

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA  
DEPARTAMENTO DE INFORMÁTICA  
SANTIAGO - CHILE



## “EXTRACTION AND PREDICTION OF BACKBONES IN PSEUDO-BOOLEAN PROBLEMS”

BRYAN JESÚS ALVARADO ULLOA

MEMORIA PARA OPTAR AL TÍTULO DE  
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Roberto Asín  
Profesor Correferente: Ricardo Ñanculef

August - 2024

## ABSTRACT

**Abstract**— The challenge of identifying backbones in Pseudo-Boolean (PB) problems is addressed, where backbones are variables that maintain fixed values across all optimal solutions in optimization problems, or across all feasible solutions in decisional problems. A novel method for both extracting and predicting these backbones is introduced. The methodology involves a three-step pipeline: (1) converting PB instances into SAT problems, (2) using SAT solvers to extract the backbone, and (3) mapping the extracted backbone back to the original PB variables. Additionally, a predictive model based on Graph Neural Networks (GNNs) is developed to estimate the likelihood of variable membership in the backbone. Experimental results demonstrate the effectiveness of this approach, highlighting its potential to uncover the underlying structure of PB problems and improve the efficiency of solving these complex optimization tasks.

**Keywords**— Pseudo-Boolean optimization, backbone extraction, SAT solvers, Graph Neural Networks, variable prediction, optimization problems

# Contents

|                                                    |     |
|----------------------------------------------------|-----|
| ABSTRACT                                           | i   |
| INDEX OF FIGURES                                   | iii |
| INDEX OF TABLES                                    | iii |
| INTRODUCTION                                       | 1   |
| CHAPTER 1: PROBLEM DEFINITION                      | 2   |
| CHAPTER 2: RELATED WORK                            | 4   |
| CHAPTER 3: PRELIMINARIES                           | 5   |
| 3.1 Decision and Optimization models . . . . .     | 5   |
| 3.2 Graph Representation of These Models . . . . . | 6   |
| 3.3 Graph Neural Networks (GNNs) . . . . .         | 7   |
| 3.3.1 Machine Learning and Deep Learning . . . . . | 7   |
| 3.3.2 Graph Neural Networks (GNNs) . . . . .       | 8   |
| 3.4 Backbones . . . . .                            | 9   |
| CHAPTER 4: BACKBONE EXTRACTION IN PB               | 11  |
| 4.1 NaPS . . . . .                                 | 11  |
| 4.2 CadiBack . . . . .                             | 12  |
| 4.3 Backbone Mapping . . . . .                     | 12  |
| 4.4 Algorithm for Backbone Extraction . . . . .    | 13  |
| CHAPTER 5: BACKBONE PREDICTION IN PB               | 14  |
| 5.1 Instance Representation and Encoding . . . . . | 14  |
| 5.2 Network Architecture . . . . .                 | 15  |
| CHAPTER 6: EXPERIMENTS AND RESULTS                 | 17  |
| 6.1 Experiments . . . . .                          | 17  |
| 6.1.1 Dataset . . . . .                            | 17  |
| 6.1.2 Backbone Extraction . . . . .                | 17  |
| 6.1.3 Backbone Prediction . . . . .                | 18  |
| 6.1.4 Training and evaluation . . . . .            | 19  |
| 6.2 Results . . . . .                              | 19  |
| 6.2.1 Extracting Backbones . . . . .               | 19  |
| 6.2.2 Predicting Backbones . . . . .               | 21  |
| CHAPTER 7: CONCLUSIONS                             | 23  |
| REFERENCES                                         | 24  |

## List of Figures

|   |                                                                                                                                                                                                                              |    |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1 | Bipartite graph representation of a problem. The circular nodes represent variables, and the rectangular nodes represent constraints or clauses. The edges can be weighted to represent coefficients in constraints. . . . . | 7  |
| 2 | Network Architecture . . . . .                                                                                                                                                                                               | 16 |
| 3 | Results of the backbone extraction pipeline. . . . .                                                                                                                                                                         | 20 |
| 4 | Distribution of the relative size of the backbones found in the entire dataset. . . . .                                                                                                                                      | 20 |
| 5 | Distribution of the relative size of the backbones found in PBO and PBS instances respectively. . . . .                                                                                                                      | 21 |
| 6 | Evolution of the Loss, AP and NB-MR metric during cross-validation. . . . .                                                                                                                                                  | 21 |

## List of Tables

|   |                                                                      |    |
|---|----------------------------------------------------------------------|----|
| 1 | Node and Edge Embeddings . . . . .                                   | 15 |
| 2 | Summary of metrics with their means and standard deviations. . . . . | 22 |

## INTRODUCTION

Pseudo-Boolean (PB) problems are a class of combinatorial problems where the goal is to find assignments to binary variables that satisfy all constraints. When an objective function is involved, the aim is to find assignments that not only satisfy all constraints but also achieve the minimum or maximum possible value for the objective function. These problems are prevalent in various fields, including computer science, operations research, theoretical physics, and manufacturing. (Boros and Hammer, 2002). Despite advances in solving PB problems, extracting meaningful information about the problem structure, such as backbones, remains a complex and challenging task.

A backbone in the context of PB problems refers to a set of variables that are fixed to a particular value across all optimal (or feasible, if it is not an optimization problem) solutions of the problem. Identifying these variables can provide valuable insights into the problem's structure and can be used to enhance the efficiency of solution methods. However, specialized tools for extracting backbones from PB problems do not currently exist.

To address this gap, a novel approach for backbone extraction from PB problems is proposed by leveraging existing tools developed for Boolean satisfiability (SAT) problems. The approach involves a three-step pipeline: (1) converting PB instances into SAT problems, (2) extracting backbones using SAT-based methods, and (3) mapping the extracted backbones back to the original PB variables. This method utilizes well-established SAT solvers and backbone extraction techniques to tackle PB problems.

In addition to backbone extraction, a predictive model based on Graph Neural Networks (GNNs) is introduced to estimate the membership of variables in the backbone. This model aims to predict which variables are part of the backbone without determining their specific values. Although the current model only predicts membership, it offers a promising direction for enhancing PB solvers and guiding backbone extraction processes.

The organization of the text is as follows: CHAPTER 1 introduces the problem and the objectives of this work. CHAPTER 2 reviews related work in backbone extraction and prediction, as well as examples of the usage of Machine Learning Models to enhance MILP and SAT solvers. CHAPTER 3 introduces and defines the principal concepts of MILP, SAT, and PB along with their graph representation, and also introduces GNNs. CHAPTER 4 and CHAPTER 5 describe the proposed pipeline and the GNN-based predictive model, respectively. The description of the experiments conducted, and their results, are presented in CHAPTER 6. Finally, a conclusion and a discussion of future work is provided in CHAPTER 7.

## CHAPTER 1

### PROBLEM DEFINITION

The problem is situated within the domain of Pseudo-Boolean (PB) problems, a specialized area of computer science focused on optimization and decision problems where both the objective function and constraints are represented as multi-linear functions over Boolean variables. This domain is significant across various fields, including computer science, operations research, theoretical physics, and manufacturing. (Boros and Hammer, 2002).

Currently, there is a lack of detailed studies or tools specifically for extracting backbones in PB instances. While there has been a lot of research on SAT problems, with algorithms like CadiBack (Biere et al., 2023) used for backbone extraction, these approaches don't directly work for PB instances. This highlights the need for effective backbone extraction methods tailored to PB problems. Additionally, there aren't advanced predictive models that can accurately identify backbone variables based on exact data. Existing methods tend to rely on solutions close to the optimum (Huang et al., 2024), but they are limited by approximation errors and don't use precise backbone information. Developing a machine learning model that can predict backbone variables based on exact data could greatly improve optimization by making accurate predictions, which would guide solvers and reduce the computational load. This model could also be integrated into the extraction process to enhance the performance of backbone extractors.

Key stakeholders in this field include researchers and professionals working on optimization and decision problems involving PB instances. This includes academic researchers, industry professionals, and organizations using optimization techniques and constraint programming to solve complex issues. Additionally, developers of optimization software and tools would benefit significantly from improved methods for backbone prediction and extraction.

Without effective backbone extraction tools for PB instances, it becomes difficult to accurately identify critical variables, leading to suboptimal solutions and inefficiencies in solving complex optimization problems. Optimization algorithms might have to rely on less accurate approximations or heuristics, which would lower their performance and increase computational costs. Moreover, the lack of advanced methods for backbone extraction and prediction could slow down progress in PB optimization research, making it harder for researchers and practitioners to develop or improve techniques. This would negatively impact real-world applications, such as resource allocation, scheduling, and decision-making. Lastly, failing to develop predictive models and specialized tools would mean missing out on the benefits of advancements in machine learning and optimization techniques, potentially stalling the development of more effective solutions for PB problems.

**General Objective** To develop a comprehensive approach for backbone extraction and prediction in PB instances.

**Specific Objectives**

- Develop a pipeline for backbone extraction from PB instances.
- Design and implement a machine learning model to predict backbone variables in PB instances.
- Evaluate the performance of the machine learning model.

## CHAPTER 2

### RELATED WORK

This work is closely related to Cadiback (Biere et al., 2023) and NeuroBack (Wang et al., 2021), which focus on computing and predicting backbones for SAT problems. Cadiback utilizes a method to compute exact backbones, whereas NeuroBack integrates Graph Neural Networks (GNNs) with Conflict-Driven Clause Learning (CDCL) to predict the phases of variables that frequently appear in satisfying assignments. NeuroBack’s approach reduces the need for frequent online model inferences, allowing the model to operate solely on the CPU.

In contrast, the focus of this work is on Pseudo-Boolean (PB) instances, which differ from SAT problems. In the domain of Mixed Integer Linear Programming (MILP), machine learning models have been employed to predict variable values, leading to the concept of a “pseudo-backbone”—a subset of variables with stable values near the optimum (Huang et al., 2024). The GNN-guided predict-and-search framework (Han et al., 2023) enhances MILP solvers by combining GNNs with optimization techniques to identify high-quality solutions. The Contrastive Predict-and-Search (ConPaS) (Huang et al., 2024) method further refines this by using contrastive learning to predict solutions, leveraging high-quality solutions as positive examples and low-quality or infeasible ones as negatives.

Additionally, other approaches have been undertaken to leverage GNNs and Machine Learning in combinatorial optimization. For instance, in MILP, the work of (Gasse et al., 2019) was a pioneer in using Graph Convolutional Neural Networks (GCNNs), training a GCNN to predict the next branching variable based on offline imitation learning from existing solver decisions. This approach demonstrated superior performance compared to expert-designed branching rules in the SCIP solver. In SAT problems, the use of GNNs has also been of interest; for example, NeuroSAT (Selsam et al., 2018) trains a GNN to create a SAT solver capable of making decisions given sufficient iterations.

All these approaches typically use bipartite graph representations, where constraints and variables are nodes connected by edges. NeuroBack add to this graph a meta node that is connected to all constraint nodes, while NeuroSAT replaces variable nodes with literal nodes and adds additional edges. Other methods adhere to traditional bipartite graph representations, focusing on different aspects of problem-solving and prediction.

## CHAPTER 3

### PRELIMINARIES

#### 3.1 Decision and Optimization models

**Satisfiability Problem (SAT)** The Satisfiability Problem (SAT) involves determining whether there exists an assignment of truth values to variables that satisfies a given Boolean formula in conjunctive normal form (CNF). Formally, a SAT problem can be defined as follows:

- **Literal:** A literal is either a variable or its negation. For example, if  $x$  is a variable, then  $x$  and  $\neg x$  are literals.
- **Formula:** A Boolean formula composed of  $n$  variables  $x_1, x_2, \dots, x_n$  in CNF.

$$\bigwedge_{i=1}^m \left( \bigvee_{j=1}^{k_i} l_{ij} \right)$$

where  $l_{ij}$  are literals (variables or their negations).

- **Output:** Determine whether there exists an assignment of  $x_i \in \{0, 1\}$  that makes the formula true.

**Mixed-Integer Linear Programming (MILP)** Is an optimization problem that involves both continuous and integer (including binary) variables. Formally, a MILP problem can be defined as follows:

- **Objective Function:** Minimize (or maximize) a linear function of continuous and integer variables.

$$\min \sum_{i=1}^n c_i x_i + \sum_{j=1}^p d_j y_j$$

where  $x_i$  are continuous variables,  $y_j$  are integer variables, and  $c_i, d_j$  are given coefficients.

- **Constraints:** Subject to a set of linear inequalities involving both types of variables.

$$\sum_{i=1}^n a_{ik} x_i + \sum_{j=1}^p b_{jk} y_j \leq e_k, \quad k = 1, \dots, q$$

where  $a_{ik}, b_{jk}, e_k$  are given coefficients.

**Pseudo-Boolean (PB)** A Pseudo-Boolean problem lies between SAT and MILP. It involves binary variables and constraints that are expressed as multi-linear polynomials, which may include both equalities and inequalities involving these variables (Boros and Hammer, 2002). Formally, a PB problem can be defined as follows:

- **Objective Function (optional):** Minimize (or maximize) a pseudo-Boolean function.

$$\min \sum_{S \subseteq \{1, \dots, n\}} c_S \prod_{i \in S} x_i$$

where  $x_i \in \{0, 1\}$  and  $c_S$  are given coefficients associated with subsets  $S$  of the variable indices.

- **Constraints:** Subject to a set of pseudo-Boolean inequalities.

$$\sum_{S \subseteq \{1, \dots, n\}} a_S \prod_{i \in S} x_i \geq b_j, \quad j = 1, \dots, m$$

where  $a_S$  and  $b_j$  are given coefficients.

When the PB problem includes an objective function, it is referred to as a Pseudo-Boolean Optimization problem (PBO). The goal in a PBO is to find an assignment of values to the variables that achieves the minimum (or maximum) of the objective function while satisfying all constraints. If the PB problem does not have an objective function, it is referred to as a Pseudo-Boolean Satisfaction problem (PBS)

### 3.2 Graph Representation of These Models

Generally, these models can be represented using **bipartite graphs**, where one set of nodes represents the variables and the other set represents constraints or clauses. The edges in the graph indicate the involvement of variables in the constraints or clauses. The figure 1 is an example of a general graph representation of a problem.

- **Nodes:**
  - **Variable Nodes:** Represent the variables in the problem. These can be Boolean variables ( $x_i \in \{0, 1\}$ ) for SAT and PB, or a mix of continuous and integer variables for MILP.
  - **Constraint Nodes:** Represent the constraints or clauses in the problem.
- **Edges:** An edge exists between a variable node and a constraint node if the variable appears in the constraint or clause.

The specific characteristics of the graphs for each model are as follows:

- **SAT:**
  - **Constraint Nodes:** Represent the clauses.
  - **Edges:** Unweighted, indicating the presence of a variable (or its negation) in a clause.
- **MILP and PB:**
  - **Constraint Nodes:** Represent the constraints.
  - **Edges:** Weighted, with weights representing the coefficients in the linear constraints.

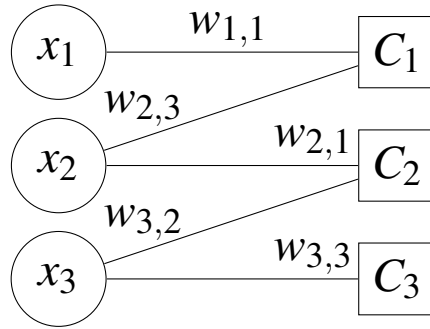


Figure 1: Bipartite graph representation of a problem. The circular nodes represent variables, and the rectangular nodes represent constraints or clauses. The edges can be weighted to represent coefficients in constraints.

### 3.3 Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are a type of neural network designed to work directly with graph structures. GNNs have proven to be effective for various tasks involving graph data, such as node classification, link prediction, and graph classification. To understand GNNs, it is helpful to first have a brief introduction to machine learning and deep learning.

#### 3.3.1 Machine Learning and Deep Learning

**Machine Learning (ML)** is a branch of artificial intelligence that focuses on developing algorithms that can learn from and make predictions or decisions based on data. Machine learning aims to create techniques and algorithms that allow a computer to learn, meaning

that its performance in a task improves as it gains experience, which is obtained from data (Russell and Norvig, 2010). Depending on the characteristics of the data used by the algorithm and its objective, machine learning can be classified into various paradigms, among which the following stand out:

- **Supervised Learning:** The data used consists of pairs, which contain examples of the inputs the algorithm is expected to receive and their corresponding outputs that it must learn. In this way, the algorithm tries to learn the relationship between these two elements from many examples.
- **Unsupervised Learning:** The data used does not contain the expected outputs, so the algorithm must learn solely from the inputs.
- **Reinforcement Learning:** The algorithm is interpreted as an agent that interacts with an environment. The environment provides the input to the algorithm and reacts to its outputs. Depending on the actions (outputs) determined by the algorithm, the environment provides a measure of reward. The goal is to maximize future rewards.

**Deep Learning (DL)** is a subset of machine learning that uses neural networks with many layers (hence “deep”) to model complex patterns in data. DL has achieved significant success in various domains, such as computer vision (Krizhevsky et al., 2012; He et al., 2016), natural language processing (Vaswani et al., 2017; Devlin et al., 2018) speech recognition (Schneider et al., 2019) and even multimodal tasks that can process multiple inputs (Achiam et al., 2023).

### 3.3.2 Graph Neural Networks (GNNs)

GNNs extend the capabilities of traditional neural networks to handle graph-structured data. In a graph, data is represented as nodes (vertices) and edges (connections between nodes), which can naturally model relationships and interactions in many real-world systems (Wu et al., 2020).

The key idea of GNNs is to iteratively update the representation of each node by aggregating information from its neighbors. This process is often referred to as “message passing” or “neighborhood aggregation”.

**General Framework of GNNs** The general framework of GNNs involves the following steps:

- **Initialization:** Each node  $v$  in the graph is initialized with a feature vector  $h_v^{(0)}$ .
- **Message Passing:** For each iteration  $t$ , each node  $v$  aggregates information from its neighbors  $\mathcal{N}(v)$ :

$$m_v^{(t+1)} = \text{AGGREGATE} \left( \{h_u^{(t)} : u \in \mathcal{N}(v)\} \right)$$

where AGGREGATE is a function that combines the features of the neighboring nodes (e.g., sum, mean, max).

- **Update:** The node  $v$  updates its feature vector based on the aggregated information and its current feature vector:

$$h_v^{(t+1)} = \text{UPDATE}(h_v^{(t)}, m_v^{(t+1)})$$

where UPDATE is a function that combines the node's current feature vector with the aggregated message (e.g., a neural network layer).

- **Output:** After  $T$  iterations, the final node representations  $h_v^{(T)}$  can be used for various tasks, such as node classification, graph classification, or link prediction.

### 3.4 Backbones

In the context of SAT (Satisfiability) problems, the **backbone** refers to a subset of variables in a CNF (Conjunctive Normal Form) formula that have a consistent truth value across all satisfying assignments.

Formally, let  $I$  be a SAT problem instance with CNF formula  $\phi$ , and let  $\mathcal{S}$  denote the set of all satisfying assignments for  $\phi$ .

The **backbone set**  $B$  of the problem instance  $I$  consists of pairs  $(x_i, v)$  where  $x_i$  is a variable and  $v$  is the value assigned to  $x_i$ , such that:

$$B = \{(x_i, v) \mid \forall \sigma \in \mathcal{S}, \sigma(x_i) = v\}$$

Here  $\sigma$  corresponds to a satisfying assignment for the CNF formula while  $\sigma(x_i)$  corresponds to the value of the variable  $x_i$  in such assignment.

Note that this can be easily extended to PBO, PBS and MILP problems.

- **Extension for PBS:**  $\mathcal{S}$  is defined as the set of *feasible* solutions of the PBS instance  $I$
- **Extension for PBO and MILP:**  $\mathcal{S}$  is defined as the set of *optimal* solutions of the PBO (or MILP) instance  $I$

Having a non-empty backbone set  $B$  implies that a valid assignment exists for the subset of variables included in  $B$  which does not remove any optimal, feasible, or satisfiable solution of the original instance  $I$ . This feature can be used to simplify the instance and potentially improve the performance of solvers. Approximating the backbone can also be used to design

heuristics for these solvers, which can help find a solution more efficiently. Additionally, predicting only the membership of a variable in the backbone set (without determining the specific value) can aid in improving algorithms that compute the backbones.

## CHAPTER 4

### BACKBONE EXTRACTION IN PB

Currently, no specialized tool exists for directly extracting backbones from Pseudo-Boolean (PB) instances. To address this, a pipeline has been developed that leverages existing SAT-based methods. The process is outlined below:

1. **Conversion to SAT:** Convert the PB instance into a SAT problem using the NaPS tool (Sakai and Nabeshima, 2015).
2. **Backbone Extraction:** Extract the backbone variables from the SAT problem using the CadiBack algorithm (Biere et al., 2023).
3. **Backbone Mapping:** Map the extracted backbone variables from the SAT problem back to the original PB variables to obtain the PB backbone.

For PBO instances, an additional step is required before conversion: solving the instance with a PB solver (e.g., RoundingSAT (Elffers and Nordström, 2018)) to find the optimal value, which is then added as a constraint to convert the PBO instance into a PBS instance. If the instance is infeasible, no backbone exists, and the process terminates. Also, if the PB instance is not linear, a previous step of linearization is also performed.

#### 4.1 NaPS

NaPS employs Reduced Ordered Binary Decision Diagrams (ROBDDs) to convert Pseudo-Boolean (PB) problems into SAT problems.

**ROBDD Overview** A Reduced Ordered Binary Decision Diagram (ROBDD) is a compact data structure for representing Boolean functions. It is *ordered* (variables follow a fixed sequence) and *reduced* (removes redundant nodes), ensuring a minimal representation of Boolean logic.

##### Transformation Process

1. **Boolean Function to ROBDD:** NaPS converts the Boolean function of the PB problem into an ROBDD.

2. **ROBDD to CNF Conversion:** The ROBDD is translated into Conjunctive Normal Form (CNF) clauses.
3. **SAT Problem Formation:** The CNF clauses are used to formulate a SAT problem that can be solved with standard SAT solvers.

## 4.2 CadiBack

CadiBack is an advanced backbone extraction algorithm that builds on the CaDiCaL SAT solver. The key idea behind CadiBack is to iteratively identify backbone variables based on their role in satisfying a CNF formula:

**Backbone Extraction Process** Once a satisfiable solution of a CNF formula is obtained, CadiBack tests each variable by solving a new CNF formula that includes an additional clause forcing a particular variable to take a different value from the one in the satisfiable solution. If the new formula remains SAT, the variable is not in the backbone. If the new formula becomes UNSAT, the variable is confirmed to be in the backbone.

**Optimizations** CadiBack incorporates several optimizations to enhance performance:

- **Chunk Processing:** Process variables in chunks to test their status as backbones.
- **Clause Updates:** Update the CNF formula and adjust chunk sizes based on satisfiability.
- **Dynamic Adjustment:** Use dynamic chunk sizes and formula rotations to optimize backbone identification.
- **Flippability Check:** Detect literals that are not backbones by evaluating different SAT solutions to the original problem where the same variables have different values.

## 4.3 Backbone Mapping

The mapping of backbones from SAT instances to PB instances is performed as follows:

1. **Obtain Variable Mapping:** NaPS provides a dictionary mapping SAT variable names to their corresponding PB variable names.
2. **Filter SAT Backbone:** Remove variables from the SAT backbone that are not part of the original PB instance.

3. **Translate Names:** Using the mapping dictionary, translate the names of the remaining SAT backbone variables to their original PB names.

## 4.4 Algorithm for Backbone Extraction

The pseudocode for the complete process is shown in Algorithm 1.

```
Input: PB instance  $I$   
Output: PB Backbone or None  
 $I = \text{Linearize}(I);$   
 $[\text{feasible}, \text{solution}] = \text{roundingSAT}(I);$   
if feasible is False then  
| return None  
end  
if  $\text{IsPBOInstance}(I)$  then  
|  $I \leftarrow \text{ConvertToPBS}(I, \text{solution});$   
end  
 $[\text{satInstance}, \text{varsMappingDictionary}] = \text{naps}(I);$   
 $\text{satBackbone} = \text{cadiback}(\text{satInstance});$   
 $\text{pbBackbone} = \text{Map}(\text{satBackbone}, \text{varsMappingDictionary});$   
return  $\text{pbBackbone};$ 
```

**Algorithm 1:** Backbone Extraction for PB Instances

## CHAPTER 5

### BACKBONE PREDICTION IN PB

In this section, a method is introduced for ranking the variables of a PB instance such that the variables that are part of the backbone are ranked higher. The approach utilizes a graph neural network (GNN) to process the bipartite graph representation of the PB instance. The network is trained to classify variable nodes, distinguishing those that are part of the backbone from those that are not. Finally, the model's predictions are used to rank the variables.

It is important to note that this method does not predict the specific values of the variables identified as part of the backbone. Although the network could be trained to perform this task, preliminary experiments indicate that predicting the values is significantly more complex, and this challenge is reserved for future work.

Despite only predicting the membership of variables in the backbone, this model is still valuable. It can be employed to design branching heuristics that optimize PB solvers, such as Gurobi (Gurobi Optimization, LLC, 2023) or RoundingSAT (Elffers and Nordström, 2018). Additionally, it can be used to guide backbone extractors, thereby improving their performance.

#### 5.1 Instance Representation and Encoding

The approach used for instance representation and encoding is inspired by the work presented in (Han et al., 2023), which provided a baseline for the embeddings of nodes and edges in the graph representation. However, several key differences are introduced in the method. The instance is represented with literal nodes instead of variable nodes, similar to the approach used in (Selsam et al., 2018). Additionally, unweighted edges are included between each pair of literals of the same variable.

The embeddings for each node and edge are based on the baseline paper (Han et al., 2023), with minor modifications to better suit the approach. The PB instance is normalized before calculating these embeddings, ensuring that all coefficients are positive and on the left side of the equations. The details of the node and edge embeddings are provided in Table 1.

The objective function of a PBO instance is represented as a constraint with the rhs and sense embeddings set as 0.

| Name                        | Meaning                                      |
|-----------------------------|----------------------------------------------|
| <b>Literal Embedding</b>    |                                              |
| obj                         | Normalized coefficient in objective function |
| l_coeff                     | Avg. coefficient in all constraints          |
| Nl_coeff                    | Degree in bipartite graph (literal)          |
| max_coeff                   | Max coefficient value                        |
| min_coeff                   | Min coefficient value                        |
| Nv_coeff                    | Degree in bipartite graph (variable)         |
| <b>Constraint Embedding</b> |                                              |
| c_coeff                     | Avg. of all coefficients in the constraint   |
| Nc_coeff                    | Degree in bipartite graph (constraint)       |
| rhs                         | Right-hand-side value                        |
| sense                       | Constraint sense                             |
| <b>Edge Embedding</b>       |                                              |
| coeff                       | Coefficient of literal in constraint         |

Table 1: Node and Edge Embeddings

## 5.2 Network Architecture

The neural network architecture used in this work builds upon the approach detailed in (Han et al., 2023). The architecture consists of several stages designed to effectively process and leverage the embeddings of the bipartite graph representation.

**Embedding Block** Initially, the embeddings of the nodes and edges are transformed to a common dimensional space. Specifically, the embeddings for literals (6 dimensions), constraints (4 dimensions), and edges (1 dimension) are mapped to a unified dimension of 64 through embedding layers. This standardization ensures compatibility and facilitates subsequent processing.

**Graph Convolution Block** The network then applies two rounds of graph convolution. In the first round, a **Message Passaging Layer** (MPL) updates the constraint nodes based on their connections to the literal nodes. In the second round, a second MPL updates the literal nodes using the newly updated constraint nodes and the edges. This bidirectional propagation ensures that information flows effectively between constraints and literals, enhancing feature representation.

**Literals Message Block** Following the graph convolutions, the **Literals Message Block** applies a linear transformation to each literal node. This transformation incorporates both the literal’s embedding and that of its negation. Mathematically, for a given literal node  $l_i$

with embedding  $e_i$  and its negation  $\neg l_i$  with embedding  $e_{\neg i}$ , the updated embedding  $e'_i$  is computed as:

$$e'_i = \alpha \cdot e_i + \beta \cdot e_{\neg i}$$

where  $\alpha$  and  $\beta$  are learnable coefficients.

The network then performs another round of **Graph Convolution Block** followed by a second **Literals Message Block**. This repetition allows for further refinement of the embeddings through additional feature propagation and messaging.

**Concatenation of literals** In the final step, the embeddings of each literal are concatenated with those of its corresponding negation, such that every variable is now represented with a unique embedding.

**Feedforward Block** The concatenated vector is then passed through two fully connected layers. The first layer has 128 neurons, and the final layer outputs a single value per variable. This value is used to predict whether a variable is part of the backbone.

The full architecture can be seen in the Figure 2.

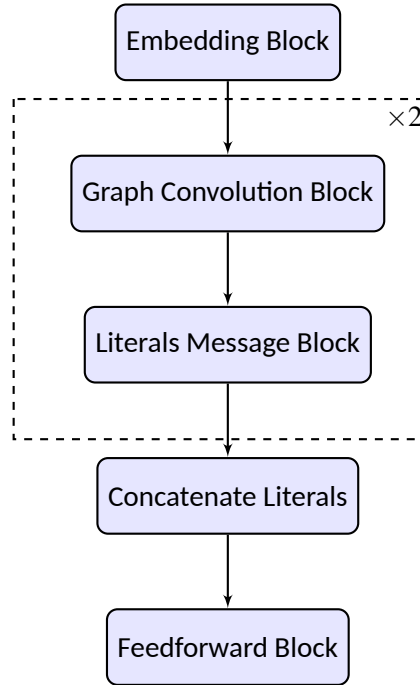


Figure 2: Network Architecture

## CHAPTER 6

### EXPERIMENTS AND RESULTS

#### 6.1 Experiments

This section outlines the experiments conducted, including details on the dataset and hardware architecture used for backbone extraction and prediction.

##### 6.1.1 Dataset

The dataset used in this study is from the Pseudo-Boolean Competition 2016 (Roussel, 2016). It consists of 5,427 instances that span both optimization and constraint satisfaction problems. This diverse collection was part of the SAT 2016 conference and provides a broad range of pseudo-Boolean problem instances for benchmarking purposes.

The dataset is structured to include:

- PBO problems, which seek to find the best solution according to a given objective function.
- PBS problems, which focus on finding a solution that satisfies all specified constraints.

This comprehensive dataset is used to evaluate the effectiveness of different solvers and methods in handling a wide variety of pseudo-Boolean problems. For more details, refer to (Roussel, 2016).

Due to computational resource limitations, the results and the linearized instances obtained in (Pezo Vergara, 2023) are used. This with the goal to determine which instances are feasible and which are not, thereby eliminating the non-feasible instances from the dataset (since these instances are known to lack backbones). Only the solutions provided by RoundingSAT were available, resulting in a total of 2708 feasible instances with solutions found. These are the instances that will be used in the experiments.

##### 6.1.2 Backbone Extraction

The backbone extraction experiments were conducted on a server with an Intel Xeon E5-2670 v3 processor, featuring 24 cores and 64 GB of RAM. For these experiments, Algorithm 1 was executed using precomputed solutions from RoundingSAT, as provided in (Pezo Vergara,

2023). The versions of NaPS and Cadiback used were 1.02b5 and 0.2.1, respectively. Also Cadiback was compiled to use CaDiCaL 2.0.0. On the server, only the steps involving NaPS, Cadiback, and the variable mapping were executed. To manage computational resources effectively, up to 5 instances were processed in parallel, each with a 2-hour time limit. Instances that did not complete within this timeframe were terminated.

### 6.1.3 Backbone Prediction

The extracted backbones served as the dataset for training the neural network. The following metrics were used to evaluate the performance of the model.

**Average Precision (AP) Metric** The primary evaluation metric used is Average Precision (AP), which is particularly suited for tasks where the correct prediction of high-confidence backbones is critical. The AP metric emphasizes predictions with higher confidence, penalizing errors more severely when they occur among top-ranked predictions (Moffat and Zobel, 2008).

Given a sorted truth vector  $\mathbf{p}$ , where elements are ordered by the model's confidence scores (e.g.,  $\mathbf{p} = [1, 0, 0, 1, 0, 1]$ ), the precision at position  $k$  is defined as the proportion of correct backbone predictions within the top  $k$  elements:

$$\text{precision}_k(\mathbf{p}) = \frac{1}{k} \sum_{i=1}^k \mathbf{p}_i$$

where  $\mathbf{p}_i$  is the truth value at position  $i$ .

The AP is then computed as the mean precision at all positions  $k$ , but only for positions where the prediction is correct:

$$\text{AP}(\mathbf{p}) = \frac{1}{b} \sum_{k=1}^n \mathbf{p}_k \cdot \text{precision}_k(\mathbf{p})$$

where  $b$  is the total number of backbone variables, and  $n$  is the total number of variables.

**Non-Backbone Mean Rank (NB-MR) Metric** Alongside the Average Precision (AP) metric, we utilize the Non-Backbone Mean Rank (NB-MR) metric, which is computed as follows:

$$\text{NB-MR}(\mathbf{p}) = \frac{\min\{k : p_k = 0\}}{n}$$

If all the variables of the instance belongs to the backbone (i.e: there is no  $k$  for which  $p_k = 0$ ) then we consider NB-MR as 1.

This metric assesses the position of the first non-backbone variable in the sorted list of predictions, effectively capturing how many correct backbone variables are predicted before the first non-backbone variable appears. A higher NB-MR value indicates a greater proportion of correct predictions at the top of the list.

#### 6.1.4 Training and evaluation

The backbone predictor model was trained using 5-fold cross-validation on the complete set of instance-backbone pairs. Within each fold, 20% of the instances were reserved for in-fold validation to select the best-performing model across epochs. The training process spanned 150 epochs, utilizing binary cross-entropy loss and the AdamW optimizer. The learning rate was set at 0.001, with a batch size of 1. Gradient accumulation was employed, updating model parameters after every 32 steps.

For completeness, we also report the Accuracy and F1 Score metrics. The metrics AP and F1 were computed only for instances with a backbone relative size that is neither 0% nor 100%. In other words, they were calculated for instances where variables are present in both classes: `isInBackbone` and `isNotInBackbone`. Additionally, metrics for a random predictor and a perfect predictor are reported. The random predictor metrics were calculated by assigning random probabilities to each variable before ranking, and this process was repeated 100 times across the entire dataset used by the GNN model.

## 6.2 Results

This section presents the results of the experiments conducted.

### 6.2.1 Extracting Backbones

Figure 3 shows the results of the backbone extraction procedure. As illustrated, backbones were successfully extracted from the majority of instances (68.32%). Among the instances where the method failed, the primary reason was due to the time limit constraint.

From the backbones successfully extracted, Figure 4 shows the distribution of their sizes, defined as the cardinality of the backbone set divided by the total number of variables. Two peaks are evident: one at 0 and the other at 100.

When separating the PBS and PBO instances (see Figure 5), the peak at 100 is associated

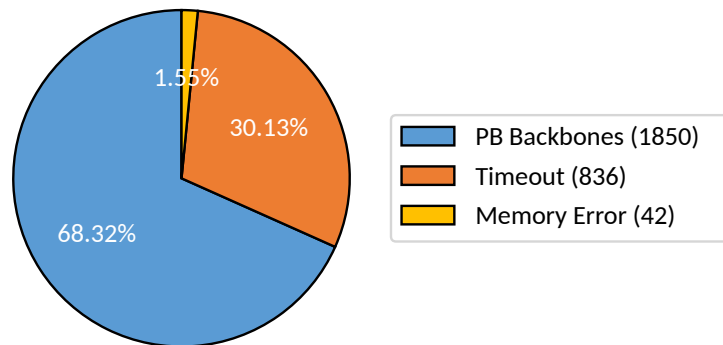


Figure 3: Results of the backbone extraction pipeline.

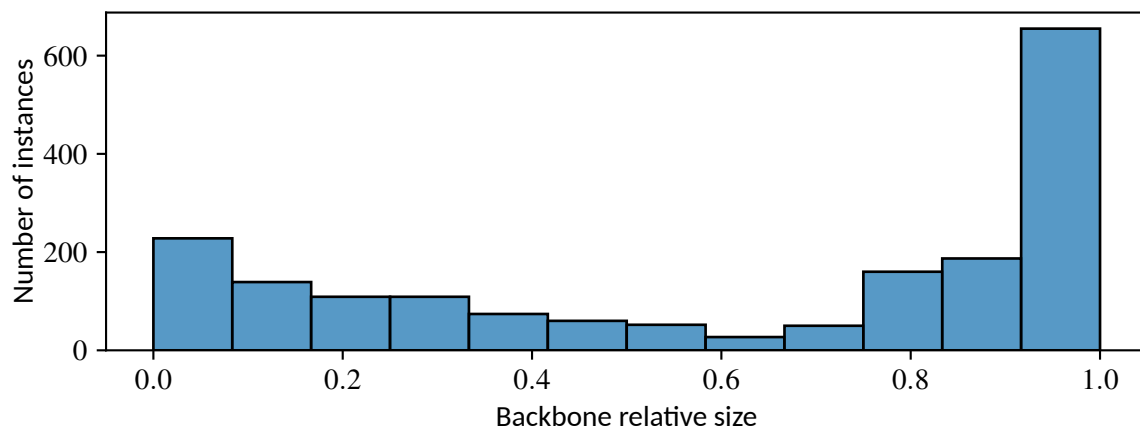


Figure 4: Distribution of the relative size of the backbones found in the entire dataset.

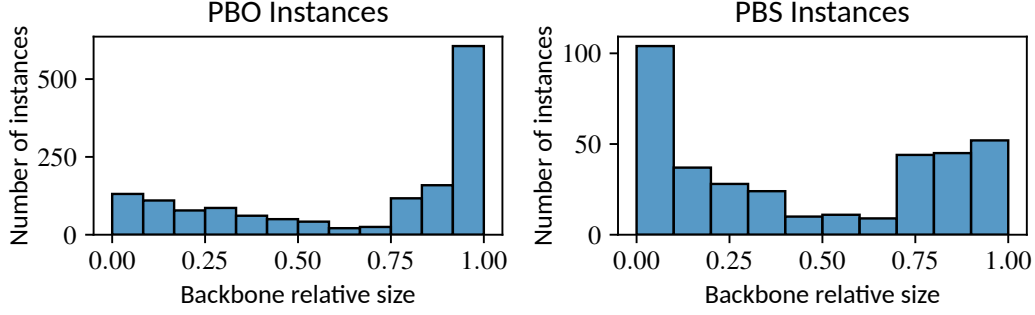


Figure 5: Distribution of the relative size of the backbones found in PBO and PBS instances respectively.

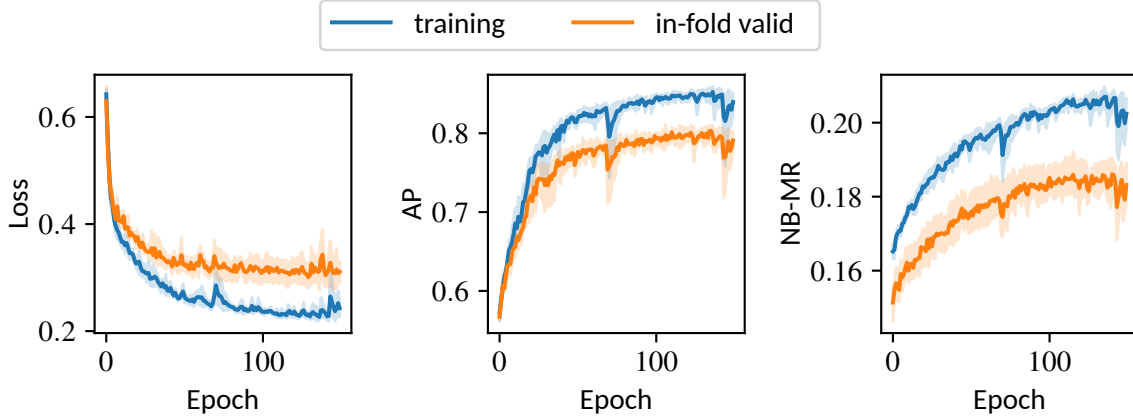


Figure 6: Evolution of the Loss, AP and NB-MR metric during cross-validation.

with the PBO instances, indicating that many optimization problems have a unique optimal solution. On the other hand, the plot of the PBS instances shows the peak at 0, suggesting that these instances have very disjoint feasible solutions.

### 6.2.2 Predicting Backbones

Figure 6 shows the evolution of metrics during the training process. The plotted lines represent the mean across all folds, while the shaded bands correspond to a 95% confidence interval.

The results of the selected models, based on the in-fold selection policy, are summarized in Table 2. For comparison, the performance of a random and a perfect predictor is also included. The model demonstrates a high Average Precision (AP) of  $0.8228 \pm 0.0150$ , significantly outperforming the random predictor by 0.3003. Additionally, the model

| <b>Metric</b> | <b>GNN</b>          | <b>Random</b>       | <b>Perfect</b> |
|---------------|---------------------|---------------------|----------------|
| AP            | $0.8228 \pm 0.0150$ | $0.5225 \pm 0.0009$ | 1              |
| NB-MR         | $0.1947 \pm 0.0059$ | $0.1453 \pm 0.0003$ | 0.3151         |
| Accuracy      | $0.8723 \pm 0.0109$ | $0.6301 \pm 0.0000$ | 1              |
| F1 Score      | $0.7172 \pm 0.0429$ | $0.6126 \pm 0.0000$ | 1              |

Table 2: Summary of metrics with their means and standard deviations.

achieves a Non-Backbone Mean Rank (NB-MR) of  $0.1947 \pm 0.0059$ , indicating that, on average, nearly the top 20% of the ranked variables are correctly identified as backbones without errors. Furthermore, the GNN model consistently outperforms the random predictor across all metrics, demonstrating its effectiveness in accurately ranking and predicting backbone variables.

## CHAPTER 7

### CONCLUSIONS

A comprehensive approach to backbone extraction and prediction for Pseudo-Boolean (PB) instances is presented. A pipeline was developed to address the absence of specialized tools for backbone extraction from PB problems. This pipeline involves converting PB instances into SAT problems, extracting backbones using existing SAT-based methods, and mapping the results back to the original PB variables. The effectiveness of this approach was demonstrated through an experiment on a diverse dataset from the Pseudo-Boolean Competition 2016, where the method successfully extracted backbones from 68.32% of the instances.

A novel method for predicting backbone variables using a Graph Neural Network (GNN) was also introduced. The model predicts the membership of variables in the backbone without determining their specific values. Despite this limitation, the model shows promise in enhancing PB solvers and guiding backbone extractors. The neural network achieved an Average Precision  $0.8228 \pm 0.0134$  and a Non-Backbone Mean Rank (NB-MR) of  $0.1947 \pm 0.0059$ , indicating that it can accurately identify nearly 20% of the backbone variables without error.

While the results are promising, several avenues for future research can be explored:

- **Value Prediction:** The current approach focuses on predicting the membership of variables in the backbone. Future work should explore methods for predicting the specific values of these variables, which poses a more complex challenge but could provide more detailed insights.
- **Optimization of Network Architecture:** The current GNN architecture is based on a specific design and parameterization. Future research could investigate alternative network architectures, hyperparameter settings, and advanced techniques such as attention mechanisms to further improve performance.
- **Scalability and Efficiency:** The backbone extraction process, especially for large instances, can be computationally intensive. Developing more scalable algorithms and optimizing existing tools to handle larger datasets efficiently will be crucial for practical applications.
- **Integration with Other Solvers:** While the method can be used to guide existing PB solvers, integrating it with other state-of-the-art solvers and optimization frameworks could lead to more effective and versatile solutions.

## REFERENCES

- Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Alteschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Biere, A., Froleys, N., and Wang, W. (2023). Cadiback: Extracting backbones with cadical. In *26th International Conference on Theory and Applications of Satisfiability Testing (SAT 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.
- Boros, E. and Hammer, P. L. (2002). Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1):155–225.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Elffers, J. and Nordström, J. (2018). Divide and conquer: Towards faster pseudo-boolean solving. In *IJCAI*, volume 18, pages 1291–1299.
- Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32.
- Gurobi Optimization, LLC (2023). Gurobi Optimizer Reference Manual.
- Han, Q., Yang, L., Chen, Q., Zhou, X., Zhang, D., Wang, A., Sun, R., and Luo, X. (2023). A gnn-guided predict-and-search framework for mixed-integer linear programming. *arXiv preprint arXiv:2302.05636*.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- Huang, T., Ferber, A. M., Zharmagambetov, A., Tian, Y., and Dilkina, B. (2024). Contrastive predict-and-search for mixed integer linear programs. In *Forty-first International Conference on Machine Learning*.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.
- Moffat, A. and Zobel, J. (2008). Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. Inf. Syst.*, 27(1).
- Pezo Vergara, C. (2023). Anytime automatic algorithm selection for the pseudo-boolean optimization problem.

- Roussel, O. (2016). Pseudo boolean competition 2016. <https://www.cril.univ-artois.fr/PB16/>. Accessed: 2024-08-10.
- Russell, S. J. and Norvig, P. (2010). *Artificial intelligence a modern approach*. London.
- Sakai, M. and Nabeshima, H. (2015). Construction of an robdd for a pb-constraint in band form and related techniques for pb-solvers. *IEICE TRANSACTIONS on Information and Systems*, 98(6):1121–1127.
- Schneider, S., Baevski, A., Collobert, R., and Auli, M. (2019). wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*.
- Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. (2018). Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- Wang, W., Hu, Y., Tiwari, M., Khurshid, S., McMillan, K., and Mikkilainen, R. (2021). Neuroback: Improving cdcl sat solving using graph neural networks. *arXiv preprint arXiv:2110.14053*.
- Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24.