prevent larger graphs from contributing disproportionately to the total loss, which could mask poor performance on smaller, yet equally significant instances, we utilize a **Mean Cross-Entropy (MCE)** loss. This approach ensures that each problem instance contributes equally to the gradient updates, regardless of its scale.

Let $\mathcal{G}$ be the set of problem instances (graphs), and for each graph $G \in \mathcal{G}$, let V_G be the set of binary variable indices. If p_{i,y_i} represents the predicted probability assigned to the ground-truth class $y_i \in \{\text{B0, B1, NB}\}$ for variable x_i, the MCE is defined as:

$$\text{MCE} = \frac{1}{|\mathcal{G}|} \sum_{G \in \mathcal{G}} \left(\frac{1}{|V_G|} \sum_{i \in V_G} -\log(p_{i,y_i}) \right) \tag{4.6}$$

As defined in Equation 4.6, the inner summation calculates the Negative Log-Likelihood (NLL) per node, which is normalized by the graph size $|V_G|$ before the global average is taken across the batch or dataset. This two-step averaging process acts as a form of instance-level weighting, preventing the model from over-optimizing for the structural characteristics of the largest graphs in the training set.

4.2 Search Step

A new method is proposed to construct the trust region by automatically determining variable selection (S), value assignment (v_i), and the permissible error threshold (Δ). These decisions are guided by the predicted probabilities of the model, allowing the trust region characteristics to adapt to each specific instance. This configuration is controlled by two parameters: $\theta \in [0, 1]$, defining the minimum confidence for variable selection, and $\alpha \in [-1, 1]$, which calibrates model confidence for unseen instance distributions.

Given the predicted probabilities $p_{i,\text{B0}}$, $p_{i,\text{B1}}$, and $p_{i,\text{NB}}$, the trust region is constructed as follows:

Variable Selection (S_θ): For a threshold θ, all variables whose predicted backbone class confidence reaches at least θ are selected:

$$S_\theta = \{i \mid \max(p_{i,\text{B0}}, p_{i,\text{B1}}) \geq \theta\}. \tag{4.7}$$

Value Assignment (v_i): Each selected variable $x_i \in S_\theta$ is assigned the value associated with its most probable backbone class:

$$v_i = \begin{cases} 0 & \text{if } p_{i,\text{B0}} \geq p_{i,\text{B1}}, \\ 1 & \text{otherwise.} \end{cases} \tag{4.8}$$

Adaptive Tolerance (Δ): The parameter Δ represents the number of incorrect predictions allowed when enforcing assignments. This parameter directly influences the size of the trust region; an excessively small Δ risks the exclusion of optimal solutions, while an excessively large Δ may fail to restrict the search space effectively.

In order to create a trust region as constrained as possible without losing optimal solutions, the parameter Δ must be set equal to the number of errors made in the selected predictions. In this way, the solver will be able to ignore all errors without increasing the size of the trust region more than necessarily.

Since the exact number of errors made by the model is not known during inference, it is estimated using the model's confidence. The expected number of errors E is defined as the aggregate uncertainty across all selected variables:

$$E = \sum_{i \in S_\theta} \left(1 - \max(p_{i,\text{B0}}, p_{i,\text{B1}})\right). \tag{4.9}$$

Because E may systematically deviate from the true error count when evaluating instances outside the training distribution, the hyperparameter $\alpha \in [-1, 1]$ is used to adjust Δ relative to E and the size of the selected set $|S_\theta|$:

$$\Delta = \begin{cases} \lceil E \cdot (1 + \alpha) \rceil, & \text{if } \alpha \leq 0, \\ \lceil (|S_\theta| - E) \cdot \alpha + E \rceil, & \text{if } \alpha > 0. \end{cases} \tag{4.10}$$

This formulation allows for the following strategic interpolations:

- $\alpha = -1 \implies \Delta = 0$: The model trusts all predicted assignments with no room for error.
- $\alpha = 0 \implies \Delta = \lceil E \rceil$: The model relies directly on the estimated number of errors.
- $\alpha = 1 \implies \Delta = |S_\theta|$: All assignments may be violated, effectively removing search space restrictions.

Notably, the hyperparameters θ and α are **size-invariant**, meaning they remain consistent and applicable across a distribution of instances with varying scales. In contrast, the parameters k_0, k_1, and Δ are inherently size-dependent, as their values are derived directly from the total number of variables in a specific instance. This distinction is critical for the model's generalization capabilities, as size-invariant parameters allow for zero-shot transfer between problems of different magnitudes, whereas size-dependent parameters must be recalibrated or scaled to match the instance dimensions.

Chapter 5

Empirical Evaluation

This chapter presents a comprehensive empirical evaluation of the proposed BackPaS framework. A series of experiments was conducted to assess the impact of the core innovations introduced in this work: backbone-based predictions, the literal-based bipartite graph representation, the specialized GNN architecture, and the novel parameterization of the search step. The evaluation primarily utilizes the primal integral metric to compare the performance of different methods, supplemented by standard machine learning metrics, such as Cross-Entropy to analyze the predictive performance of the model.

5.1 BackPaS Variants

To isolate and evaluate the impact of the specific design choices underpinning BackPaS, several intermediate variants are introduced that incrementally incorporate the proposed modifications. This methodical approach allows for the attribution of performance changes to individual components, including the prediction task redesign, the proposed GNN architecture, and the adaptive trust region parameterization.

- **BackPaS-V0: Backbone Prediction Only.** This foundational variant maintains the original GNN architecture and bipartite graph representation utilized by ConPaS and earlier PaS implementations, as well as their fixed trust region parameters (k_0, k_1, Δ). The sole modification is the replacement of the original regression output with the proposed *three-class classification* scheme. Each binary variable x_i is classified into one of three backbone classes: B0 (fixed to 0), B1 (fixed to 1), or NB (not in the backbone). Consequently, the trust regions are constructed by selecting the top k_0 variables with the highest $p_{i,\mathrm{B0}}$ scores and the top k_1 variables with the highest $p_{i,\mathrm{B1}}$ scores.
- **BackPaS-Net: Incorporating the GNN Architecture.** This variant builds upon BackPaS-V0 by integrating the proposed *specialized GNN architecture*. This architecture features eight Bipartite Graph Transformer layers

and operates on the literal-based bipartite graph representation of the PBO instances. All other components remain identical to those in BackPaS-V0.

- **BackPaS-Param: Incorporating Adaptive Parameters.** This variant also builds upon BackPaS-V0 but, instead of the architectural changes, it incorporates the proposed *adaptive trust region parametrization* defined by the parameters (θ, α). This allows the search space adjustment to dynamically respond to the GNN's prediction confidence, replacing the fixed parameters (k_0, k_1, Δ) utilized in BackPaS-V0.
- **BackPaS: Full Model.** This represents the complete and final model, incorporating *all proposed modifications*: the backbone classification task, the specialized GNN architecture (as in BackPaS-Net), and the adaptive trust region parametrization (as in BackPaS-Param).

5.2 Baselines

Two primary baselines are utilized to evaluate the effectiveness of the proposed framework. The first consists of a direct application of the Gurobi solver to the original, unmodified problem instances. This baseline, referred to as Gurobi, serves to determine whether the BackPaS pipeline actually enhances solver performance or if the introduction of additional trust-region constraints inadvertently degrades search efficiency.

The second baseline is ConPaS [34], which represents the current state-of-the-art in Predict-and-Search implementations. While ConPaS was primarily designed for general Mixed-Integer Linear Programming (MILP) instances, its methodology is directly applicable to PBO instances. Utilizing ConPaS as a baseline allows for a direct comparison between the proposed backbone-based classification approach and existing contrastive-learning-based prediction strategies.

5.3 Computational Environment

Hardware: For CPU-intensive tasks, such as the extraction of ConPaS solutions, backbone extraction with GuroBack, and baseline instance solving with Gurobi, a computing cluster was used with nodes equipped with Intel Xeon E5-2670 v3 processors and 64 GB of RAM. For GPU-intensive tasks, including network training and inference for instance modification, an NVIDIA A40 GPU with 48 GB of VRAM was employed. Both hardware configurations operated under Rocky Linux 8.

Software: The experimental framework utilized Gurobi v10.0.0rc2, PyTorch v2.7.1 (built with CUDA 12.8), and PyTorch Geometric v2.6.1. The code for GuroBack and backpas is publicly avaibale at the links below:

- GuroBack: `https://github.com/bryan-alvarado-ulloa/guroback`
- BackPaS: `https://github.com/bryan-alvarado-ulloa/backpas`

Resource Allocation: For backbone extraction with GuroBack and instance solving with Gurobi, instances were processed in parallel with up to 12 concurrent executions. Each instance was allocated 5 GB of RAM and a single CPU core. For ConPaS solution extraction, up to 12 instances were processed in parallel using single cores without explicit RAM limits.

5.4 Datasets and Benchmarks

Bench	Partitions	Characteristics	Origin
MIS	`train/val-backpas`	1000 nodes, 4 avg. degree	D-MIPLIB
MIS	`train/val-conpas`, `val`, `test`	6000 nodes, 5 avg. degree	Generated
CA	`train/val-backpas`	300 items, 600 bids	Generated
CA	`train/val-conpas`, `val`, `test`	2000 items, 4000 bids	D-MIPLIB
MVC	`train/val-backpas`	1200 nodes, 5 avg. degree	D-MIPLIB
MVC	`train/val-conpas`, `val`, `test`	6000 nodes, 5 avg. degree	Generated

Table 5.1: Benchmarks (Bench) and partitions utilized, including their primary characteristics and origins. MIS and MVC instances are based on Barabási–Albert graphs, and CA instances follow the Arbitrary distribution [42]. The partitions used for training BackPaS are smaller than those used for ConPaS and for the `test` sets.

Table 5.1 summarizes the details of each benchmark and partition utilized in this work. Three PBO benchmarks are employed: Maximum Independent Set (MIS), Minimum Vertex Cover (MVC), and Combinatorial Auctions (CA). these match the ConPaS [34] benchmarks, with the exception of the Item Placement benchmark, which contains non-binary variables.

Due to the computational costs associated with backbone extraction, BackPaS requires smaller training instances than ConPaS, as the latter trains on test-sized instances without the need to compute backbones. The following partitions are defined for MIS, CA, and MVC:

- `train-backpas/val-backpas`: The BackPaS GNN was trained on 800 instances and validated on 100 instances, all of which utilized pre-computed backbones. The model checkpoint corresponding to the lowest validation Cross-Entropy Loss was selected for inference.

- `train-conpas/val-conpas`: The ConPaS GNN was trained on 400 instances and validated on 100 instances, using the positive and negative labels obtained through the ConPaS methodology. The model checkpoint with the lowest validation Contrastive Loss was selected for inference.
- `val`: A set of 100 instances used for hyperparameter tuning for BackPaS (α, θ) and BackPaS-V0 (k_0, k_1, Δ).
- `test`: A set of 100 instances used for the final evaluation of the primal integral.

The partitions `val-conpas` and `val` are identical and are distinguished only by their specific application in the experiments. D-MIPLIB [36] datasets were utilized where available; otherwise, instances were generated using D-MIPLIB generators (MVC, CA) and the GeCo library (MIS).

For BackPaS, the extraction of backbone labels from the `train-backpas` and `val-backpas` partitions required approximately 55 CPU hours for CA, 279 for MIS, and 12 for MVC. Conversely, ConPaS obtains positive training samples by solving instances with Gurobi using a 1-hour timeout. As none of these instances were solved within the limit, the extraction of the training signal for ConPaS required at least 500 CPU hours per benchmark.

5.5 Evaluation Metrics

5.5.1 Pipeline Performance (Primal Integral):

To evaluate the overall performance of the complete pipeline we use the Primal Integral (see Section 2.1.2), with a timelimit of 1000 seconds. We defined the best known solution $\mathbf{v_{opt}}$ as the one with the best result across all tested methods within 1000 seconds. When reporting this metric, the time required to run the GNN and construct the trust regions is excluded for all methods.

5.5.2 GNN Prediction Performance (Cross-Entropy):

The divergence between the predicted class distribution and the ground-truth labels is evaluated using the Mean Cross-Entropy (MCE) (see Equation 4.6). This metric ensures that performance is weighted equally across instances regardless of their size.

5.6 Hyperparameter Tuning

The trust region parameters (k_0, k_1, Δ for BackPaS-V0, or θ and α for BackPaS) were fine-tuned using Bayesian Optimization with Tree-structured Parzen Estimators (TPE) [10]. This process aimed to minimize the primal integral at 1000 seconds

over the `val` partitions. The optimization began with a random sample of 30 points in the hyperparameter space, followed by three sequential batches of 10 points each suggested by the TPE method. For ConPaS, the hyperparameter values reported in the original paper were utilized to maintain consistency. Tables 5.2 and 5.3 summarize the hyperparameters used in the `test` partitions of the CA, MIS, and MVC benchmarks for each method.

Benchmark	BackPaS	BackPaS-Param
CA	$(0.8231211, -0.7828316)$	$(0.6294576, 0.8672775)$
MIS	$(0.6161128, -0.9946887)$	$(0.6071678, -0.7594742)$
MVC	$(0.9958419, -0.9282533)$	$(0.9470682, -0.0458712)$

Table 5.2: Hyperparameters (θ, α) used by BackPaS and BackPaS-Param on the `test` partitions of CA, MIS and MVC benchmarks.

Benchmark	BackPaS-V0	BackPaS-Net	ConPaS
CA	$(826, 1824, 1378)$	$(1063, 839, 1449)$	$(1500, 0, 0)$
MIS	$(1091, 591, 53)$	$(2180, 935, 23)$	$(500, 500, 10)$
MVC	$(1087, 1414, 296)$	$(728, 2286, 2)$	$(500, 100, 15)$

Table 5.3: Hyperparameters (k_0, k_1, Δ) used by BackPaS-V0, BackPaS-Net and ConPaS on the `test` partitions of CA, MIS and MVC benchmarks.

Additionally, for the experiment of generalization on MIS (See section 5.7.3), the selected parameters for BackPaS are $(\theta, \alpha) = 0.62370877, -0.84192583$.

5.7 Experimental Results

The performance of BackPaS is evaluated through a series of comparative experiments. Subsection 5.7.1 benchmarks the method against previous PaS implementations and ablation variants. Subsection 5.7.2 details the effect of architectural choices on the GNN. Finally, Subsection 5.7.3 investigates the robustness of BackPaS when applied to the MIS problem over varying graph distributions.

5.7.1 Performance Evaluation of BackPaS and Variants

Table 5.4 compares the anytime performance of the different methods across the three domains. The baseline is represented by Gurobi (solving the original instances without augmentation) and ConPaS (the ConPaS-LQ variant).

Table 5.4: **Anytime Performance (Primal Integral) Comparison on Test Benchmarks.** Mean Primal Integral (PI) values and standard deviations (in parentheses) are reported for the CA, MIS, and MVC test partitions. The ranks provided are determined by pairwise Wilcoxon signed-rank tests ($p < 0.05$). Lower PI values indicate superior anytime performance.

Method	CA	MIS	MVC
Gurobi	20.74 (6.08) 2^{nd}	44.09 (7.73) 6^{th}	7.79 (1.15) 6^{th}
ConPaS	19.50 (7.36) 2^{nd}	8.93 (1.94) 5^{th}	6.83 (2.53) 5^{th}
BackPaS	**16.09** (6.79) 1^{st}	**0.82** (0.53) 1^{st}	**0.49** (0.50) 1^{st}
BackPaS-Param	30.13 (9.54) 4^{th}	4.09 (1.44) 4^{th}	1.50 (0.45) 2^{nd}
BackPaS-Net	24.05 (10.22) 3^{rd}	1.13 (0.17) 2^{nd}	1.74 (0.49) 3^{rd}
BackPaS-V0	17.11 (6.79) 1^{st}	2.31 (0.61) 3^{rd}	2.20 (0.44) 4^{th}

A systematic evaluation of the proposed components—Backbones, the Network Architecture (Net), and the Search Parameters (Param)—indicates that they are strongly complementary. The fully integrated BackPaS model achieves the lowest PI across all benchmarks and is statistically superior to every other variant. Notably, the BackPaS-V0 variant, using only extracted backbones, already outperforms both Gurobi and ConPaS on all benchmarks.

The addition of the proposed architecture yields substantial gains over the backbone-only foundation. On the MIS benchmark, the PI decreases from 2.31 (BackPaS-V0) to 1.13 (BackPaS-Net), representing a reduction of 51%. Similarly, on MVC, the PI drops from 2.20 to 1.74 (21%). While the search parameters alone (BackPaS-Param) can be detrimental on CA and MIS, the largest overall gains occur when they are combined with the new architecture. In the full BackPaS model, the PI on MIS reaches 0.82, an 80% improvement over BackPaS-Param. This suggests that the specialized parameters realize their full potential only when coupled with the more accurate predictions provided by the new architecture.

5.7.2 Performance of Alternative GNN Architectures

A comparative study was conducted to analyze the effect of specific architectural design choices:

Graph Layer Type: Graph Convolutional Layers (GCN) from original PaS implementations were compared with the proposed Graph Transformer Layers (GTR).

Network Depth: The effect of layer counts (2, 4, 6, and 8) on backbone prediction was investigated.

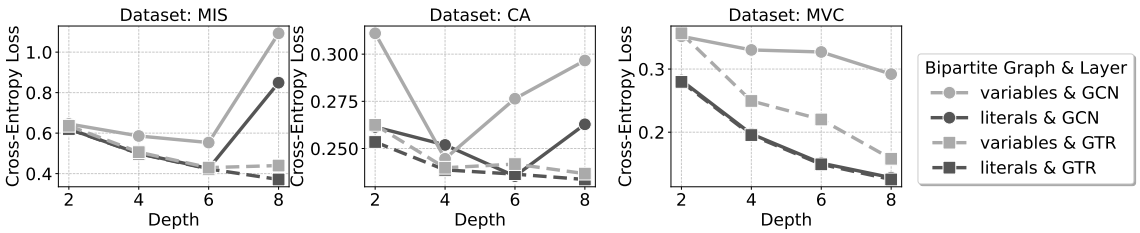


Figure 5.1: Evaluation of alternative BackPaS architectures: Cross-Entropy Loss on the validation set (`val-backpas`) for MIS, CA, and MVC benchmarks. Performance is shown by network depth (number of layers), comparing Graph Convolutional (GCN) vs. Graph Transformer (GTR) layers, and variable-based vs. literal-based graph representations. Lower values are better.

Bipartite Graph Representation: The standard variable-based representation was contrasted with the proposed literal-based representation.

Figure 5.1 illustrates the Mean Cross-Entropy Loss for each configuration. Models utilizing GTR layers and literal-based representations consistently achieve lower loss across all datasets. Furthermore, deeper networks generally led to lower loss, with the 8-layer GTR architecture combined with literal-based bipartite graphs emerging as the best-performing configuration.

5.7.3 Generalization on MIS

To determine if BackPaS can generalize to larger instances while maintaining or increasing graph density, an experiment was conducted using Barabási–Albert graphs generated with the GeCo library. The partitions and their characteristics are detailed in Table 5.5.

Benchmark	Partitions	Characteristics
MIS-666	`train`, `val`	666 nodes, 2 avg. degree
MIS-1000	`train`, `val`	1000 nodes, 3 avg. degree
MIS-1333	`train`, `val`	1333 nodes, 4 avg. degree
MIS-1666	`val`, `test`	1666 nodes, 5 avg. degree
MIS-2000-6	`val`, `test`	2000 nodes, 6 avg. degree
MIS-2000-10	`val`, `test`	2000 nodes, 10 avg. degree

Table 5.5: Partitions for MIS generalization experiments. All instances are BA graphs.

The model was trained on MIS-666, MIS-1000, and MIS-1333. Hyperparameters were tuned using the `val` partitions of all benchmarks. The results on the larger test partitions are summarized in Table 5.6.

Method	MIS-1666	MIS-2000-6	MIS-2000-10
Gurobi	3.81 (1.58)	14.94 (3.57)	32.64 (6.56)
BackPaS	**0.04** (0.17)	**0.19** (0.25)	-

Table 5.6: Primal Integral results for MIS generalization. Mean and Standard Deviations (in parentheses) are reported. Lower values are better. The missing entry indicates that BackPaS did not provide a trust region.

BackPaS demonstrates strong generalization on instances where density remains consistent relative to the training set. On MIS-1666 and MIS-2000-6, it significantly outperforms Gurobi. However, the method failed to generalize to the MIS-2000-10 benchmark. As shown by the missing entry in Table 5.6, the model could not generate predictions as it failed to identify a backbone; predicted probabilities were not certain enough to meet the selection threshold. This suggests a limitation in handling simultaneous increases in both graph size and density significantly beyond the training regime.

Chapter 6

Conclusions and Future Work

This thesis presented BackPaS, a novel iteration of the Predict-and-Search (PaS) framework specifically engineered to enhance the resolution of Pseudo-Boolean Optimization (PBO) problems. The primary contribution of this research lies in its multifaceted architectural design, which integrates four key innovations: 1) a learning paradigm based on *backbone-based supervision*; 2) a specialized Graph Neural Network (GNN) architecture utilizing *Graph Transformer layers*; 3) a *literal-based bipartite graph representation*; and 4) an *adaptive trust region construction* that dynamically adjusts to prediction confidence independently of instance scale.

6.1 Summary of Findings and Hypothesis Validation

The experimental results obtained through a systematic ablation study demonstrate that the proposed components are highly complementary. The fully integrated BackPaS model achieved the lowest Primal Integral (PI) values across all evaluated benchmarks, establishing statistical superiority over both traditional baselines and contemporary state-of-the-art methods.

A critical finding of this work is the efficacy of backbone-based supervision. Even the foundational variant, BackPaS-V0, which relies solely on backbone predictions without the advanced architectural refinements, significantly outperformed the commercial solver Gurobi and the leading PaS implementation, ConPaS. Furthermore, the adoption of literal-based graph representations and Graph Transformer layers was shown to be essential for minimizing prediction loss, thereby providing the high-fidelity guidance necessary for the search process.

In light of these results, it is concluded that the **Hypothesis** of this research has been **validated**. The empirical evidence confirms that predicting backbones in PBO problems using a GNN, when integrated within a PaS framework, yields a measurable and significant improvement in the primal integral metric compared to previous PaS-based approaches.

6.2 Achievement of Objectives

The general objective of developing a novel approach for PBO that leverages GNNs to predict backbones within a PaS framework has been successfully fulfilled. This success is supported by the completion of the following specific objectives:

1. **Backbone Extraction:** A robust tool for PBO instances was designed and implemented, enabling the generation of high-quality labels essential for supervised learning.
2. **GNN Development:** A Graph Neural Network capable of accurately predicting backbone variables was developed, optimized through the introduction of Graph Transformer layers and literal-based bipartite graphs.
3. **Framework Integration:** The GNN-generated predictions were successfully integrated into an adaptive search framework, allowing the solver to utilize neural recommendations through an intelligent trust-region mechanism.
4. **Rigorous Evaluation:** The BackPaS framework was benchmarked against competitive solvers and learning-based methods across MIS, CA, and MVC problem sets, confirming its superior performance in terms of the primal integral metric.

Ultimately, this work reinforces the design principle that the synergy between high-quality machine learning predictions and adaptive search guidance is a viable and powerful path toward solving complex combinatorial problems.

6.3 Future Work

Future research directions will involve extending the BackPaS methodology to broader classes of combinatorial optimization problems beyond the Pseudo-Boolean domain. Additionally, there is significant potential in developing sophisticated weighting schemes for backbone variables. By quantifying the relative importance of specific variables to the solver's overall success, the framework could further refine its search guidance and improve runtime efficiency in increasingly constrained environments.

Bibliography

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] T. Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, Jul 2009.

[3] T. Achterberg, T. Berthold, and G. Hendel. Rounding and propagation heuristics for mixed integer programming. In *Operations Research Proceedings 2011: Selected Papers of the International Conference on Operations Research (OR 2011), August 30-September 2, 2011, Zurich, Switzerland*, pages 71–76. Springer, 2012.

[4] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97, Jan 2002.

[5] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. Generic ilp versus specialized 0-1 ilp: An update. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 450–457, 2002.

[6] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[7] R. Asín-Achá, A. Espinoza, O. Goldschmidt, D. S. Hochbaum, and I. I. Huerta. Selecting fast algorithms for the capacitated vehicle routing problem with machine learning techniques. *Networks*, 84(4):465–480, 2024.

[8] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[9] Y. Bengio, A. Lodi, and A. Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[10] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[11] 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.

[12] 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.

[13] A. Biere, N. Froleyks, and W. Wang. Cadiback: Extracting backbones with cadical. In *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[14] 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.

[15] E. Boros and P. L. Hammer. Pseudo-boolean optimization. *Discrete applied mathematics*, 123(1-3):155–225, 2002.

[16] A. Bouras, M. A. Ghaleb, U. S. Suryahatmaja, and A. M. Salem. The airport gate assignment problem: a survey. *The scientific world journal*, 2014(1):923859, 2014.

[17] J. Cai, T. Huang, and B. Dilkina. Learning backdoors for mixed integer programs with contrastive learning. *arXiv preprint arXiv:2401.10467*, 2024.

[18] M. C. Chen, S. N. Sze, S. L. Goh, N. R. Sabar, and G. Kendall. A survey of university course timetabling problem: Perspectives, trends and opportunities. *IEEE Access*, 9:106515–106529, 2021.

[19] 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.

[20] Y. Crama and P. L. Hammer. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, 2011.

[21] S. De Vries and R. V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on computing*, 15(3):284–309, 2003.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[23] 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.

[24] O. Dubois and G. Dequen. A backbone-search heuristic for efficient solving of hard 3-sat formulae. In *IJCAI*, volume 1, pages 248–253, 2001.

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

[26] A. Ferber, B. Wilder, B. Dilkina, and M. Tambe. Mipaal: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1504–1511, 2020.

[27] M. Francia-Carramiñana, B. Alvarado-Ulloa, D. S. Hochbaum, R. Ñanculef, and R. J. Asín-Achá. Backbones in pseudo-boolean optimization: Extraction and analysis. In *18th International Conference on Agents and Artificial Intelligence (ICAART)*, 2026. Accepted for publication.

[28] M. Gasse, D. Chetelat, N. Ferroni, L. Charlin, and A. Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[29] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

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

[31] Q. Han, L. Yang, Q. Chen, X. Zhou, D. Zhang, A. Wang, R. Sun, and X. Luo. A gnn-guided predict-and-search framework for mixed-integer linear programming. In *The Eleventh International Conference on Learning Representations*, 2023.

[32] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[33] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[34] T. Huang, A. M. Ferber, A. Zharmagambetov, Y. Tian, and B. Dilkina. Contrastive predict-and-search for mixed integer linear programs. In *Forty-first International Conference on Machine Learning*, 2024.

[35] T. Huang, J. Li, S. Koenig, and B. Dilkina. Anytime multi-agent path finding via machine learning-guided large neighborhood search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9368–9376, 2022.

[36] W. Huang, T. Huang, A. M. Ferber, and B. Dilkina. Distributional miplib: a multi-domain library for advancing ml-guided milp methods, 2024.

[37] I. I. Huerta, D. A. Neira, D. A. Ortega, V. Varas, J. Godoy, and R. J. As'in Ach'a. Improving the state-of-the-art in the traveling salesman problem: An anytime automatic algorithm selection. *Expert Syst. Appl.*, 187:115948, 2022.

[38] M. Janota, I. Lynce, and J. Marques-Silva. Algorithms for computing backbones of propositional formulae. *Ai Communications*, 28(2):161–177, 2015.

[39] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[41] G. Lederman, M. N. Rabe, and S. A. Seshia. Learning heuristics for automated reasoning through deep reinforcement learning. *arXiv preprint arXiv:1807.08058*, 57, 2018.

[42] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 66–76, 2000.

[43] Z. Li, Q. Chen, and V. Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

[44] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki. Learning rate based branching heuristic for sat solvers. In *Theory and Applications of Satisfiability Testing–SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings 19*, pages 123–140. Springer, 2016.

[45] J. P. Marques-Silva and K. A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on computers*, 48(5):506–521, 2002.

[46] J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, 1(1):65–77, 2002.

[47] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic 'phase transitions'. *Nature*, 400(6740):133–137, 1999.

[48] R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and R. Zhao. Speeding up pseudo-boolean propagation. In *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[49] C. H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[50] 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.

[51] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[52] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[53] S. J. Russell and P. Norvig. *Artificial intelligence a modern approach*. London, 2010.

[54] D. Selsam and N. Bjørner. Guiding high-performance sat solvers with unsat-core predictions. In *Theory and Applications of Satisfiability Testing–SAT 2019: 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9–12, 2019, Proceedings 22*, pages 336–353. Springer, 2019.

[55] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[56] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[57] J. Slaney, T. Walsh, et al. Backbones in optimization and approximation. In *IJCAI*, pages 254–259, 2001.

[58] P. Toth and D. Vigo. An overview of vehicle routing problems. *The vehicle routing problem*, pages 1–26, 2002.

[59] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[60] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[61] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[62] W. Wang, Y. Hu, M. Tiwari, S. Khurshid, K. L. McMillan, and R. Miikkulainen. Neuroback: Improving CDCL SAT solving using graph neural networks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024.

[63] H. Xiong, S. Shi, D. Ren, and J. Hu. A survey of job shop scheduling problem: The types and models. *Computers Operations Research*, 142:105731, 2022.